jeckle.de

Vorlesung Datenbanken

Hinweis

Aufgrund der verfügbaren Menge guter einführender (auch deutschsprachiger) Datenbankliteratur verzichtet das vorliegende Scriptum darauf den in der Literatur verfügbaren Stoff nochmals aufzubereiten, sondern versammelt die Definitionen, Beispiele und Anmerkungen der Vorlesungen in einer übersichtlichen Zusammenstellung.
Für die vertiefende ausführliche lehrbuchartige Darstellung des Stoffes sei auf die empfohlene Literatur verwiesen.

1 Motivation und Einführung

1.1 Begriffsbestimmung: Was ist eine Datenbank?

Motivation für die Einführung einer Datenbank anstatt selbsterstellter Verwaltungs- und Zugriffsroutinen:

- Daten-Programm-Unabhängigkeit.
  Die verwalteten Daten sollen unabhängig vom sie verarbeiteten Programm gespeichert und zugreifbar sein.
  Dies wäre zwar in einem ersten Schritt auch durch die Verwendung des Dateisystems möglich, allerdings würde hierfür ein Programm zur Abbildung der Programmdaten auf die Dateistrukturen benötigt, welches selbst wieder eine Abhängigkeitsbeziehung zwischen Daten und Programm --- nun eben dem Abbildungsprogramm --- darstellen würde.
- Flexible Speicherung.
  Datenbankmanagementsysteme speichern die verwalteten Daten deutlich flexibler als selbsterstellte Routinen und sind somit hinsichtlich der Zukunftsfähigkeit effizienter.
- Verwaltungsfunktionen.
  Datenbankmanagementsysteme bieten in der Regel eine Reihe über die reine Datenverwaltung hinausgehende Funktionen wie Backup-Recovery, Integritätssicherung, Synchronisation gleichzeitiger Zugriffe oder Transaktionskontrolle an, die nicht selbständig implementiert werden

müssen.

Grundlegende Begriffe

Definition 1: *Daten*
Daten sind durch die Maschine verarbeitbare Einheiten.

Definition 2: *Information*
Daten die Bedeutung für den Empfänger besitzen.
Nach Shannon ermißt sich der Wert einer Information durch den Zuwachs der durch den Adressaten nach Kenntnis der Information beantwortbaren Ja/Nein-Fragen.

Mehr zum Unterschied zwischen *Daten* und *Information*:

- Homepage des Arbeitskreises *Bildung*, einem Zusammenschluß von Stipendiaten der Friedrich Naumann Stiftung
- Glossar der deutschsprachigen Anleitung zu PGP

Definition 3: *Datenbank*
Eine Datenbank (engl. *data base*) ist ein integrierter, persistenter Datenbestand einschließlich aller relevanten Informationen über die dargestellten Information (sog. Metainformation, d.h. Integritätsbedingungen und Regeln), der einer Gruppe von Benutzern in nur einem Exemplar zur Verfügung steht und durch ein DBMS verwaltetet wird.

Definition 4: *Datenbankmanagementsystem (DBMS)*
Ein Datenbankmanagementsystem (DBMS) ist die Gesamtheit aller Programme zur Erzeugung, Verwaltung und Manipulation einer Datenbank.
Im Deutschen wird auch der Begriff *Datenbankverwaltungssystem* (DBVS) synonym verwendet.

Beispiele verfügbarer DBMS:

- Die DBMS-Produkte des Herstellers Sybase
- IDMS von Computer Associates
- IMS, DB2 und Informix von IBM
- MySQL des gleichnamigen Herstellers
- Oracle 9i des Herstellers Oracle
- SQLServer und Access des Herstellers Microsoft

Beispiel 1: Am Markt verfügbare DBM-Systeme

Definition 5: *Relationales DBMS*
Ein relationales Datenbankmanagementsystem (RDBMS) ist ein DBMS, welches intern gemäß dem relationalen Modell organisiert ist.

Bei den genannten DBMS *MySQL*, *SQLServer*, *Access*, *DB2* und *Oracle* handelt es sich um relationale Systeme, bzw. Weiterentwicklungen davon.

Beispiel 2: Am Markt verfügbare RDBM-Systeme

Definition 6: *Relation*
Eine Relation $R(A_1, A_2 \ldots A_n)$ ist eine benannte Menge von *n*-Tupeln, wobei ein *n*-Tupel eine Anordnung von *n* atomaren, d.h. einfachen (nicht weiter zerlegbaren) Attributen $A_1, A_2 \ldots A_n$ ist.

Die Relation *Person* mit den Attributen *Vorname*, *Nachname* und *Geburtsdatum*.
Werteausprägungen davon:
$Person_1$("Meier", "Schorsch", "1955-10-01")
$Person_2$("Huber", "Franz", "1945-08-03")
...

Die Relation *Student* mit den Attributen *Name*, *Matrikelnummer*, *Semester* und *regelmäßigerMensabesucher*.
Werteausprägungen davon:
$Student_1$("Meier Schorsch", "08154711", "WIB 1", "true")
$Student_2$("Müller Xaver", "73619452", "BCM 4", "false")

...

Beispiel 3: Relationen

Definition 7: *Tabelle*
Eine Tabelle unterscheidet sich von einer Relation darin, daß ein Tupel mehrfach auftreten darf; eine Tabelle ist mathematisch keine Menge.

Die Tabelle *Student* mit den Attributen *Name*, *Matrikelnummer*, *Semester* und *regelmäßigerMensabesucher*.
Werteausprägungen davon:
$Student_1$("Meier Schorsch", "08154711", "WIB 1", "true")
$Student_2$("Müller Xaver", "73619452", "BCM 4", "false")
$Student_3$("Müller Xaver", "73619452", "BCM 4", "false")
...

Man beachte, daß der dritte Eintrag doppelt vorkommt, d.h. in all seinen Wertbelegungen mit dem zweiten übereinstimmt.

Beispiel 4: Tabelle

Definition 8: *Modell*
Ein Modell bildet einen existierenden Sachverhalt deskriptiv nach oder nimmt einen Zukünftigen präskriptiv voraus.
Teilweise wird der Begriff *Schema* synonym gebraucht.

Deskriptive Modelle: Modelleisenbahn, Stadtplan, Photo.
Präskriptive Modelle: Bauplan eines Hauses, Skizze eines Gemäldes, maßstäblich verkleinerte Skulptur als Vorbild.

Beispiel 5: Modelle

Definition 9: *Datenbanksprache*
Eine Sprache die zur Erzeugung oder Interaktion mit den Daten bzw. zu deren Verwaltung eingesetzt wird.
Es werden unterschieden:

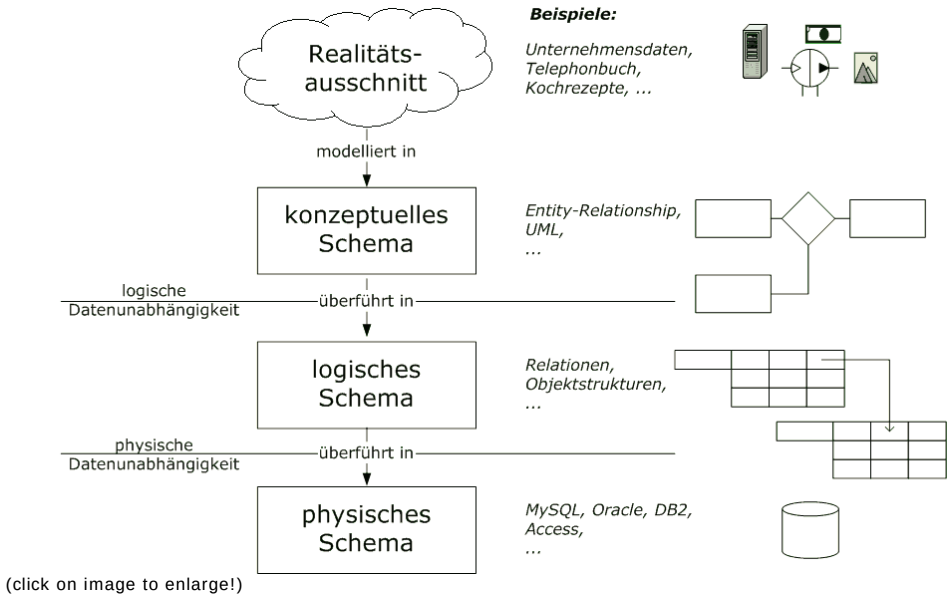
- Data Definition Language (DDL).
  Zur Erzeugung eines Datenmodells.
- Data Manipulation Language (DML).
  Zur Modifikation der verwalteten Daten.
- Data Retrieval Language (DRL).
  Zur Anfrage der in einer Datenbank gespeicherten Daten.
- Data Control Language (DCL).
  Zur Festlegung und Kontrolle von Zugriffsberechtigungen.

