Datenbanken

Was ist ein Datenbanksystem?

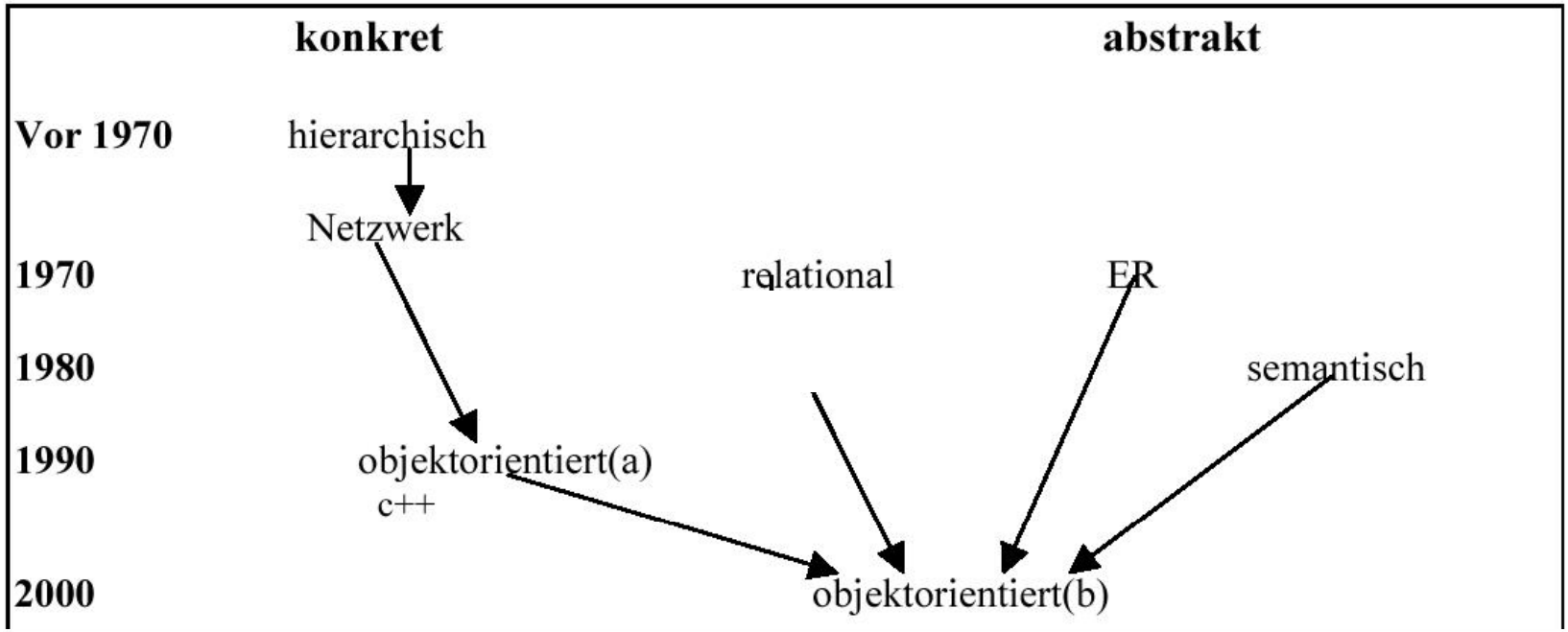
- Rechnergestütztes Informationssystem
- Stellt Daten allgemein einer Vielzahl von Anwendungen zur Verfügung
- Effizientes Hilfsmittel zur Erzeugung, Manipulation und Verwaltung großer Datenmengen

- Beispiele:
  - Uni-Bibliothek
  - Internet-Auktionshaus eBay

Historische Entwicklung

- 1960
  Dateiverwaltung
- 1970
  Hirarchische Datenbanken
- 1975
  Netzwerk-Datenbanken, Entity-Relationship-Diagramm,
  Drei Ebenen Architektur
- 1980
  Relationale Datenbanken
- 1985
  Normierung der Abfragesprache SQL, Standard-Datenbanken
  (dBase, Oracle, ...)
- 1990
  Objektorientierte Datenbanksysteme
- 1995
  Relational-objektorientierte Datenbanksysteme,
  verteilte Datenbanksysteme

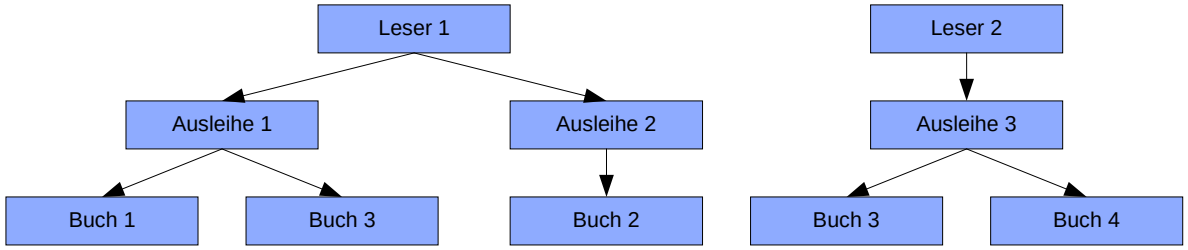
Historische Entwicklung



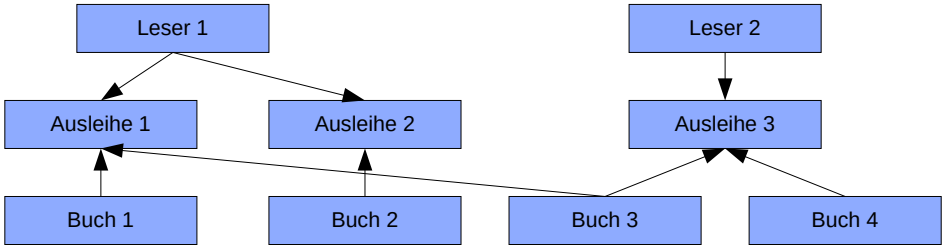
Synonyme	Klasse	Attribut	Objekt	Beziehung
Hierarchisch	Recordtyp	Feld	Record	Parent-child Relation
Netzwerkorient.	Recordtyp	Attribut	Record	Set
Relation. Algebra	Relation	Attribut	Tupel	(Join)
ERM	Entitytyp	Attribut	Entity	Relation
Relational	Tabelle	Spalte	Zeile	(Join)
Objektorientiert	Klasse	Attribut	Objekt	Vererbung, Assoziation, Aggregation

Datenmodelle

- Hierarchisches Datenmodell
  - **Datenbeschreibung durch Bäume**
  - **Knoten = Datensätze, Kanten = Beziehungen zueinander**

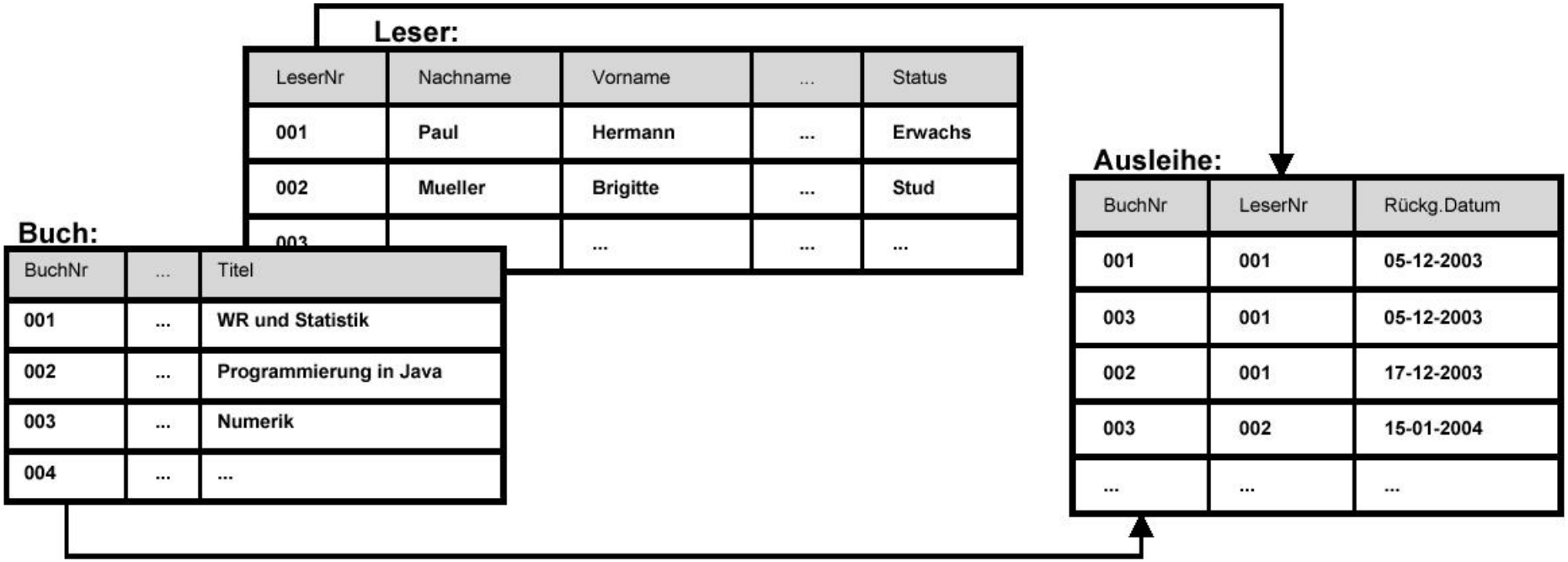


- Netzwerk-Modell
  - **Datenbeschreibung durch Graphen**
  - **Vernetzung der Dateien untereinander**



Relationale Datenbanksysteme

- Datendarstellung in Form von Tabellen
- Starke Unabhängigkeit der Daten durch deutliche Trennung zwischen logischem und physischem Datenmodell



Dateisystem

- Ein Dateisystem besteht aus:
  - einer oder mehreren Dateien und
  - einem oder mehreren Anwenderprogrammen
- Existenz einer festen Zuordnung zwischen den Daten und damit arbeitenden Programmen
- „Wissen“, das ein Anwenderprogramm haben muss
  - Dateiformat in dem die Daten abgespeichert sind,
    z.B. Textdatei, die zum Lesen und/oder Schreiben zu öffnen ist
  - Spezielle Gestalt des Datenformats
    => wie sind die Daten innerhalb der Datei aufgebaut
  - Randbedingungen zur Kontrolle auf gültige Datensätze

Vorteile und Grenzen von Dateisystemen

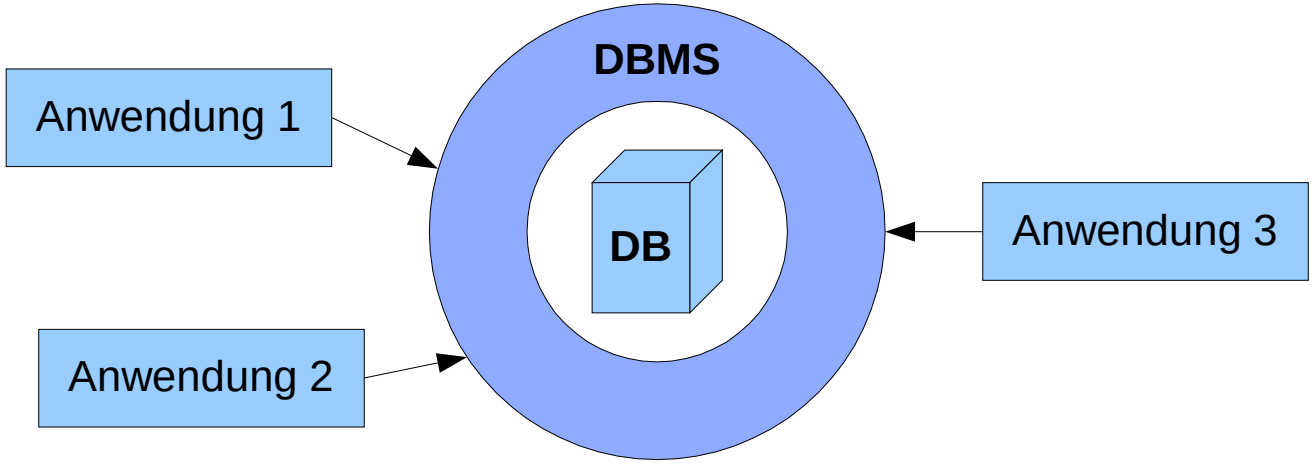
- Vorteile eines Dateisystems
  - Zugriffssoftware und Datenverarbeitung sehr spezifisch
  - Gut einsetzbar bei geringen Datenmengen und unkomplizierten Zugriffsarten mit einfachen Suchfunktionen
  - Software exakt zugeschnitten => Vermeidung des Black-Box-Effektes
  - Schnelle Durchführung einfacher Abfragen und Operationen möglich
- Probleme und Grenzen beim Einsatz von Dateisystemen
  - Hohes Maß an Redundanz
  - Gefahr der Inkonsistenz
  - Inflexibilität des Systems
    (hoher Aufwand bei Änderungen, hohe Kosten)
  - Fehleranfälligkeit und großer Programmieraufwand bei komplexen Abfragen und Funktionen
  - Probleme im Bereich der Datensicherheit und des Datenschutzes
  - Kaum einheitlicher Standard durchsetzbar
  - Mehrbenutzerbetrieb nicht immer möglich. Zugriff auf eine bereits geöffnete Datei kann – je nach System – nicht möglich sein

Aufbau eines Datenbanksystems

- Datenbanksystem (DBS):
  Bestandteil eines Informationssystems (IS)
  - Durchführung eines Informationsaustausches
- Komponenten eines Informationssystems:
  - Daten,
  - Datenbank-Software,
  - Rechner-Hardware,
  - Personal und Anwendungssoftware
- Zunehmende Datenunabhängigkeit durch Unterscheidung zwischen logischer und physischer Information
  - Anwenderprogramme können unabhängig von der Speicherform der eigentlichen Daten arbeiten

Datenbankmanagementsystem

- Datenbank oder Datenbasis (DB)
  - Systematische Sammlung zusammengehöriger Daten
  - Abbildung einer Miniwelt
- Datenbankmanagementsystem (DBMS)
  - Mittelsmann zwischen Daten und Anwendern
  - Aus zahlreichen Programmen gebildete Software
  - Nutzt Dienste des Betriebsystems zur Verwaltung der DB

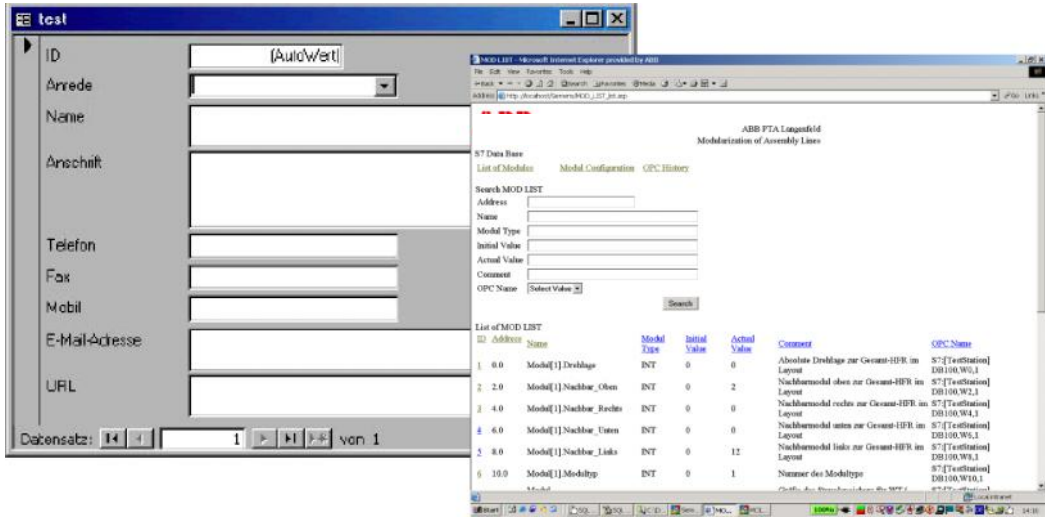


Drei Ebenen Architektur

- 1975 entwickelt von einer Studiengruppe des Standards Planning and Requirements Committee (SPARC) of the American National Standards Institute (ANSI)
- Erleichterung des Datenbankentwurfs durch Standardisierung
- Einteilung eines DBS in drei Ebenen mit verschiedenen Schemata der formalen Datenbeschreibung
- Festlegung dieser Schemata im Datenlexikon

Externe Ebene

- Definition der Sicht (View) des einzelnen Benutzers
- Verwendung einer Datenmanipulationssprache (Data Manipulation Language, DML) (z.B. SQL)
- Direkter Dialog über Kommandozeileneingabe oder durch eingebettete Befehle => Implementierung in eine Wirtssprache (Java, C/C++, ...)
- Grafische Benutzerschnittstellen zur Eingabe von Anfragen, etc. und Ergebnisanzeige



Konzeptionelle Ebene

- Darstellung des konzeptionellen Aufbaus und Definition des Informationsgehalts der Datenbank
- Verwendung der Datendefinitionssprache (Data Definition Language, DDL)
- Ausführliche Beschreibung aller Datenobjekte
- Erläuterungen aller Beziehungen zwischen diesen Objekten
- Definition aller notwendigen Integritätsbedingungen
  - Wahrung der semantischen Korrektheit
- Festlegung aller notwendigen Zugriffsberechtigungen

Interne Ebene

- Information über Art und Aufbau der Datenstrukturen
- Regelung des Datenzugriffs auf dem verwendeten Sekundärspeichermedium
- Nutzung der Betriebsystem-Funktionen
  - Abbildung des logischen in den physikalischen Adressraum
- Definition der internen Datenbankstrukturen über die Datenabspeicherungssprache
  (Data Storage Definition Language, DSDL)

Komponenten des DBMS

- Aufgaben des DBMS:
  - Überprüfung der Zugriffsberechtigungen => Datenschutz
  - Durchführung notwendiger Operationen
  - Einhaltung der Integritätsbedingungen
  - Recovery-Funktionen => Datensicherheit
  - Optimierungsmechanismen zur Effizienzsteigerung
- Das DBMS arbeitet auf drei Datenbeständen
  - Datenbank / Datenbasis (DB)
  - Logbuch: Protokollierung der Veränderungen an der Datenbank
    -> notwendig für Recovery-Mechanismen
  - Datenlexikon: Information über Daten und Strukturen der DB
    -> bereitgestellt für Kontroll-Funktionen