Im Verlauf der Vorlesung wird mit *SQL* die bekannteste Sprache im Umfeld relationaler DBMS eingeführt.
Beispiel einer SQL-Anfrage:
```
SELECT FNAME, BDATE FROM EMPLOYEE ORDERED BY BDATE
```

Beispiel 6: Die Datenbanksprache SQL

3-Schema-Architektur

Abbildung 1: *3-Schema-Architektur*



(click on image to enlarge!)

Die Abbildung 1 stellt die 3-Schema-Architektur dar, welche die drei zentralen Modelltypen des Datenbankentwurfsprozesses miteinander in Beziehung setzt.

Definition 10: *Konzeptuelles Schema*
Ein konzeptuelles Schema ist ein Modell, welches den relevanten Realitätsausschnitt (auch *Miniwelt*, *Diskursbereich* oder *Universe of Discourse* genannt) in Struktur und Inhalt beschreibt.

Definition 11: *Logisches Schema*
Ein logisches Schema ist ein Modell, welches paradigmenspezifisch aus einem konzeptuellen Schema abgeleitet wurde.

Die Definition von Relationen als mathematisches Konzept zur Datenstrukturierung stellt ein logisches Schema dar.
Beispielsweise die Festlegung der Struktur der *Person* oder des *Studenten* in Beispiel 3.

Beispiel 7: Relationen sind ein logisches Schema

Definition 12: *Physisches Schema*
Ein physisches Schema ist ein implementierungsspezifisches Modell, welches aus einem logischen Schema abgeleitet wurde.

Definition 13: *Datenunabhängigkeit*
Die Formulierung einer Modellschicht (d.h. eines Datenmodells) ist von den darunter- bzw. darüberliegenden Modellschichten dann datenunabhängig, wenn Änderungen in den „umgebenden“ Modellschichten sich nicht auf die betrachtete Modellschicht auswirken.

Der Vorgang der *Ableitung* zwischen den verschiedenen Modelltypen der 3-Schema-Architektur sollte hierbei idealerweise (aus Gründen der Überprüfbarkeit, Nachvollziehbarkeit, Wiederholbarkeit und Qualitätssicherung) durch einen deterministischen Algorithmus erfolgen.

Die Abbildung 1 zeigt rechts neben den Modelltypen symbolhaft typische graphische Veranschaulichungen der jeweiligen Modellausprägungen.

1.2 Anforderungen an Datenbanksysteme

Allgemein: Speicherung, Verwaltung und Kontrolle der Daten sowie Organisation des u.U. gleichzeitig erfolgenden Zugriffs.
Spezieller:

- Redundanzfreie Datenspeicherung.
  Von dieser Forderung kann bewußt aus Gründen der Geschwindigkeitsoptimierung abgewichen werden.
- Gewährleistung von Integritätsbedingungen und Einhaltung von Regeln.

- Daten-Programm-Unabhängigkeit.

Wünschenswerte Eigenschaften:

- Leistungsfähigkeit
- Skalierbarkeit
- Benutzerfreundlichkeit
- Flexibilität
- ... spezifische Anforderungen, die sich aus der Anwendungssituation ergeben

1.3 Typen von Datenbankmanagementsystemen

Datenbanken werden heute vielfälltig in Wirtschaft, Technik und Wissenschaft eingesetzt.
Für verschiedene Anwendungsgebiete und Strukturen der verwalteten Daten haben sich daher spezifische DBMS-(Unter-)Typen herausgebildet, die diese Anwendungsfelder besonders gut unterstützen:

- **Deduktive Datenbanken**
  Ein um eine Menge von Regeln (Deduktionskomponenten) erweitertes Datenmodell welches logische Schlüsse auf Basis der hinterlegten Fakten ziehen kann.
- **Multimedia Datenbanken**
  Ein System, welches sich besonders zur Verwaltung großer Bild-, Audio- oder Videodaten eignet.
- **Objektdatenbanken**
  Ein System zur Speicherung von Strukturen gemäß dem logischen Objektmodell.
- **Geographische Datenbanken**
  Ein System das sich besonders zur Verwaltung geographischer Daten (z.B. Landkarten) eignet.
- **XML-Datenbanken**
  Ein System zur Speicherung gemäß dem logischen Modell des XML Information Sets.
- **Aktive Datenbanken**
  Ein System zur selbständigen Reaktion auf externe Ereignisse.
- **Temporale Datenbanken**
  Ein System, welches neben den reinen Datenbeständen auch die Zeit des Datenzustandes mitverwaltet.

Exkurs

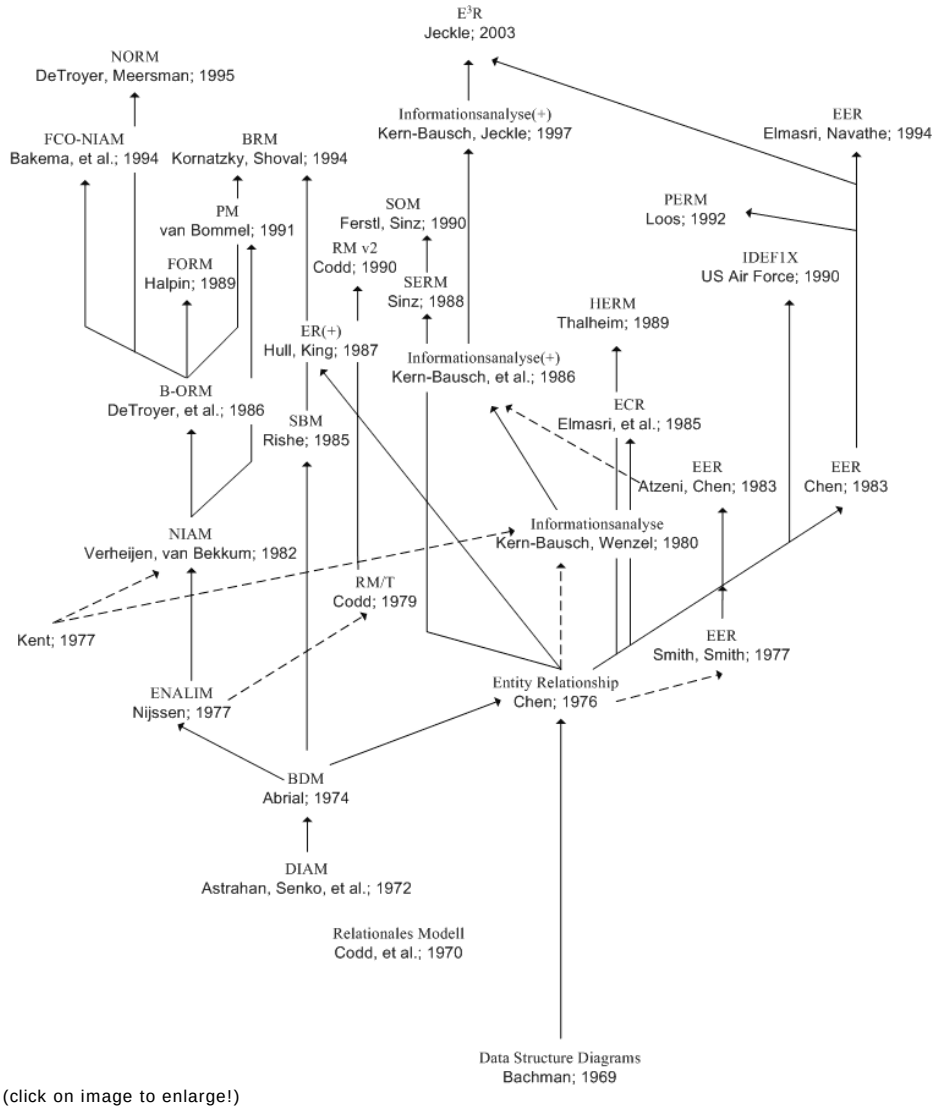
Erste Gehversuche mit dem RDBMS MySQL

2 Entwurf einer Datenbank

2.1 Graphischer Entwurf des konzeptuellen Schemas mit dem Entity-Relationship Modell

Seit der wirkungsmächtigen Erstveröffentlichung des *Entity Relationship Modells* (ERM) durch P. Chen 1976 kommt dieser Modellierungssprache zur Erstellung des konzeptuellen Schemas die uneingeschänkt größte Bedeutung in der Praxis zu.
In der Folgezeit wurden verschiedene Weiterentwicklungen des ursprünglichen ERM vorgeschlagen, die das Originalmodell in verschiedenen Richtungen erweitern. Hierunter fallen die Einführung von Konstrukten zur Abbildung hierarchischer Beziehungen ebenso wie Primitive zur Darstellung von Aggregationsbeziehungen.
Die Graphik der Abbildung 2 zeigt eine Auswahl verschiedener Entwicklungen rund um das initiale ERM sowie einige zentrale Weiterentwicklungen. Innerhalb der Abbildung ist unterhalb des Namens der Modellierungssprache (sofern vorhanden, bei Weiterentwicklungen ohne eigenständige Namensgebung ist zur Unterscheidung vom Vorgängermodell ein geklammertes Pluszeichen angetragen) der Autor sowie das Jahr der Erstveröffentlichung dargestellt.

Abbildung 2: *Entwicklungslinien des ER-Modells*



(click on image to enlarge!)

Im oberen Bereich der Abbildung ist das *Semantically Enriched Extended Entity Relationship Model* (E^3R) dargestellt, welches im Rahmen dieser Vorlesung behandelt wird.
Es stellt eine kompatible Entwicklung dar, die versucht die datenorientierten Aspekte der ER-Nachfolgemodelle mit denen der semantischen Datenmodellierung zu vereinigen.

Die Grundkonzepte des E^3R-Modells sind:

Definition 14: *Entität*
Eine Entität ist ein eindeutig identifizierbares und daher wohlunterscheidbares „Ding".
Im graphischen E^3R-Modell werden Entitäten durch benannte Rechecke dargestellt, die im Zentrum den unterstrichenen Namen der Entität, gefolgt vom -- in Klammern angegebenen -- Namen des Entitätstypen, tragen.

Anmerkung: Der Begriff *Ding* wird hierbei in seiner Bedeutung als Synonym von *Seiendes, Gegenstand* oder *Objekt* gebraucht. Die philosophische Terminologie detailliert den Begriff zusätzlich hinsichtlich seiner Verwendung zur Beschreibung raumzeitlicher Gegenstände mit festgelegten charakteristischen (substantiellen) und zufällig anhaftenden Eigenschaften (Akzidenzien) aus.

- Die Tafel direkt vor ihnen.
- Sie selbst.
- Dieser Hörsaal.

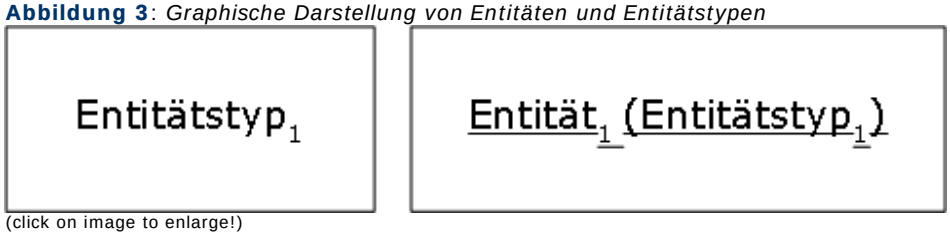
Beispiel 8: Beispiele für Entitäten

Definition 15: *Entitätstyp*
Ein Entitätstyp ist eine ungeordnete und duplikatfreie Sammlung von als logische zusammengehörig betrachteten Entitäten.
Im graphischen E^3R-Modell werden Entitätstypen durch benannte Rechtecke dargestellt. Der Name eines Entitätstyps muß dabei schemaweit eineindeutig sein.

- Tafel.
- Person.
- Student.

Beispiel 9: Beispiele für Entitätstypen

Abbildung 3: *Graphische Darstellung von Entitäten und Entitätstypen*



(click on image to enlarge!)

Soll ein Hinweis auf eine spätere physische Realisierung (d.h. die gewählte Form der Abspeicherung von Entitäten in der Datenbank) gegeben werden, so kann einem Entitätstypen ein Repräsentationstyp zugeordnet werden, bzw. einer Entität eine Repräsentation.

Definition 16: *Repräsentationstyp*
Ein Repräsentationstyp führt einen physischen Typ in das konzeptuelle Schema ein, der zur technischen Implementierung eines durch ihn annotierten Entitätstypen herangezogen werden kann.
Als Repräsentationstypen sind beliebige atomare (d.h. in ihrer Semantik nicht weiter verlustfrei zerlegbare) Datentypen eines logischen oder physischen Modells zugelassen.

- Integer
- Datum
- Money
- String

Beispiel 10: Beispiele für Repräsentationstypen

Definition 17: *Repräsentation*
Eine Repräsentation ist eine Ausprägung genau eines Repräsentationstypen.

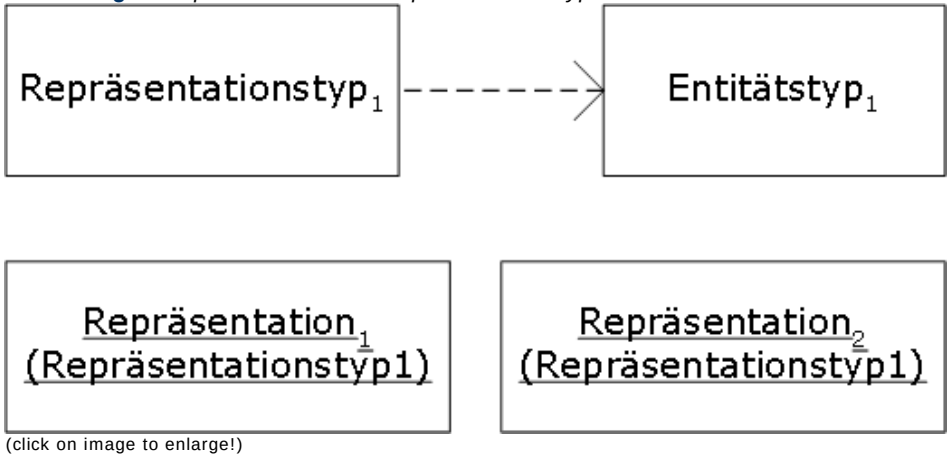
Der zugehörige Repräsentationstyp ist in Klammern angegeben.

- 42 (Integer)
- 2004-06-08 (Datum)
- €99,95 (Money)
- "Hallo Welt!" (String)

Beispiel 11: Beispiele für Repräsentationen

Die graphische Darstellung erfolgt durch benannte Rechtecke. Repräsentationen werden unterstrichen mit der geklammerten nachfolgenden Angabe des Repräsentationstypen dargestellt. Repräsentationstypen werden durch eine gerichtete Kante mit unterbrochener Linienführung mit dem durch sie repräsentierten Entitätstypen verbunden.

Abbildung 4: *Repräsentation und Repräsentationstyp*

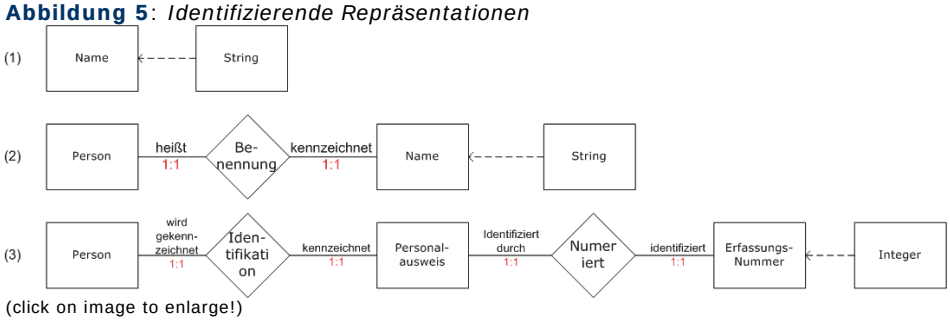


(click on image to enlarge!)

Das Beispiel der Abbildung 5 zeigt verschiedene Beispiele für die Verknüpfung von Entitätstypen

mit ihren zugehörigen Repräsentationstypen.
Im Teilbeispiel (1) wird der Entitätstype `Name` unmittelbar durch den Repräsentationstypen `String` repräsentiert, d.h. die spätere physische Realisierung des Entitätstypen `Name` wird durch den Datentyp `String` erfolgen.
Teilbeispiel (2) zeigt eine transitive Repräsentation (genaugenommen eine transitive Repräsentation erster Ordnung). Hier ist der Entitätstyp `Person`, welcher selbst über keinen Repräsentationstypen verfügt, in eindeutiger Weise (d.h. über einen Assoziationstyp der ausschließlich über Kardinalitsintervalle von `1:1` verfügt (nach Maßgabe der Anmerkung zur Struktur der Kardinalitätsintervalle kann es sich daher nur um einen binären Assoziationstypen handeln)) mit dem Entitätstypen `Name` verknüpft, der über die Repräsentation `String` verfügt.
Abschließend zeigt das Teilbeispiel (3) die transitive eindeutige Assoziierung des Entitätstypen `Person` mit dem Entitätstypen `Personalausweis` durch den Assoziationstypen `Identifikation`, wobei `Personalausweis` seinerseits in eindeutiger Weise mit der durch `Integer` repräsentierten Erfassungsnummer assoziiert ist.