Datenbankzugriff und Transformationsregeln

- Abfrageeingabe über Benutzerschnittstelle
- Durchführung von Kontrollfunktionen
- Weitergabe an konzeptionelle Ebene:
  Transformation extern <-> konzeptionell
- Bearbeitung des Befehls und Weiterleitung an die interne Ebene:
  Transformation konzeptionell <-> intern
- Bestimmung von physischen Datensätzen und Lokalisation der Speicherbereiche
- Ablage des Ergebnisses im System-Puffer inklusive Aufbereitung für die Darstellung über die grafische Benutzerschnittstelle

Ziele eines Datenbankeinsatzes

- Datenunabhängigkeit
- Datenintegrität
- Vermeidung von unbeabsichtigter Redundanz
- Sicherheit
- Effizienz
- Mehrbenutzerbetrieb

Datenintegrität

- **Operationale Integrität**
  - Verhinderung von Fehlern, die durch den *gleichzeitigen Zugriff mehrerer Benutzer* entstehen -> Synchronisation paralleler Transaktionen
- **Recovery**
  - Wiederherstellung eines konsistenten Datenbankzustandes nach einem Systemausfall, Platten-Crash etc.
- **Datenschutz**
  - Verhinderung von Änderungen durch unberechtigte Personen
- **Semantische Integrität**
  - Verhinderung, dass die Datenbank wegen fehlerhafter Eingaben die Realität nicht korrekt abbildet

Integritätsüberwachung

- Auslöseregel einer Integritätsbedingung:
  - *operationsabhängig*:
    Die Prüfung soll bei bestimmten Operationen ausgeführt werden, z.B. beim Ändern, Einfügen, Löschen *(unverzögerte IB*).
  - *periodisch*:
    Das System überprüft periodisch die Einhaltung der Integritätsbedingung *(verzögerte IB*).
  - Benutzer- oder DB-Administrator-initiiert
    (verzögerte IB).
- Transaktion*:*
  Logische Verarbeitungseinheit, die integritätsbewahrend ist. (Integrität kann eventuell erst nach Ausführung mehrerer Operationen erfüllt werden)

Integritätsverletzung

- Reaktionen bei Feststellung einer Integritätsverletzung:
  1. Markierung
     Markierung der unzulässigen Daten, Warnung
     (soft integrity constraint)
  2. Meldung
     Meldung an zuständige Stelle (z.B. periodischer
     Zustandsbericht), im Allgemeinen zusammen mit 1
  3. Behandlung
     Behandlung als Fehler, Verweigerung der verlangten
     Änderung, unter Umständen Prozeß rückgängig machen;
     verursachender Benutzer wird informiert
  4. Unterbrechung
     Unterbrechung der Verarbeitung, Meldung
  5. Korrektur
     Versuch einer Korrektur

Integrität beim Datenbankentwurf

- Integritätsbedingungen sind Bestandteil des konzeptionellen Datenbankschemas
- Datendefinitionen (Struktur, Typ) sind spezielle Integritätsbedingungen

- Modellspezifische Typen von sematischen Integritätsbedingungen z.B. im Relationenmodell:
  - Domain-Eigenschaften
  - Schlüsseleigenschaften
  - funktionale Abhängigkeiten
  - Fremdschlüssel / referentielle Integrität

Fehler im Datenbankbetrieb

- **Transaktionsfehler (TF)**:
  - Eine Transaktion erreicht nicht ihr Ende (COMMIT).
  - **Mögliche Ursachen**:
    - Absichtliches Zurücksetzen durch den Benutzer.
    - Fehler im Benutzerprogramm.
    - Zurücksetzen durch das DBMS wegen eines Deadlocks, einer Integritätsverletzung o.ä.
  - **Häufigkeit**: In „größeren“ Systemen 10 – 100 mal pro Minute.
  - **Recovery-Dauer**: Nicht länger als die korrekte Ausführung der zurückgesetzten Transaktion.
- **Systemfehler (SF, „soft crash“)**
  - Fehler, bei dem das DBS selbst funktionsunfähig wird und Arbeitsspeicherinhalte verlorengehen.
  - **Mögliche Ursachen**:
    - Absturz des Betriebssystems.
    - Hardware-Fehler (außer Sekundärspeicher).
  - **Häufigkeit**: Mehrmals pro Woche.
  - **Recovery-Dauer**: Wenige Minuten.
- **Speicherfehler (SpF, „hard crash“)**
  - Fehler, die zum Verlust von Daten auf Massenspeicher führen.
  - **Mögliche Ursachen**:
    - Fehler in den Betriebssystemroutinen zum Schreiben auf Platte.
    - Hardware-Fehler der Festplattensteuerung.
    - „Head Crash“.
    - Verlust von Information durch magnetische Störfelder oder Alterung des Plattenmaterials.
  - **Häufigkeit**: Ein- oder zweimal pro Jahr.
  - **Recovery-Dauer**: 1 – 2 Stunden.

Entity Relationship Modell

- Entities, Domänen und Attribute
  - Entities sind wohlunterscheidbare Dinge der realen Welt z.B. Personen, Autos, Firmen, Bücher
  - Attribute sind die Eigenschaften der Entities z.B. zu einer Person gehören Name, Geburtsdatum, Anschrift, Augenfarbe, etc.
  - Wert ist die konkrete Ausprägung eines Attributes z.B. „Peter“ als Wert für das Attribut „Vorname“
    Man erhält ein konkretes Entity, indem den Attributen dieses Entities Werte zugeordnet werden
  - Domäne bzw. Wertebereich = Menge aller möglichen oder zugelassenen Werte für ein Attribut z.B. „Name“ als Zeichenkette der Länge 20 -> Domäne für Attribut „Name“ ist Menge aller Zeichenketten der Länge 20
  - Entity-Set = Zusammenfassung von einzelnen zusammen-gehörigen Entities z.B. alle Personen, alle Autos einer Firma, alle Schüler einer Schule

Attribute

- Arten von Attributen:
  - Einwertig:
    Attribut nimmt genau einen Wert an, z.B. Geburtsdatum bei einer Person
  - Mehrwertig:
    Attribut kann mehrere Werte annehmen, z.B. Hobbys einer Person
  - Zusammengesetzt:
    Attribut wird aus mehreren einzelnen Attributen zusammengesetzt, z.B. Name besteht aus Vorname und Nachname

Schlüssel

- Bedeutet Kombination von Attributen, die ein spezielles Entity identifizieren
- Schlüsselkandidat = Entfernung aller überflüssigen Attribute aus einem Schlüssel
  - es kann mehrere Schlüsselkandidaten geben
- Schlüssel muss redundanzfrei sein
- Primärschlüssel = ausgewählter Schlüsselkandidat
  - Andere Schlüssel werden Sekundärschlüssel genannt
  - Oft künstliches Attribut als Primärschüssel gewählt - z.B. KDNR

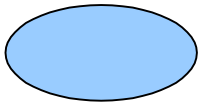
Entity

- Entity-Definition
  - Formale Beschreibung der Eigenschaften eines Entity-Sets
  - Form: Name = (Format,Primärschlüssel) -> E = (X,K)‘
  - Primärschlüssel K ist aus einwertigen Elementen von X zusammengesetzt
  - Notation der Elemente des Formats X:
    - Einwertige Attribute: A
    - Mehrwertige Attribute: {A}
    - Aus B_1 , ..., B_k zusammengesetztes Attribut: A(B_1 , ..., B_k)
- Beispiel
  - Charakterisierung eines Mitarbeiters durch folgende Eigenschaften: Personalnummer, Vorname, Nachname, Arbeitsbereiche, Adresse
  - E des Entity-Typs: Mitarbeiter
    Format X = {PerNr, Vorname, Nachname,{Bereiche}, Adresse(Strasse, Ort)}
    Primärschlüssel K = {PerNr}
  - Ergebnis: Mitarbeiter = ({PerNr, Vorname, Nachname,{Bereiche}, Adresse(Strasse, Ort)}, {PerNr})

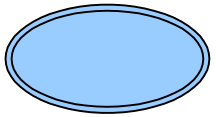
Grafische Darstellung von Entities

- Name des Entity-Typs im Rechteck

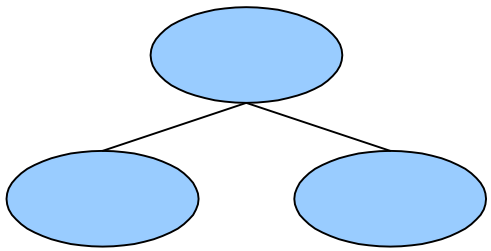
- Attribute in Ellipsen durch nicht gerichtete Kanten mit dem Entity-Typ verbunden
  - Einwertig:
    einfache Ellipse