Abbildung 5: *Identifizierende Repräsentationen*



(click on image to enlarge!)

Definition 18: *Assoziation*
Eine Assoziation ist eine benannte n-äre Beziehung (n>1) zwischen Entitäten.
Die Semantik jeder durch eine Assoziation verbundenen Entität wird durch Angabe einer innerhalb einer Assoziation für jede verbundene Entität eindeutigen Rolle konkretisiert.
Im graphischen E^3R-Modell wird eine Assoziation durch eine Raute dargestellt, die durch ungerichtete Kanten mit Entitäten verbunden ist. Im Zentrum wird der Name der Assoziation, gefolgt vom in Klammern geschriebenen Namen des Assoziationstypen plaziert. Zusätzlich sind die beiden Namen unterstrichen dargestellt.

Definition 19: *Assoziationstyp*
Ein Assoziationstyp ist eine duplikatfreie ungeordnete Sammlung von logisch als zusammengehörig betrachteten Assoziationen.
Jede zu einem Assoziationstypen beitragende Rolle wird durch ein Kardinalitätsintervall ergänzt.
Im graphischen E^3R-Modell wird ein Assoziationstyp durch eine Raute dargestellt, die durch ungerichtete Kanten mit Entitätstypen verbunden ist. Im Zentrum des Assoziationstypen wird sein schemaweit eineindeutiger Name plaziert.

- Arbeitsverhältnis.
- Ehe.
- Verwandschaft.

Beispiel 12: Beispiele für Assoziationstypen

Definition 20: *Kardinalitätsintervall*
Ein Kardinalitätsintervall legt die Anzahl derjenigen Entitäten fest, die mit einer die Rolle einnehmenden Entität zu einem Zeitpunkt innerhalb einer Assoziation verbunden sein können.
Das Intervall wird in der Schreibweise „i:j“ angegeben, wobei *i* eine beliebige natürliche Zahl oder die Null ist und *j* eine beliebige natürliche Zahl oder das Symbol *n* ist. Zusätzlich gilt: i<=j.

- 0:1.
- 3:7.
- 0:n.
- 1:n.
- 99:n.

Beispiel 13: Beispiele für Kardinalitätsintervalle

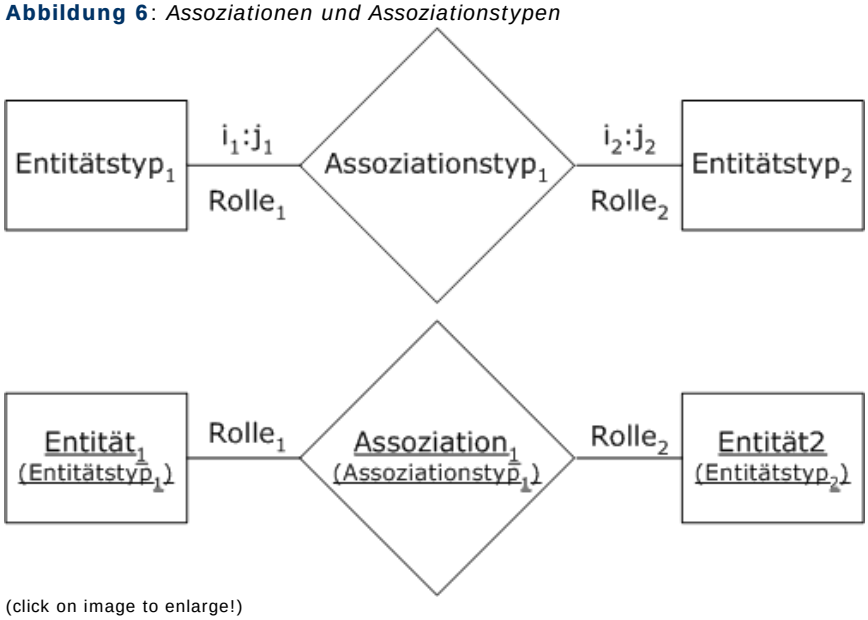
Ungültig hingegen sind:

- 7:0 (Obergrenze kleiner als Untergrenze).
- -5:7 (-5 ist keine natürliche Zahl.)

- n:8 (*n* ist nicht als Untergrenze erlaubt.)

Allgemein gilt: Für *n*-äre Assoziationstypen gilt die Einschränkung, daß die Maximalkardinalität die ein Entitästyp zu einem *n*-ären Assoziationstypen beitragen darf größer gleich *n*-1 ist.

Abbildung 6: *Assoziationen und Assoziationstypen*



(click on image to enlarge!)

Zentrales Konzept des E^{3}R-Modells ist die Idee der Rolle, welche als hauptinformationstragendes Konstrukt fungiert:

Definition 21: *Rolle*
Eine Rolle die durch einen Entitätstypen innerhalb eines Assoziationstypen eingenommen wird charakterisiert die konkrete Verwendung von Entitäten des gegebenen Typs im Kontext der Assoziationen die zum betrachteten Assoziationstyp zusammengefaßt werden.

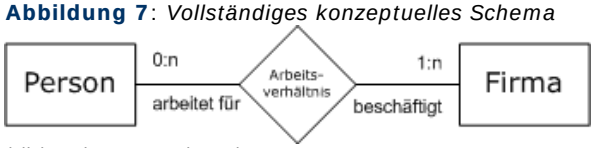
Anmerkung: Für den relationalen Datenbankentwurf ist es notwendig, daß jeder im konzeptuellen Schema modellierte Entitätstyp entweder über einen Repräsentationstyp verfügt oder über eine *Namenskonvention*, d.h. eine binäre Assoziationstyp deren Kardinalitätsintervalle ausschließlich auf 1:1 festgelegt sind, die den Entitätstyp direkt oder transitiv mit einem mit Repräsentation versehenen Entitätstypen verbindet.

Gleichzeitig wird durch die Rolle der Brückenschlag zwischen natürlicher Sprache und formaler graphischer Darstellung im E^{3}R-Modell ermöglicht.
So lassen sich die Sätze

- Jede Person arbeitet optional für mehrere Firmen.
- Jede Firma beschäftigt ein oder mehrere Personen.

in das nachfolgende konzeptuelle Schema überführen:

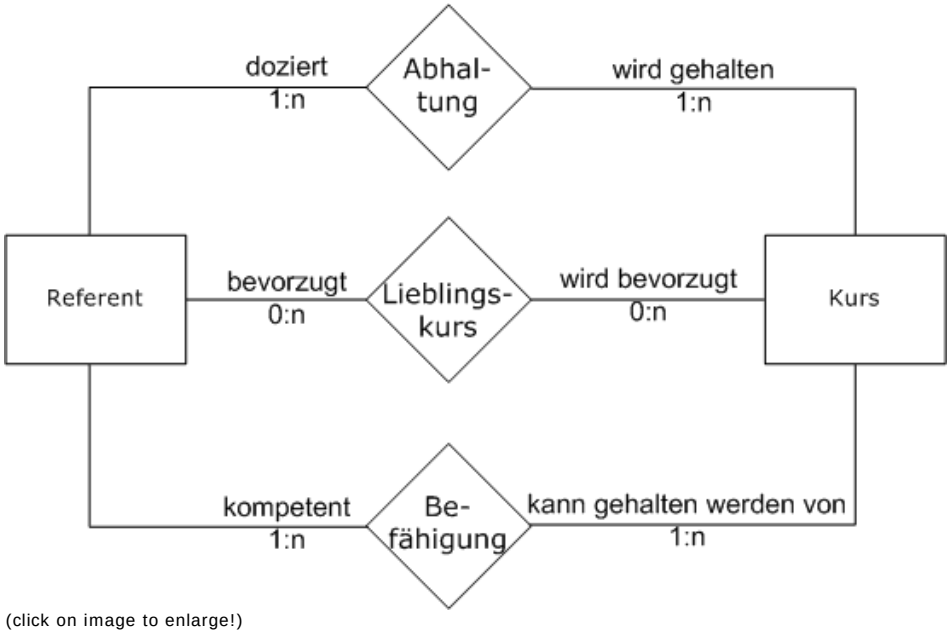
Abbildung 7: *Vollständiges konzeptuelles Schema*



(click on image to enlarge!)

Zusätzlich zeigt das konzeptuelle Schema der Abbildung 8 die Mächtigkeit des Rollenkonzepts zur Darstellung verschiedener Informationszusammenhänge.
So enthält das abgebildete konzeptuelle Schema die drei verschiedenen Assoziationstypen `Abhaltung`, `Lieblingskurs` und `Befähigung` welche ausschließlich Rollen enthalten die durch die beiden dargestellten Entitätstypen `Referent` und `Kurs` gespielt werden.