  - Mehrwertig:
    Ellipse doppelt umrandet

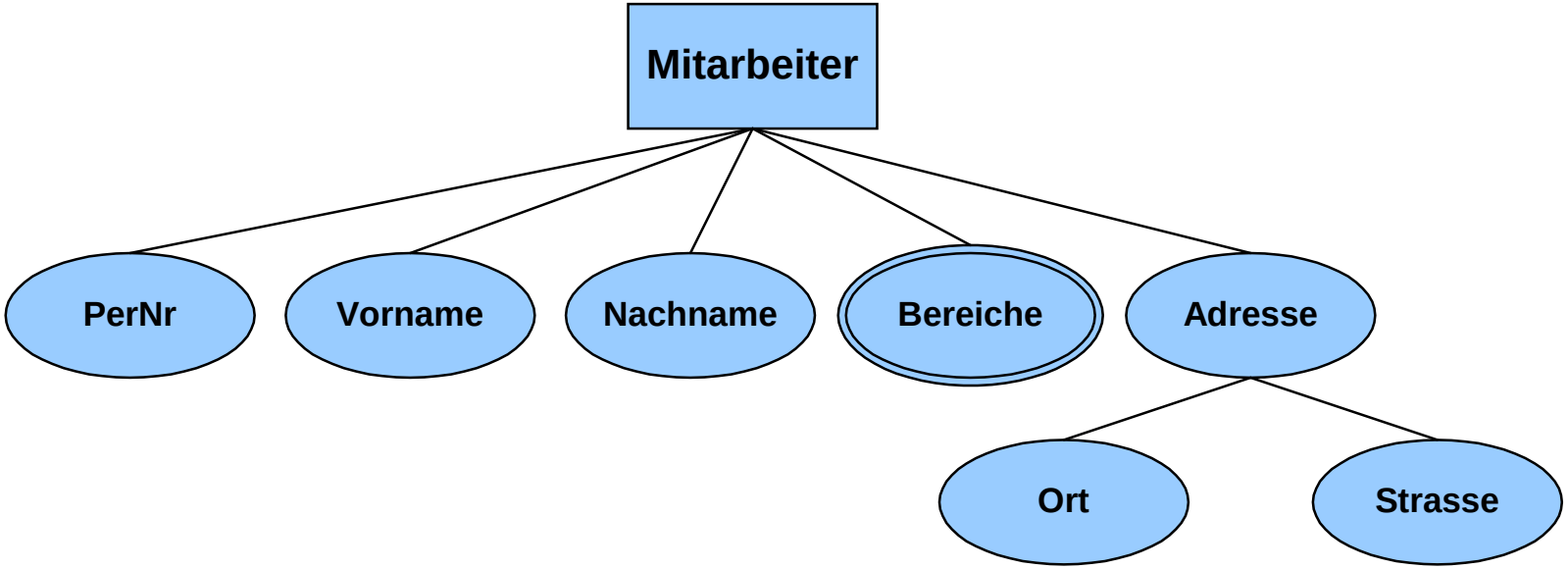


  - Zusammengesetzt:
    einzelne Komponenten durch Unterellipsen dargestellt



Beispiel grafische Darstellung von Entities

- Beispiel Mitarbeiter

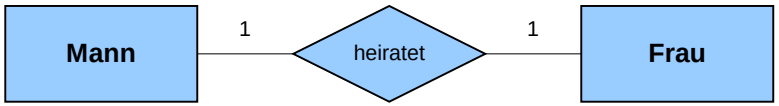


Beziehungen zwischen Entities

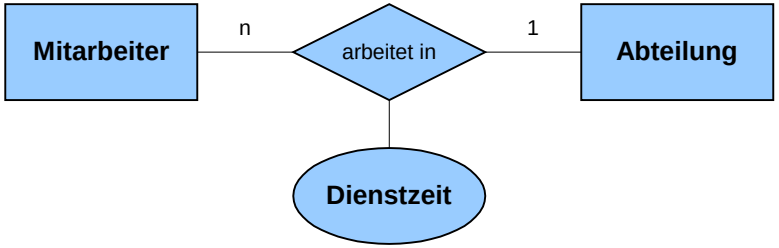
- Beziehungen bzw. Relationships
  - Verknüpfung zweier Entities durch eine Beziehung
  - Dargestellt durch Raute, die mit den Entity-Typen verknüpft ist
  - z.B.: Entity-Sets „Mitarbeiter“ und „Abteilung“, Beziehung „arbeiten in“ -> „Mitarbeiter arbeiten in Abteilung“
  - Verknüpfung mit genau zwei Entity-Typen := binär
  - Beziehung kann eigene Attribute enthalten

1:1 und 1:n Beziehung

- 1:1 – Beziehung
  - Jedem Entity e_1 aus E_1 wird höchstens ein Entity e_2 aus E_2 zugeordnet und umgekehrt.
  - Beispiel: E_1 : Mann, E_2 : Frau, Beziehung: heiratet

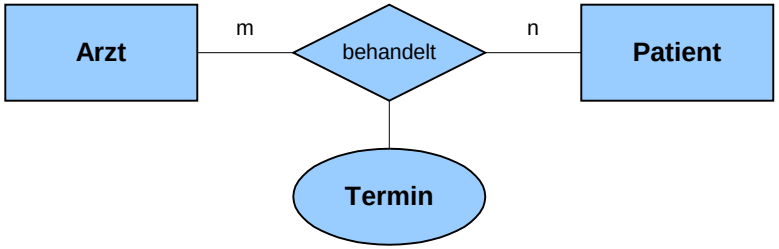


- 1:n oder n:1 – Beziehung
  - Jedem Entity e_1 aus E_1 können keines, eines oder mehrere (max. n) Entities e_2 aus E_2 zugeordnet werden.
  - Jedes Entity e_2 kann max. nur einem e_1 zugeordnet werden.
  - Beispiel: E_1 : Abteilung, E_2 : Mitarbeiter, Beziehung: arbeiten



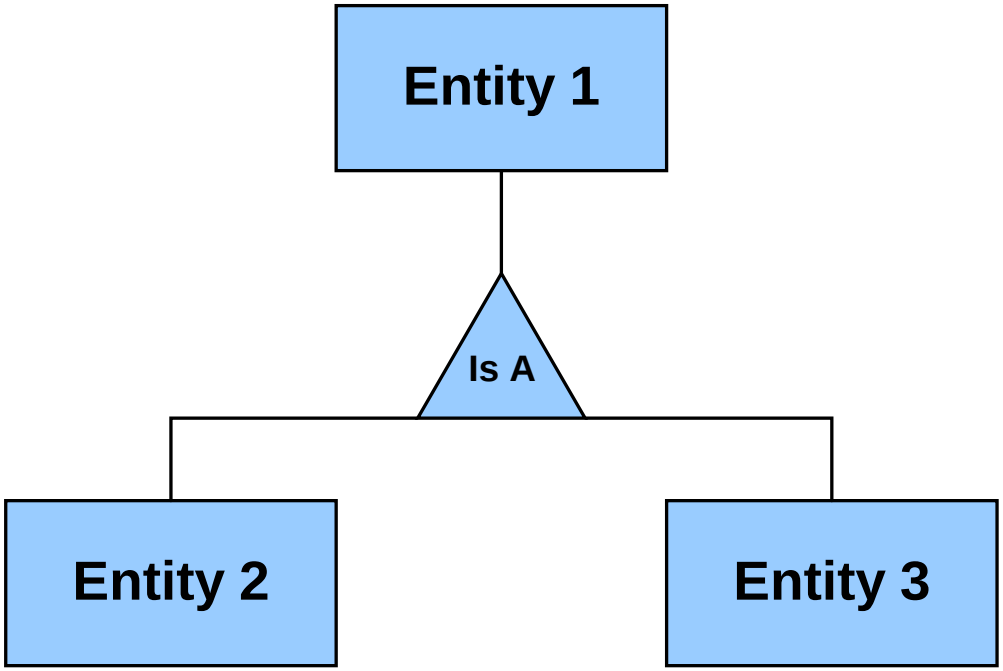
m:n Beziehung

- m:n – Beziehung
  - Jedem Entity e_1 aus E_1 können keines bis mehrere (max n) Entities e_2 aus E_2 zugeordnet werden und umgekehrt gilt e_2 kann max m e_1 zugeordnet bekommen.
  - Beispiel: E_1 : Arzt, E_2 : Patient, Beziehung: behandelt



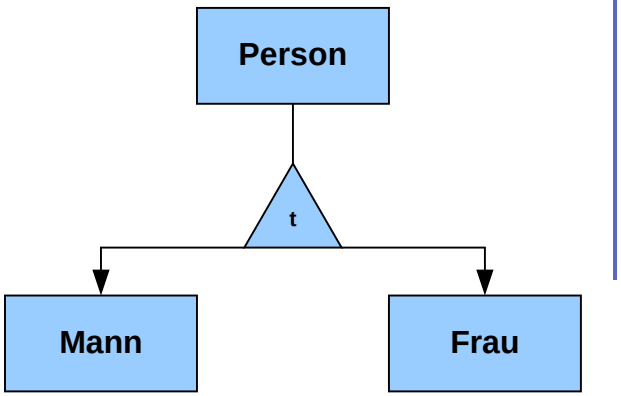
Is a Beziehung

- Stellt die Spezialisierung bzw. Verallgemeinerung dar
- Dargestellt durch Dreieck im ER-Diagramm
- Beschriftung der Is-A-Beziehung neben oder im Dreieck

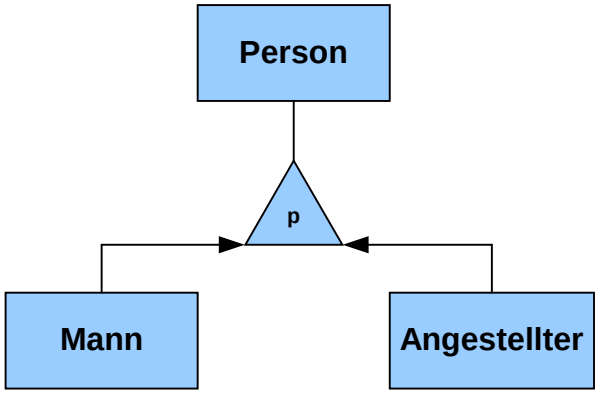


Is a Beziehung

- total oder partiell
  - total := es gibt keine weiteren Spezialisierungen außer den genannten
  - partiell := es kann noch weitere Spezialisierungen geben



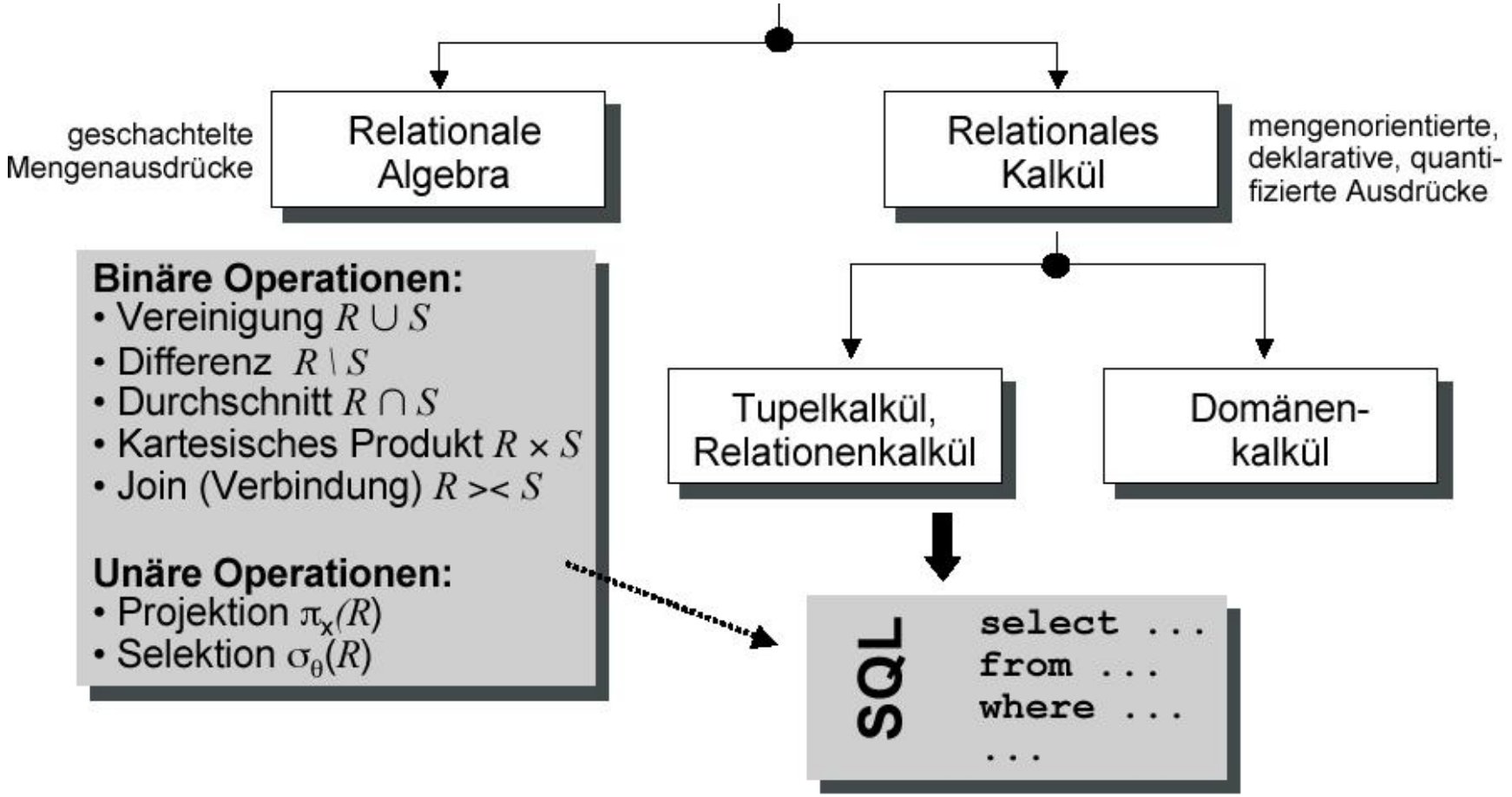
- disjunkt oder nicht-disjunkt
  - disjunkt: keine gemeinsamen Elemente
    -> Pfeil zur Spezialisierung
  - Dagegen nicht-disjunkt: gemeinsame Elemente
    -> Pfeil zur Verallgemeinerung



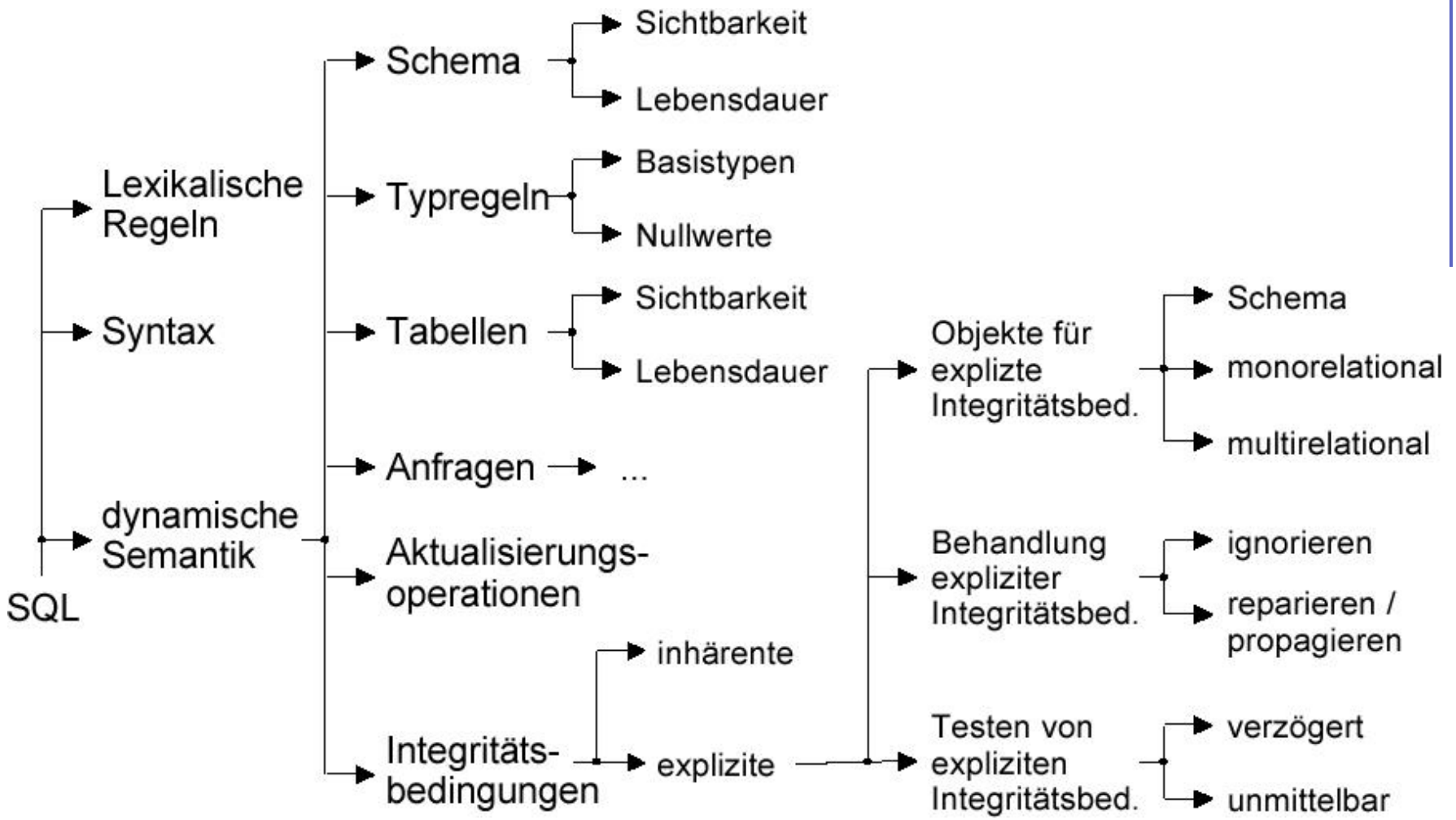
Modellierungsregeln

- Nicht mehr als 10 Tabellen pro Datenbank
- M:N Relationen nur wenn unbedingt notwendig
- Sprechende Tabellennamen und Bezeichner
- ...