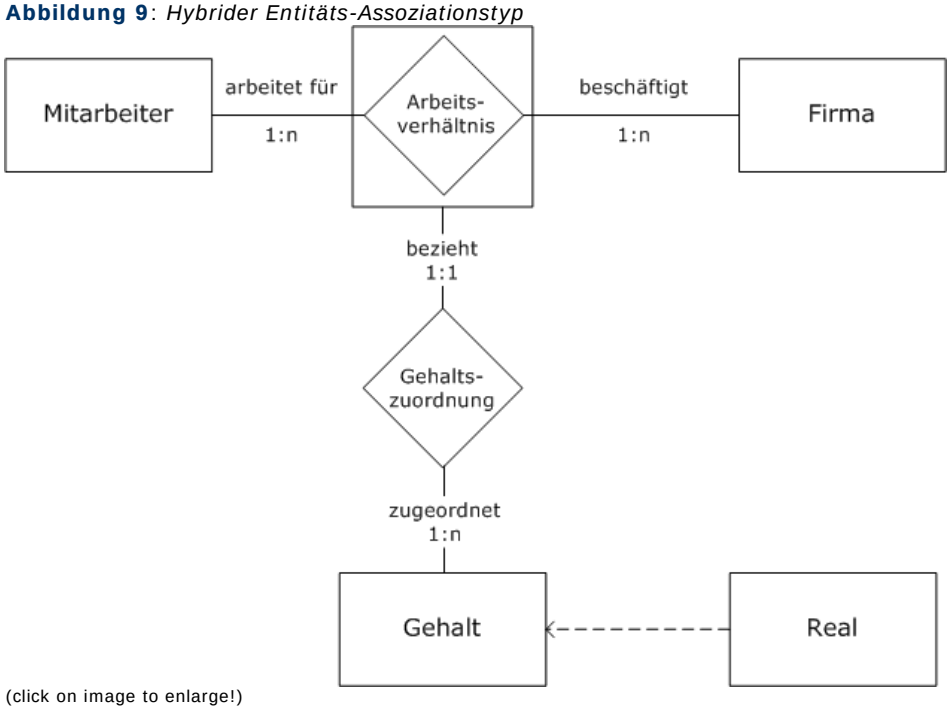
Abbildung 8: *Verschiedene Rollen*



(click on image to enlarge!)

Definition 22: *Hybrider Entitäts-Assoziationstyp*
Ein hybrider Entitäts-Assoziationstyp vereinigt das Sprachelement des Entitätstyps und des Assoziationstyps in sich und bewahrt die Semantik beider Konstrukte.

Abbildung 9: *Hybrider Entitäts-Assoziationstyp*

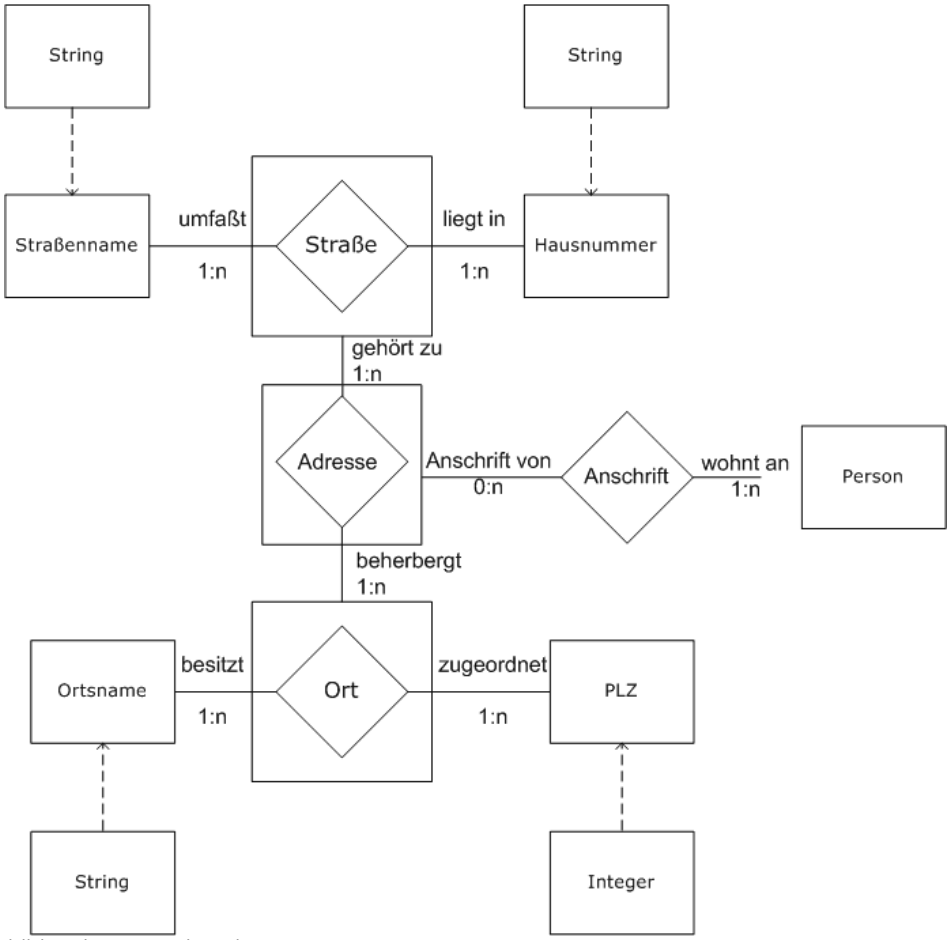


(click on image to enlarge!)

Abbildung Abbildung 10 stellt die Informationsstruktur einer Adresse dar.
Dabei zeigt das konzeptuelle Schema die Verwendung der hybriden Entitäts-Assoziationstypen.
So können jedem `Straßennamen` beliebig viele `Hausnummern` zugeordnet werden und umgekehrt.
Jeweils zwei dieser Angaben zusammen bilden die `Straße`.
Jedem `Ortsnamen` kann über mehrere `Postleitzahlen` verfügen, ebenso kann dieselbe Postleitzahl mehreren gleich benannten Orten zugeordnet werden (Beispiel: Ortsteile).
`Postleitzahl` und `Ortsname` zusammen bilden den `Ort`.
Aus der Kombination von `Straße` und `Ort` wird eine `Adresse` gebildet. Dabei kann jede `Straße` (=Kombination aus `Straßenname` und `Hausnummer`) mehreren Orten (=Kombination aus `Ortsname` und `Postleitzahl`) und umgekehrt zugeordnet sein.

Das Beispiel unterstreicht die alleinige Bildbarkeit hybrider Entitäts-Assoziationstypen beim vorliegen von Kardinalitätsintervallen, die alle über ein Maximum größer 1 verfügen.
Vgl. hierzu Aussagen der Anmerkung zur Bildung von Kardinalitätsintervallen

Abbildung 10: *Informationsstruktur Adresse*

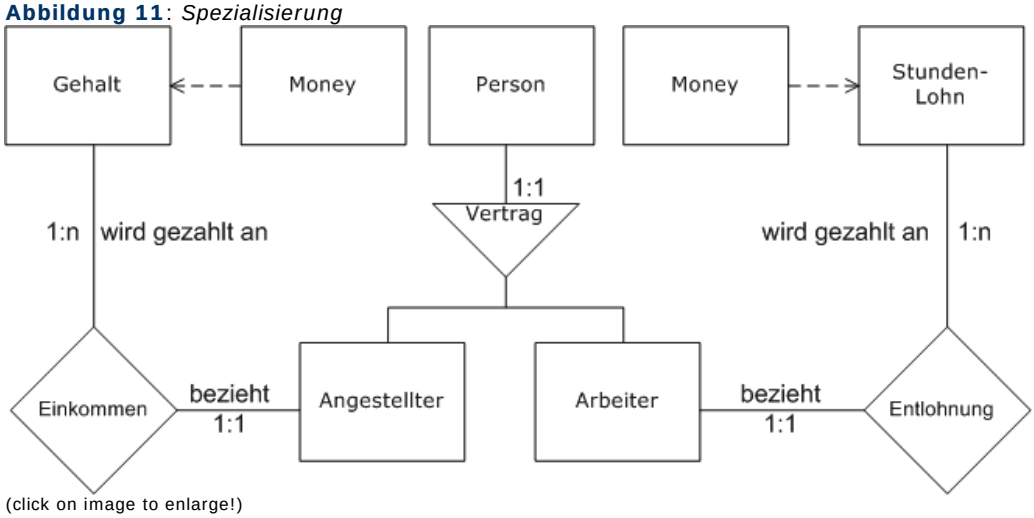


(click on image to enlarge!)