Relationale Anfragesprachen



SQL und relationales DB Modell



SQL – Structured Query Language

- **Entwicklungsziele:**
  - möglichst einfach für den Endbenutzer;
  - möglichst deskriptiv, nicht prozedural
- **Benennung:**
  - SQL: „structured query language"
  - ursprünglich: SEQUEL - „structured english query language"
- **Entwicklung:** (im IBM-Labor)
  - basiert auf relationaler Algebra und Tupelkalkül
  - erste Vorschläge 1974; weiterentwickelt 1976-77
  - Implementierungen 1977 im System R (Prototyp eines rel. DBS)
- **Erste kommerzielle Implementierungen:**
  - Oracle: Ende 70er
  - IBM: SQL/DS (1981), DB2 (1983)

SQL Standardisierung

- SQL-86 (1986):
  erster Standardisierungsversuch; Standard noch lückenhaft
- SQL-89 (1989):
  Lücken von SQL-86 z.T. geschlossen; allerdings immer noch kein dynamisches SQL aus Anwendungsprogrammen
- SQL-2 (1992):
  Aktueller, umfassender Standard; drei Standardisierungslevel (entry-, intermediate-, full level); wird von den meisten Datenbanken noch nicht vollständig unterstützt
- SQL-3:
  Weiterentwicklung mit objekt-orientierten Ansätzen

Allgemeines zu SQL

- **Schnittstellen:**
  - SQL kann sowohl *interaktiv*,
  - als auch *eingebettet* in einer konventionellen Programmiersprache (z.B. C, FORTRAN, …) verwendet werden.
- **Terminologie:**
  - Table – Relation
  - record, row – Tupel
  - field, column – Attribut
- **relationales Modell:**
  - SQL ist keine direkte und vollständige Umsetzung des relationalen Modells.
  - Insbesondere sind die Tabellen in SQL keine Mengen, d.h. *Tupel können doppelt* auftreten.

SQL – DDL

- DDL – Data Definition Language

Anweisungen		Bedeutung
CREATE	SCHEMA	Anlegen eines Datenbankschemas
DROP	SCHEMA	Löschen eines Schemas
CREATE	TABLE	Definition einer Basistabelle
ALTER	TABLE	Ändern einer Basistabelle (Hinzufügen von Attributen)
DROP	TABLE	Löschen einer Basistabelle
CREATE	DOMAIN	benutzerdefinierte Domäne anlegen
DROP	DOMAIN	benutzerdefinierte Domäne löschen
CREATE	VIEW	Bilden einer Sicht aus Basistabellen
DROP	VIEW	Löschen einer Sicht

SQL – Datentypen

- SQL bietet folgende Basis-Datentypen (Auswahl)

Beschreibung	Datentyp	Kurzform
Zeichen(ketten):		
ein einzelnes Zeichen	CHARACTER	
Kette fester Länge	CHARACTER (n)	CHAR (n)
Kette variabler Länge	CHARACTER VARYING (n)	VARCHAR (n)
Bit-Datentyp:		
ein Bit	BIT	
Bitfolge fester Länge	BIT (n)	
Bitfolge variabler Länge	BIT VARYING (n)	
Exakte Zahlen:		
mit Nachkommastellen	DECIMAL [(p [, s])]	DEC (p, s)
dto.	NUMERIC [(p [, s])]	
Ganzzahl	INTEGER	INT
kleine Ganzzahl	SMALLINT	
Gleitkommazahlen:		
hohe Genauigkeit	DOUBLE PRECISION	
benutzerdefinierte Gen.	FLOAT (n)	
geringere Genauigkeit	REAL	

Beschreibung	Datentyp	Kurzform
Zeit, Datum, Zeitintervall:		
Datum (y ,m, d)	DATE	
Zeitpunkt (h, min, sec)	TIME	
Zeitpunkt (y, m, d, h, min,s ec)	TIMESTAMP	
Zeitintervall	INTERVAL sf TO ef INTERVAL f f ∈ {YEAR, MONTH, DAY, HOUR, MINUTE, SECOND}	

SQL – DML

- Data Manipulation Language
  - Update
  - Delete
  - Insert

Die Anfragesprache SQL

- Tabellen sind die Grundlagen einer relationalen Datenbank
- Speicherung der Daten strukturiert nach den Vorgaben der Tabellendefinitionen
- **Anlegen einer Datenbank**
  - CREATE DATABASE <Datenbankname>
- **Löschen einer Datenbank**
  - DROP DATABASE <Datenbankname>
- **Übersicht über die vorliegenden Datenbanken**
  - SHOW DATABASES
- **Eine konkrete Datenbank auswählen**
  - USE <Datenbankname>
- **Übersicht über die bestehenden Tabellen**
  - SHOW TABLES
- **Definition einer bestehenden Tabelle anzeigen lassen**
  - DESCRIBE <Tabellenname>

Relationsschema zur Tabellenerstellung

```
> CREATE TABLE t_personal
( pernr INTEGER NOT NULL PRIMARY KEY,
  nachname VARCHAR(50),
  vorname VARCHAR(50),
  geschlecht CHAR(1),
  abtnr SMALLINT,
  eintritt DATETIME,
  gehalt FLOAT DEFAULT 1200.50,
  verheiratet CHAR(1)
)
```

Personal	PerNr	Nachname	Vorname	Geschlecht	AbtNr	Eintritt	Gehalt	Verheiratet

Describe

```
mysql>DESCRIBE t_personal;
+-------------+-------------+------+-----+---------+-------+
| Field       | Type        | Null | Key | Default | Extra |
+-------------+-------------+------+-----+---------+-------+
| pernr       | int(11)     |      | PRI | 0       |       |
| nachname    | varchar(50) | YES  |     | NULL    |       |
| vorname     | varchar(50) | YES  |     | NULL    |       |
| geschlecht  | char(1)     | YES  |     | NULL    |       |
| abtnr       | smallint(6) | YES  |     | NULL    |       |
| eintritt    | date        | YES  |     | NULL    |       |
| gehalt      | float       | YES  |     | 1200.5  |       |
| verheiratet | char(1)     | YES  |     | NULL    |       |
+-------------+-------------+------+-----+---------+-------+
8 rows in set (0.00 sec)

mysql>
```

Tabellendefinition

- Erzwingen der Eingabe

```
> CREATE TABLE t_test (
  id INTEGER NOT NULL )
```

- Vorbesetzen mit Standartwerten

```
> CREATE TABLE t_test
(
  geschlecht CHAR(1) DEFAULT "m",
  anrede CHAR(10) DEFAULT 'Herr',
  jahre SMALLINT DEFAULT 35,
  beitrag FLOAT DEFAULT 12.50,
  anzahl INTEGER DEFAULT NULL
)
```

Primärschlüssel

1. Möglichkeit

```
CREATE TABLE t_test
(
  nummer INTEGER NOT NULL PRIMARY KEY,
  bezeichnung VARCHAR(100)
)
```

2. Möglichkeit

```
> CREATE TABLE t_test
(
  artnr INTEGER NOT NULL,
  kdnr INTEGER NOT NULL,
  bestellmenge SMALLINT,
  PRIMARY KEY(artnr,kdnr)
)
```

Sekundärschlüssel

1. Möglichkeit

```
> CREATE TABLE t_test
( nummer INTEGER NOT NULL PRIMARY KEY,
  test INTEGER NOT NULL UNIQUE
)
```

2. Möglichkeit

```
> CREATE TABLE t_person
( nummer INTEGER NOT NULL,
  vorname VARCHAR(50) NOT NULL,
  nachname VARCHAR(50) NOT NULL,
  strasse VARCHAR(100) NOT NULL,
  plz VARCHAR(5) NOT NULL,
  ort VARCHAR(100), tel VARCHAR(30),
  fax VARCHAR(30),
  PRIMARY KEY(nummer),
  UNIQUE(vorname, nachname, strasse, plz)
)
```

Indizes

- Suche über den Suchbaum – Adressen aus dem Index
- Aktualisierung bei jeder Tabellenänderung
  -> unsymmetrische Struktur
- von Zeit zu Zeit Index ab- und wieder neu aufbauen

- **Erstellen eines Index**

```
CREATE INDEX i_vorname_person ON t_person(vorname)
```

- **Löschen eines Index**

```
DROP INDEX i_vorname_person ON t_person
```

Ändern einer Tabelle

- Einfügen einer Spalte
  - Standardmäßig am Ende

```
> ALTER TABLE t_person
  ADD kravers INTEGER
```

  - Nach einer speziellen Spalte

```
> ALTER TABLE t_person
  ADD kravers INTEGER
  AFTER nummer
```

  - Am Anfang

```
> ALTER TABLE t_person
  ADD kravers INTEGER
  FIRST
```

- Löschen einer Spalte
  -

```
> ALTER TABLE t_person
  DROP kravers
```

Nachträgliches Ändern

- Nachträgliches Ändern der Spaltendefinition

```
> ALTER TABLE t_person CHANGE nachname
  nachname VARCHAR(100) NOT NULL
```

  - Verkürzen von Zeichenketten führt bei bestehenden Werten unter Umständen zu ungültigen Einträgen
    -> überschüssige Zeichen werden einfach abgeschnitten
  - Veränderung von FLOAT in INTEGER führt zum „Abschneiden“ der Nachkommastellen

- Löschen eines Index

```
> ALTER TABLE t_person
  DROP INDEX i_vorname_person
```

Bsp.: Einfügen eines Primärschlüssels

- Beispiel einer Tabellenerstellung

```
> CREATE TABLE t_abt
( nummer INTEGER NOT NULL,
  bezeichnung VARCHAR(100)
)
```

- Primärschlüssel in eine Tabelle einfügen

```
> ALTER TABLE t_abt
  ADD PRIMARY KEY(nummer)
```

- Fehlermeldung falls Primärschüsselspalte nicht NOT NULL

```
> All parts of a PRIMARY KEY must be NOT NULL;
```

- Sekundärschlüssel

```
> ALTER TABLE t_abt
  ADD UNIQUE(bezeichnung)
```

- Löschen eines Primärschlüssels

```
> ALTER TABLE t_abt DROP PRIMARY KEY
```

Datensätze in Tabelle einfügen (1)

1. **Möglichkeit**
   - **Reihenfolge der einzufügenden Werte und die der Spalten müssen übereinstimmen**
   - **Werttypen müssen mit Datentypen der Spaltendefinition übereinstimmen**
   - **Zeichenketten sind in Hochkommas einzuschließen**

```
> INSERT INTO t_personal
  VALUES
  ( 11,
    "Kunze",
    "Manfred",
    "m",
    3,
    "2002-12-01",
    2400.50,
    "n“
  )
```

Datensätze in Tabelle einfügen (2)

2. **Möglichkeit**
   - **Angabe der Spalten in die Werte eingefügt werden sollen**
   - **Reihenfolge kann dabei von der Tabellendefinition abweichen**
   - **Reihenfolge der Werte muss der Reihenfolge der aufgelisteten Spalten entsprechen**

```
> INSERT INTO t_personal
  ( nachname,
    pernr,
    vorname
  )
  VALUES
  ( "Schmitz",
    12,
    "Peter“
  )
```

Datensätze in Tabelle einfügen (3)

3. **Möglichkeit**
   - **Mehrere Zeilen gleichzeitig eintragen**
   - **jede Zeile begrenzt durch runde Klammern, Komma separiert**
   - **Reihenfolge der Werte muss wieder zur Reihenfolge der aufgelisteten Spalten passen**

```
> INSERT INTO t_personal
  ( nachname,
    pernr,
    vorname
  )
  VALUES
  ("Schmitz", 12, "Peter"),
  ("Kunze", 11, "Manfred")
```

Datensätze in Tabelle einfügen (4)

4. **Möglichkeit**
   - **Werteeingabe über Datei**
   - **Werte standardmäßig Tabulator-Trennung, sonst separate Angabe des Trennungszeichens**
   - **Begrenzung der Zeichenketten und Zeilenendezeichen können gezielt angegeben werden**

```
> LOAD DATA LOCAL
  INFILE "daten.txt“
  INTO TABLE t_personal
  FIELDS TERMINATED BY ’;‘
  OPTIONALLY ENCLOSED BY ’"‘
  LINES TERMINATED BY "\n„
```

```
daten.txt:
11;"Kunze";"Manfred";"m";3;"2002-12-01";2400.50;"n“
12;"Schmitz";"Peter";"m";;;;
```

Tabelle leeren/löschen

- Eine Tabelle leeren über DELETE FROM
  - `> DELETE FROM t_test`
  - **Tabelle wird vollständig geleert**
  - **Tabellendefinition bleibt bestehen**
  - **Es können neue Zeilen eingefügt werden**

- Eine Tabelle löschen mittels DORP TABLE
  - `> DROP TABLE t_test`
  - **Tabelle wird vollständig geleert**
  - **Tabellendefinition wird ebenfalls gelöscht**
  - **Es können keine neuen Zeilen mehr eingefügt werden**

Daten auswählen und bearbeiten

- Ausgangstabelle

pernr	nachname	vorname	geschlecht	abtnr	eintritt	gehalt	verheiratet
1	Meier	Hans	m	4	1998-02-01	2400	J
2	Müller	Michael	m	1	1984-10-15	1800	J
3	Pauly	Manfred	m	3	1991-04-15	2000	N
4	Bergstein	Claudia	w	3	1998-08-01	2200	N
5	Jansen	Josefine	w	1	1979-08-01	2100	J
6	Abels	Gisela	w	2	1989-09-15	1900	J
7	Derichs	Holger	m	4	1986-11-01	2300	J
8	Müller	Volker	m	1	1996-03-15	1900	N
9	Mertens	Heinz	m	3	1999-06-01	2100	J
10	Buchholz	Elisabeth	w	2	1988-07-15	2400	N

Tabelle selektieren und anzeigen

- Tabelle vollständig anzeigen

```
> SELECT * FROM t_personal
```

- Bestimmte Spalten anzeigen

```
> SELECT pernr, nachname, vorname FROM t_personal
```

pernr	nachname	vorname
1	Meier	Hans
2	Müller	Michael
3	Pauly	Manfred
4	Bergstein	Claudia
5	Jansen	Josefine
6	Abels	Gisela
7	Derichs	Holger
8	Müller	Volker
9	Mertens	Heinz
10	Buchholz	Elisabeth

Doppelte Einträge unterdrücken

- Ohne Entfernen der Doppeltwerte

```
> SELECT gehalt FROM t_personal
```

gehalt
2400
1800
2000
2200
2100
1900
2300
1900
2100
2400

- Mit Entfernen der Doppeltwerte

```
> SELECT DISTINCT gehalt FROM t_personal
```

gehalt
2400
1800
2000
2200
2100
1900
2300

Auswahl mittels WHERE

- Einfache Anfrage

```
> SELECT * FROM t_personal WHERE abtnr=4
```

pernr	nachname	vorname	geschlecht	abtnr	eintritt	gehalt	verheiratet
1	Meier	Hans	m	4	1998-02-01	2400	J
7	Derichs	Holger	m	4	1986-11-01	2300	J

- Zusätzliche Anfragebedingung

```
> SELECT * FROM t_personal WHERE gehalt > 2000
```

pernr	nachname	vorname	geschlecht	abtnr	eintritt	gehalt	verheiratet
1	Meier	Hans	m	4	1998-02-01	2400	J
4	Bergstein	Claudia	w	3	1998-08-01	2200	N
5	Jansen	Josefine	w	1	1979-08-01	2100	J
7	Derichs	Holger	m	4	1986-11-01	2300	J
9	Mertens	Heinz	m	3	1999-06-01	2100	J
10	Buchholz	Elisabeth	w	2	1988-07-15	2400	N

Verknüpfung von Bedingungen

- Operatoren

Operator	Bedeutung	Operator	Bedeutung
=	Gleichheit	<>	Ungleichheit
<	kleiner	<=	kleiner-gleich
>	größer	>=	größer-gleich

- AND Verknüpfung

```
> SELECT nachname, abtnr,
  eintritt, gehalt
  FROM t_personal
  WHERE abtnr<>3 AND gehalt>2100
```

nachname	abtnr	eintritt	gehalt
Meier	4	1998-02-01	2400
Derichs	4	1986-11-01	2300
Buchholz	2	1988-07-15	2400

- OR Verknüpfung

```
> SELECT nachname, abtnr,
  gehalt, verheiratet
  FROM t_personal
  WHERE gehalt > 2100
  OR verheiratet="J"
```

nachname	abtnr	gehalt	verheiratet
Meier	4	2400	J
Müller	1	1800	J
Bergstein	3	2200	N
Jansen	1	2100	J
Abels	2	1900	J
Derichs	4	2300	J
Mertens	3	2100	J
Buchholz	2	2400	N

Not Verknüpfung

```
> SELECT * FROM t_personal WHERE NOT abtnr=3
```

- Alternative

```
> Select * FROM t_personal WHERE abtnr <> 3
```

pernr	nachname	vorname	geschlecht	abtnr	eintritt	gehalt	verheiratet
1	Meier	Hans	m	4	1998-02-01	2400	J
2	Müller	Michael	m	1	1984-10-15	1800	J
5	Jansen	Josefine	w	1	1979-08-01	2100	J
6	Abels	Gisela	w	2	1989-09-15	1900	J
7	Derichs	Holger	m	4	1986-11-01	2300	J
8	Müller	Volker	m	1	1996-03-15	1900	N
10	Buchholz	Elisabeth	w	2	1988-07-15	2400	N

Klammerung

- Prioritäten durch Klammerung erzwingen

```
> SELECT * FROM t_personal
  WHERE geschlecht="w“
  AND (abtnr=1 OR gehalt>2000)
```

pernr	nachname	vorname	geschlecht	abtnr	eintritt	gehalt	verheiratet
4	Bergstein	Claudia	w	3	1998-08-01	2200	N
5	Jansen	Josefine	w	1	1979-08-01	2100	J
10	Buchholz	Elisabeth	w	2	1988-07-15	2400	N

```
> SELECT * FROM t_personal
  WHERE (geschlecht="w“ AND abtnr=1)
  OR gehalt>2300
```

pernr	nachname	vorname	geschlecht	abtnr	eintritt	gehalt	verheiratet
1	Meier	Hans	m	4	1998-02-01	2400	J
5	Jansen	Josefine	w	1	1979-08-01	2100	J
10	Buchholz	Elisabeth	w	2	1988-07-15	2400	N

Aggregatfunktionen

- Minimum und Maximum

```
> SELECT MIN(gehalt),
  MAX(gehalt) FROM t_personal
```

MIN(gehalt)	MAX(gehalt)
1800	2400

- Anzahl aller Zeilen

```
> SELECT COUNT(*) FROM t_personal
```

COUNT(*)
10

- Anzahl bestimmter Zeilen

```
> SELECT COUNT(*)
  FROM t_personal WHERE abtnr=2
```

COUNT(*)
2

- Summe und Durchschnitt

```
> SELECT SUM(gehalt), AVG(gehalt)
  FROM t_personal
```