Weiterführende Konzepte

Definition 23: *Spezialisierungsassoziationstyp*
Ein Spezialisierungsassoziationstyp ist eine duplatfreie ungeordnete Sammlung von logisch als zusammengehörig betrachteten Assoziationen.Zu den in der Menge enthaltenen Assoziationen tragen der zu spezialisierende Entitätstyp (der sog. *Super-* oder *Obertyp*) und der spezialisierende (entsprechend als sog. *Sub-* oder *Untertyp* bezeichnet) Rollen bei.
Die Rolle des Supertyps ist hierbei auf *wird spezialisiert zu* fixiert, als Kardinalitätsintervalle sind ausschließlich `0:1`, `0:n`, `1:1` und `1:n` zulässig.
Die Rolle des Subtyps ist auf *ist Spezialisierung von* mit dem Kardinalitätsintervall `1:n`fixiert.
Jeder Spezialisierungsassoziationstyp wird durch ein Distinktionsmerkmal charakterisiert, das expliziert hinsichtlich welchen Merkmals die Spezialisierung gebildet wird.
Die Verknüpfung durch einen Spezialisierungsassoziationstyp bewirkt, daß alle Assoziations- und Repräsentationstypen, die für den Supertyp definiert sind auch automatisch für alle Subtpyen definiert werden.

Abbildung 11: *Spezialisierung*



(click on image to enlarge!)

Die Abbildung 11 zeigt die Spezialisierung des Entitätstypen `Person` hinsichtlich des Distinktionsmerkmals `Vertrag`. Dabei gilt: Jede Person kann nur genau einmal (`1:1`) hinsichtlich

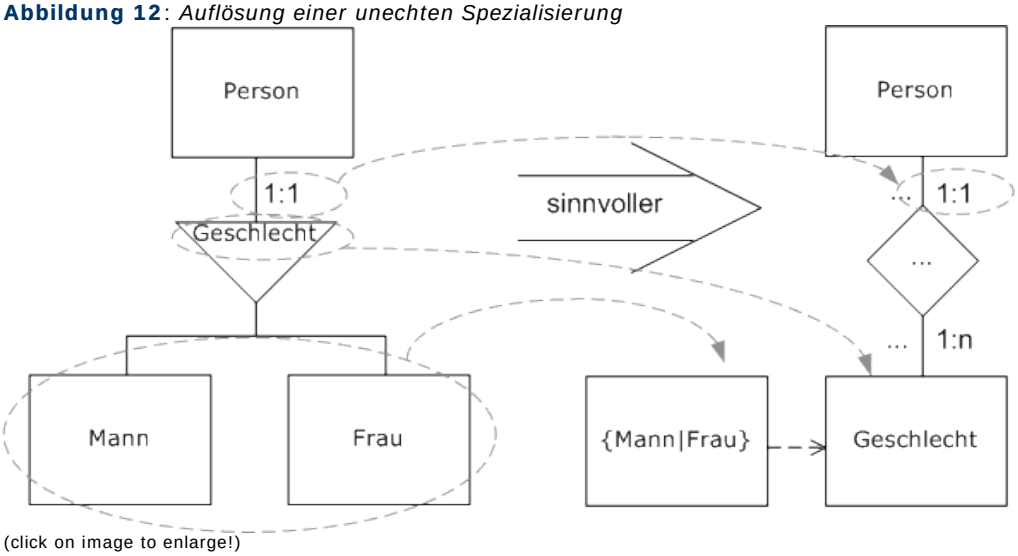
ihres (Arbeits-)Vertrages zu `Angestellter` oder `Arbeiter` spezialisiert werden.

Abbildung 11 zeigt ferner, daß die spezialisierten Entitätstypen `Angestellter` und `Arbeiter` über zusätzliche, d.h. für den Obertyp `Person` nicht definierte, Eigenschaften (`Gehalt` bzw. `Stundenlohn`) verfügen.

Zur Pragmatik des Spezialisierungsassoziationstyps:
Spezialisierungsassoziationstypen sollten ausschließlich dann eingesetzt werden, wenn es sich um „echte“ Spezialisierungen handelt. Eine *echte Spezialisierung* liegt immer dann vor, wenn jeder spezialisierte Entitätstyp über Assoziationstypen verfügt, die der allgemeinere Obertyp nicht besitzt.

Gilt dieses Kriterium nicht, d.h. können für die spezialisierten Entitätstypen keine zusätzlichen Eigenschaften gebildet werden, dann solte die Spezialisierung *nicht vorgenommen* werden.
In diesem Falle bietet sich die Überführung der unnötigen Spezialisierung eine Eigenschaft des (vermeintlichen) Obertypen an. Zusätzlich sollte ein Entitätstyp zur Aufnahme derjenigen Information gebildet werden, für welche die Darstellung als Spezialisierungsassoziationstyp intendiert war.
Dieser neue Entitätstyp kann geeignet mit einem Repräsentationstyp versehen werden, um die unterscheidende (distingierende) Information darzustellen.
Die einzelnen Schritte sind in Abbildung 12 beispielhaft zusammengestellt.

Abbildung 12: *Auflösung einer unechten Spezialisierung*



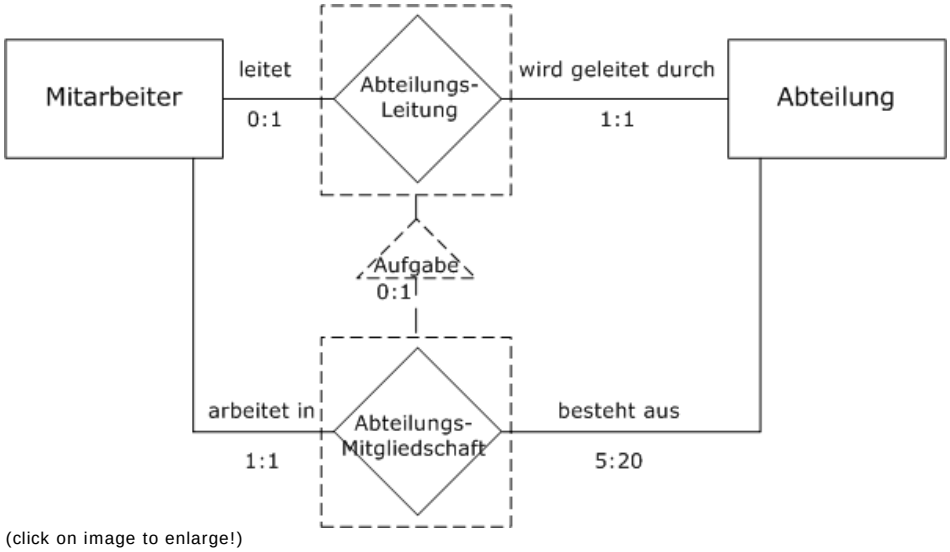
(click on image to enlarge!)

Definition 24: *Metainformation*
Informationsanteile eines Modells, die nicht direkt in das logische Schema übernommen werden, sondern in konsistenzgarantierende Regeln oder Applikationscode abgebildet werden.

Die Abbildung 13 veranschaulicht die Nutzung des Spezialisierungsassoziationstyps zur Formulierung von Metainformation. Im Beispiel wird gefordert, daß jeder `Abteilungsleiter` auch gleichzeitig als Mitarbeiter der durch ihn geleiteten Abteilung erfaßt sein muß.

Hinweis: Metainformation muß nicht zwingend semantisch irreduzibel erfaßt werden, wie die --- eigentlich illegale Bildung der beiden hybriden Entitäts-Assoziationstypen `Abteilungsleitung` und `Abteilungsmitgliedschaft` zeigt.

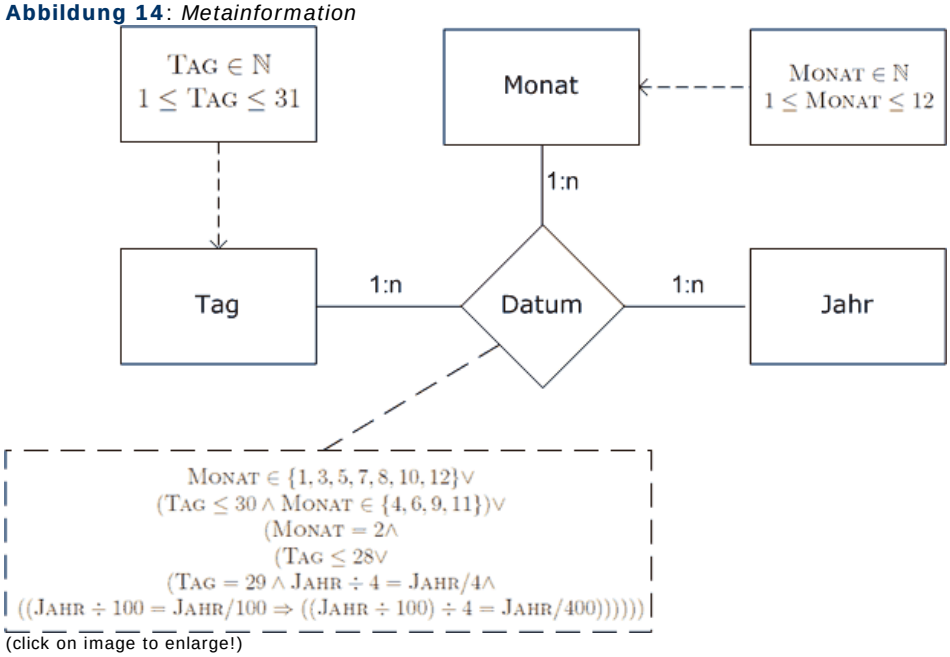
Abbildung 13: *Metainformation*



(click on image to enlarge!)

Das konzeptuelle Schema der Abbildung 14 zeigt ein Beispiel für konsistenzgarantierende Metainformation, die nicht durch E^{3}R-Syntax ausdrückbar ist und daher in textueller Form annotiert wird.

Abbildung 14: *Metainformation*



(click on image to enlarge!)

Phasenmodell der Erstellung eines konzeptuellen Schemas mit E^{3}R

E^{3}R ist eine Notation und Methode zur Entwicklung des konzeptuellen Schemas für jede beliebige Art von Kommunikationssituationen.

Phase 0 ist die Vorbereitungsphase, die von der Idee, ein E^3-Schema für einen Realitätsausschnitt zu erstellen, über die Auswahl der Beteiligten bis zu ihrer Ausbildung in den Techniken zur Darstellung von Information und in der Vorgehensweise der Analyse reicht.

Es folgt die Festlegung des Informationsbereichs (**Phase 1**). Hier werden die für den gewünschten Anwendungsbereich relevanten Entitäten und Assoziationen gesammelt und zu Entitäts- und Assoziationstypen zusammengefaßt. Eine große Hilfe dabei sind verbale Beschreibungen der in der Datenbank zu verwaltenden Information, kommentierte Listen mit Daten des betrachteten Informationsbereichs oder ähnliches. Das Ergebnis ist eine erste, grobe Struktur der relevanten Information.

Diese Struktur wird in **Phase 2** immer weiter verfeinert, wobei man für jeden Entitätstyp entweder direkt oder transitiv eine Repräsentation definiert.
Dann werden alle relevanten Eigenschaften, die eine Entität eines Typs haben kann, in Form von semantisch irreduzibel formulierten Assoziationstypen beschrieben. Dabei treten erfahrungsgemäß neue, zuvor nicht berücksichtigte Entitätstypen auf.

Deshalb wird die Phase 2 solange inkrementell iteriert, bis keine neuen Entitätstypen mehr

identifiziert werden zu denen noch Repräsentation zu definieren oder durch Assoziationstypen anzubinden sind.

Bis zu diesem Punkt standen strukturelle, formale Gesichtspunkte im Vordergrund. In **Phase 3** treten diese zurück; nun stehen semantische Gesetzmäßigkeiten im Vordergrund, soweit diese nicht bereits in den Phasen 1 und 2 erkannt und behandelt worden sind.

Ziel der Phase 3 ist es, die bis dato erstellte Informationsbeschreibung geeignet zu ergänzen um auch alle nicht durch die E^3R-Notation darstellbaren Konsistenzregeln zu erfassen.
Zusätzlich kann die E^3R-Notation zur Formulierung von Metainformation auf einer höheren Modellebene angewendet werden.

Hinweis: Es kann beim Erstellen des konzeptuellen Schemas durchaus vorkommen, daß sich das Ergebnis der vorausgegangenen Phase als unvollständig herausstellt. In diesem Fall ist es unbedingt notwendig, in diese Phase zurückzukehren und dann mit dem korrigierten Ergebnis dieser Phase weiterzuarbeiten. Dies ist kein Wegwerfen der bisher geleisteten Arbeit, denn meist genügen einige wenige Streichungen und Ergänzungen.
Bleibt diese Regel unberücksichtigt, so nimmt begibt man sich der Möglichkeit wichtige Eigenschaften der Information im konzeptuellen Schema eindeutig festzuhalten. Dabei spricht das Verhältnis zwischen der gewonnen Exaktheit und dem zusätzlichen Aufwand sehr zugunsten der exakten und sauberen Lösung.

Resultat der korrekten Anwendung des Phasenmodelles ist ein *vollständiges konzeptuelles Schema* als Voraussetzung der Umsetzbarkeit in beliebige logische Strukturen.

Definition 25: *Vollständiges konzeptuelles Schema*
Ein vollständiges konzeptuelles Schema ist ein E^3R-Schema in dem alle Entitäts- und Assoziationstypen, sowie alle Rollen benannt sind. Darüberhinaus ist jede Rolle mit einem korrekten Kardinalitätsintervall versehen, sowie jedem Entitätstypen direkt oder transitiv ein Repräsentationstyp zugeordnet.
Sofern Metainformation existiert, ist diese auch in adäquater Weise dargestellt.

Fallstudie: Fächerdatenbank

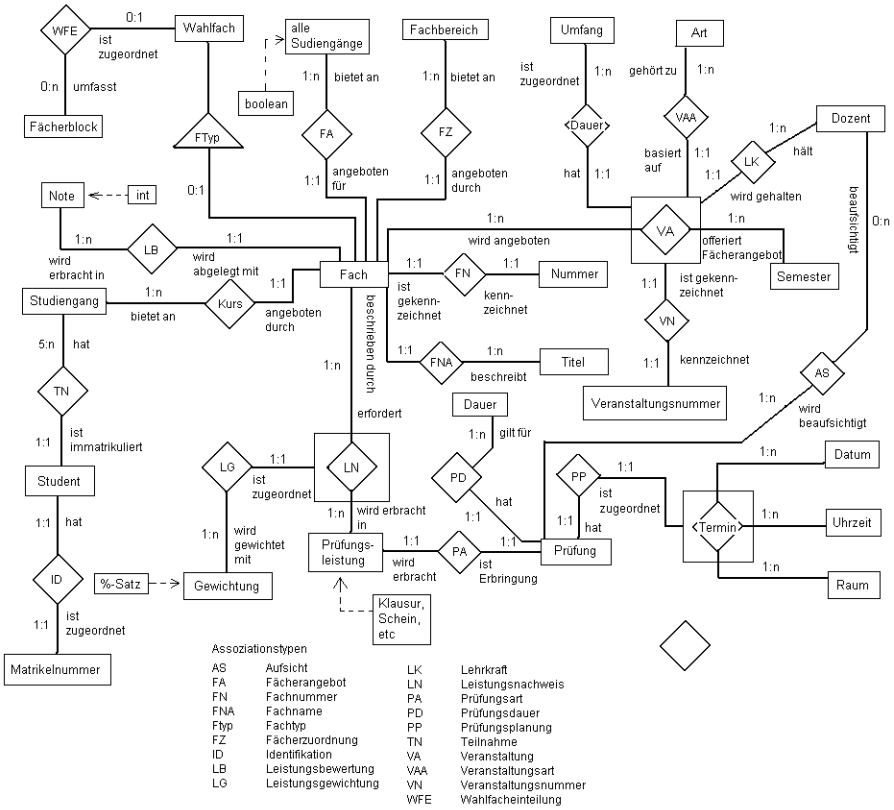
Der Fachbereich möchte die Belegung der Fächer in einer Datenbank abspeichern; hierfür gelten folgende semantische Regeln:

Die vorgesehenen Fächer haben eine feste Nummer, die sich niemals ändert sowie eine einen längeren Titel. Zusätzlich sind sie von einem Fachbereich entweder für einen speziellen Studiengang (z.B. *WIB*) oder allen Studiengängen angeboten, dann gilt die Zuordnung *FH*. Gleichzeitig kann jedes Wahlfach einem Fächerblock (z.B. *Consulting* oder *Informatik*) zugeordnet sein.

Die in einem bestimmten Semester angebotenen Fächer erhalten eine Veranstaltungsnummer, die nur für dieses Semester gilt. Dazu wird der jeweilige Dozent angegeben und die Art der Lehrveranstaltung (*Vorlesung*, *Seminar*, *Praktikum* etc.) sowie ihr Umfang in Semesterwochenstunden.
Die Notenbildung in jedem Fach kann durch eine oder mehrere Prüfungsleistungen erfolgen (z.B. *Leistungsnachweis*, *Schein*, *Prüfung*, etc.), die in unterschiedlichen Prozentsätzen gewichtet werden. Jede Prüfung findet zu einem festgelegten Datum in einem Raum zu einer Uhrzeit statt und wird durch mindestens einen Dozenten beaufsichtigt. Zusätzlich soll die Dauer der Prüfungsleistung vermerkt werden.