SUM(gehalt)	AVG(gehalt)
21100	2110.0000

Spalten berechnen und benennen

- Benennen von Spalten

```
> SELECT COUNT(*) AS Zeilenanzahl
  FROM t_personal
```

Zeilenanzahl
10

- Berechnen von Spalten

```
> SELECT nachname, vorname, gehalt
  AS brutto, gehalt*0.7
  AS netto FROM t_personal
```

nachname	vorname	brutto	netto
Meier	Hans	2400	1680.0
Müller	Michael	1800	1260.0
Pauly	Manfred	2000	1400.0
Bergstein	Claudia	2200	1540.0
Jansen	Josefine	2100	1470.0
Abels	Gisela	1900	1330.0
Derichs	Holger	2300	1610.0
Müller	Volker	1900	1330.0
Mertens	Heinz	2100	1470.0
Buchholz	Elisabeth	2400	1680.0

- Berechnungsoperatoren

Operator	Funktion
+	Addition
-	Subtraktion
*	Mulitplikation
/	Division

Anfragen mit in und between

- IN-Operator

```
> SELECT * FROM t_personal WHERE (gehalt=1800)
  OR (gehalt=2000) OR (gehalt=2200)
```

einfacher:

```
> SELECT * FROM t_personal
  WHERE gehalt IN (1800, 2000, 2200)
```

pernr	nachname	vorname	geschlecht	abtnr	eintritt	gehalt	verheiratet
2	Müller	Michael	m	1	1984-10-15	1800	J
3	Pauly	Manfred	m	3	1991-04-15	2000	N
4	Bergstein	Claudia	w	3	1998-08-01	2200	N

- BETWEEN-Operator

```
> SELECT * FROM t_personal
  WHERE gehalt>=2000 AND gehalt<=2300
```

einfacher:

```
> SELECT * FROM t_personal
  WHERE gehalt BETWEEN 2000 AND 2300
```

Anfragen mit like

- LIKE-Operator

```
> SELECT nachname, gehalt FROM t_personal
  WHERE nachname="Pauly“
```

```
> SELECT nachname, gehalt FROM t_personal
  WHERE (nachname LIKE "Pauly")
```

```
> SELECT pernr, nachname, vorname
  FROM t_personal
  WHERE (nachname LIKE "M%")
  OR (vorname LIKE "%g%")
  OR (nachname LIKE "%holz")
```

%: Joker-Zeichen

pernr	nachname	vorname
1	Meier	Hans
2	Müller	Michael
6	Abels	Gisela
7	Derichs	Holger
8	Müller	Volker
9	Mertens	Heinz
10	Buchholz	Elisabeth

Sortierte Ausgabe

- ORDER BY

```
> SELECT * FROM t_personal
  ORDER BY nachname ASC

> SELECT * FROM t_personal
  ORDER BY geschlecht DESC, abtnr ASC, gehalt
```

pernr	nachname	vorname	geschlecht	abtnr	eintritt	gehalt	verheiratet
5	Jansen	Josefine	w	1	1979-08-01	2100	J
6	Abels	Gisela	w	2	1989-09-15	1900	J
10	Buchholz	Elisabeth	w	2	1988-07-15	2400	N
4	Bergstein	Claudia	w	3	1998-08-01	2200	N
2	Müller	Michael	m	1	1984-10-15	1800	J
8	Müller	Volker	m	1	1996-03-15	1900	N
3	Pauly	Manfred	m	3	1991-04-15	2000	N
9	Mertens	Heinz	m	3	1999-06-01	2100	J
7	Derichs	Holger	m	4	1986-11-01	2300	J
1	Meier	Hans	m	4	1998-02-01	2400	J

Anfragen mehrerer Tabellen

- Ausgangstabellen
  - **Tabelle: t_personal**

pernr	nachname	vorname	geschlecht	abtnr	gehalt	vernr
1	Meier	Hans	m	4	2400	1
2	Müller	Michael	m	1	1800	3
3	Pauly	Manfred	m	3	2000	2
4	Bergstein	Claudia	w	3	2200	4
5	Jansen	Josefine	w	1	2100	4
6	Abels	Gisela	w	2	1900	1
7	Derichs	Holger	m	4	2300	5
8	Müller	Volker	m	1	1900	5
9	Mertens	Heinz	m	3	2100	2
10	Buchholz	Elisabeth	w	2	2400	3

- **Tabelle: t_krankenkasse**

vernr	kkname
1	AOK
2	Barmer
3	DAK
4	Techniker
5	Privat

- **Tabelle: t_abteilung**

abtnr	abtname
1	EDV
2	Logistik
3	Sekrtariat
4	Fuhrpark

Full Join

- Betreffende Tabellen hinter FROM angeben

```
> SELECT vorname, nachname, abtname
  FROM t_personal, t_abteilung
```

vorname	nachname	abtname
Hans	Meier	EDV
Hans	Meier	Logistik
Hans	Meier	Sekretariat
Hans	Meier	Fuhrpark
Michael	Müller	EDV
Michael	Müller	Logistik
Michael	Müller	Sekretariat
Michael	Müller	Fuhrpark
...	...	...

- Falls Namensgleichheit der Spalten, Tabellennamen mittels Punkt-Operator dem Spaltennamen voranstellen, z.B.

```
t_personal.abtnr
```

Inner Join

- Explizite Angabe der Verknüfung

```
> SELECT vorname, nachname, abtname
  FROM t_personal, t_abteilung
  WHERE t_personal.abtnr=t_abteilung.abtnr
```

vorname	nachname	abtname
Hans	Meier	EDV
Josefinie	Jansen	EDV
Volker	Müller	EDV
Gisela	Abels	Logistik
Elisabeth	Buchholz	Logistik
Manfred	Pauly	Sekretariat
Claudia	Bergstein	Sekretariat
Heinz	Mertens	Sekretariat
Hans	Meier	Fuhrpark
Holger	Derichs	Fuhrpark

Beispiele Inner Join

- ```
  > SELECT * FROM t_personal, t_abteilung
  WHERE abtname="EDV“
  AND t_personal.abtnr=t_abteilung.abtnr
  ```

pernr	nachname	vorname	geschlecht	abtnr	gehalt	vernr	abtnr	abtname
2	Müller	Michael	m	1	1800	3	1	EDV
5	Jansen	Josefine	w	1	2100	4	1	EDV
8	Müller	Volker	m	1	1900	5	1	EDV

- ```
  > SELECT SUM(gehalt) FROM t_personal, t_abteilung
  WHERE abtname="EDV“
  AND t_personal.abtnr=t_abteilung.abtnr
  ```

SUM(gehalt)
5800

- ```
  > SELECT nachname, vorname FROM t_personal,
  t_krankenkasse
  WHERE geschlecht="m“
  AND t_personal.vernr=t_krankenkasse.vernr
  AND kkname="Barmer"
  ```

Vorname	Nachname
Pauly	Manfred
Mertens	Heinz
  ```

Update und Delete

- Ändern über UPDATE

```
> UPDATE t_personal SET gehalt=2000
> UPDATE t_personal SET gehalt=2500
  WHERE nachname="Pauly“
```

- Löschen mittels DELETE FROM ... WHERE

```
> DELETE FROM t_personal WHERE pernr=3
```

Objekt Database Management Group

- ODMG:
  Objekt Database Management Group
- ODMG-93:
  Erster Standard für objektorientierte Datenbanken der ODMG von 1993
  - Objektmodell
  - Objekt-Definitionssprache ODL
  - deklarative Objekt-Anfragesprache OQL
  - C++-Sprachanbindung (C++-OML)
  - Java- Sprachanbindung
  - Smalltalk-Sprachanbindung

Die Anfragesprache OQL (object query language)

- Deklarative Anfragesprache für ODBS
- Analog zum DML-Teil von SQL
- Basiert auf dem ODMG-Objektmodell
- Ist eigenständige, interaktive Datenbanksprache
- Ist unabhängig von einer Programmiersprache
- Ist eine einbettbare Sprache, welche die Sprachkonstrukte der Gastsprache nutzt
- Ist eine Obermenge des SQL-Standards
- Besitzt keine expliziten Änderungsoperationen
- Erlaubt Anfragen nicht nur auf Mengen, sondern auch auf Strukturen und Listen
- Ermöglicht vordefinierte Anfragen, die aber nicht - wie externe Sichten bei SQL – im Datenbankschema abgelegt werden.
- Erlaubt neben dem Zugriff auf Attribute auch den Aufruf von Operationen und das Durchlaufen von Beziehungen

Anfragen auf Sammlungen

- OQL besteht aus einer Menge von Anfrageausdrücken, die auf Sammlungen arbeiten.
- Möglich ist:
  - Zugriff auf Attribute
  - Durchlaufen von Beziehungen
  - Aufruf von Operationen
  - Quantoren
  - Sortieren und Gruppieren
- **Zugriff auf Attribute**

  Allgemeine Form (in „[ ]“ geschriebenes ist optional):

  select e **from** e 1 **[a**s] x 1 , ..., e n **[a**s] x n **where** e’

  und (falls keine Duplikate erlaubt sind)

  select distinct e **from** e 1 **[a**s] x 1 , ..., e n **[a**s] x n

  where e’

  e, e’, e 1 ,..., e n , sind Ausdrücke

  x 1 , ..., x n sind Variablennamen