Ein Student ist durch eine eindeutige Matrikelnummer gekennzeichnet. Zusätzlich wird sein Studiengang gespeichert.

Abbildung 15: *Konzeptuelles Schema der Fallstudie*



(click on image to enlarge!)

2.2 Ableitung logischer Relationenstrukturen

Erweiterung der Grundbegriffe des Relationenmodells

Definition 26: *Superschlüssel*
Ein Superschlüssel *SK* ist eine nicht-leere Teilmenge von Attributen einer Relation für die gilt, daß zwei verschiedene Tupel t_1 und t_2 dieser Relation keine gleiche Wertbelegung aufweisen.

Der Superschlüssel definiert damit eine Eindeutigkeitkeitseinschränkung, nach der zwei Tupel allein über die Betrachtung der im Superschlüssel zusammengefaßten Attribute unterscheidbar sind.

Gegeben sei die Relation *Mitarbeiter*:

Vorname	Nachname	Geburtsdatum	Persausweisnummer
Xaver	Obermüller	1970-03-04	134975459
Rosi	Hinterhuber	1973-06-02	781367519
Rosi	Obermüller	1963-11-03	783148384
Hans	Hinterhuber	1970-03-04	977554422

Mögliche Superschlüssel dieser Relation sind:

- (Vorname, Nachname, Personalausweisnummer)
- (Nachname, Geburtsdatum, Personalausweisnummer)
- (Geburtsdatum, Personalausweisnummer)
- ...

Beispiel 14: Beispiele für Superschlüssel

Definition 27: *Schüssel*
Ein Schlüssel *K* ist ein Superschlüssel, der sofern man ein Attribut aus ihm entfernt nicht mehr eindeutigkeitseinschränkend wirkt.

Der einzige Schlüssel der Relation Mitarbeiter ist `Personalausweisnummer`.

Beispiel 15: Beispiele für Schlüssel

Anmerkungen:

- Die Menge aller Attribute einer Relation ist immer Superschlüssel.
- Jeder Schlüssel ist auch ein Superschlüssel.
  Der Umkehrschluß gilt nicht, da Schlüssel eine schärfere Forderung darstellt.
- Jeder Schlüssel ist zwingend eine minimal identifizierende Attributkombination.

Häufig tritt es in der Praxis auf, daß sich in einer Relation mehr als ein Schlüssel finden läßt. Jeder dieser möglichen gleichwertigen Schlüssel wird daher als *Schlüsselkandidat* bezeichnet.

Gegeben sei die Relation *Lagerverwaltung*:

Lagerplatz	Produktnummer	Produktname	Menge
7952	7946	Wusch Superfein	3
7412	9854	Blitzbank Extra	5
7894	6542	Maiengrün natur	7
9461	8954	Gelber Gigant	5

Die Relation enthält folgende Schlüsselkandidaten.

- `Lagerplatz`
- `Produktnummer`
- `Produktname`

Beispiel 16: Beispiele für Superschlüssel

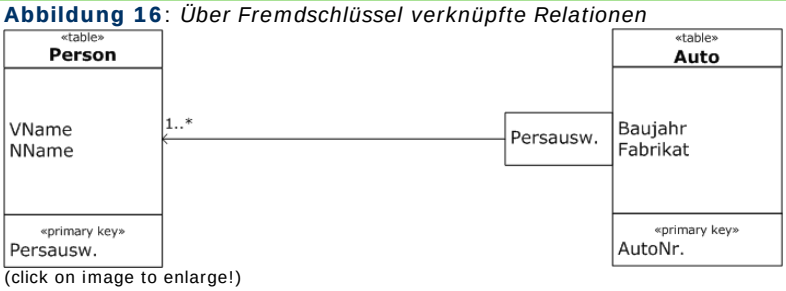
Definition 28: *Primärschüssel*
Ein Primärschlüssel *P* ist ein Schlüssel, der als identifizierendes Merkmal ausgewählt wurde.

In der graphischen Darstellung der Demo-DB sind die Primärschlüssel durch Unterstreichung der beitragenden Attribute hervorgehoben.

Zur Wahrung der Konsistenz innerhalb einer relationalen Datenbank wird üblicherweise u.a. das Mittel der *referentiellen Integrität* eingesetzt, um gleicher Wertinhalte in Attributen (derselben oder verschiedener Relationen) aufeinander abzustimmen.

Definition 29: *Referentielle Integrität*
Attributwerte einer durch referentielle Integrität verknüpften Relation müssen auch in der verknüpften Relation existieren.

Abbildung 16: *Über Fremdschlüssel verknüpfte Relationen*



(click on image to enlarge!)

Das Attribut `Persausw.` der Relation `Auto` verweist auf den Primärschlüssel gleichen Namens der Relation `Person`.
Als Konsequenz dürfen für `Persausw.` in `Auto` nur Werte definiert werden, die sich bereits im Attribut `Persausw.` von `Person` finden.

Beispiel 17: Beispiel für referentielle Integrität

Die Prüfung von Primärschlüsselwerten erfordert u.U. eine Reihe zusätzlicher Datenbankzugriffe. Zu ihrer Beschleunigung können zusätzliche Speicherbereiche, sog. *Indexe* angelegt werden.

Definition 30: *Index*
Ein Index ist ein zusätzlicher Speicherbereich der in der Datenbank verwaltet wird um den lesenden Zugriff auf einzelne Tupel zu beschleunigen.

In der Konsequenz der Beschleunigung der lesenden Zugriffe durch zusätzlichen Speicherplatz verringert sich die Geschwindigkeit der schreibenden Zugriffe (Erzeugung, Aktualisierung und Löschung) etwas.

Die Tabelle `tab` besteht aus zwei Attributen `UUID1` und `UUID2`, wobei ersteres duplikatfrei indexiert wird.

Aktion	Dauer ohne Index [sec]	Dauer mit Index [sec]
Einfügen von 10.000.000 Tupeln `INSERT INTO tab VALUES (...)`	1812	2025
Auswahl aller Tupel `SELECT COUNT(*) FROM tab WHERE UUID1<> "X"` (`UUID1` enthält niemals den Wert X daher werden alle Tupel selektiert)	2100	1800
Auswahl genau eines Tupels `SELECT UUID2 FROM tab WHERE UUID1="..."`	0,422	0,028
Aktualisierung genau eines Tupels `UPDATE tab SET UUID2="Z" WHERE UUID1="..."`	0,415	0,033
Aktualisierung keines Tupels, jedoch vollständige Durchsuchung eines Attributs. `UPDATE tab SET UUID2="Z" WHERE UUID1<>"X"`	0,395	0,014
Löschung eines Tupels `DELETE FROM tab WHERE UUID1="..."`	0,431	0,043
Löschung keines Tupels, jedoch vollständige Durchsuchung eines Attributs. `DELETE FROM tab WHERE UUID1="x"`	0,037	0,008

Beispiel 18: Geschwindigkeitsverhalten mit/ohne Index

Die Erstellung des Index nimmt, bei in der Tabelle gehaltenen 10.000.000 Tupeln 363,063 Sekunden in Anspruch.

Definition 31: *NULL-Wert*
Fehlende Attributwerte in einer Relation werden durch den gesonderten Datenbankeintrag `NULL` dargestellt.

Für die Wertbelegung `NULL` stellt das DBMS sicher, daß sie nicht mit der Ziffer 0 oder dem leeren String kollidiert.

Der Algorithmus

Zentrale Zielsetzung der Erstellung des konzeptuellen Schemas ist die Möglichkeit von ihm ausgehend unterschiedliche logische Modelle, die später in die physische Implementierungssicht abgebildet werden, ableiten zu können.
Dieser Abschnitt stellt einen Algorithmus vor, der es erlaubt aus dem mit E^3R formulierten konzeptuellen Schema logische Strukturen gemäß dem Relationenmodell abzuleiten.

Dabei operiert der Algorithmus ausschließlich auf der graphischen Repräsentation des konzeptuellen Schemas und kann daher auch von entsprechend ausgebildeten Fachexperten manuell durchgeführt werden.
Der durch den Algorithmus abgeleitete logische Datenbankentwurf orientiert sich an festgelegten Gütekriterien um einen redundanzfreien und somit anomalienfreinen Entwurf zu gewährleisten.

Schritt 1

Markiere alle Verbindungslinien (d.h. Rollen), an denen das Kardinalitätsintervall 1:1 steht.

Anmerkung: Eine ausschließliche 1:1-Markierung stellt einen logisch korrekten DB-Entwurf sicher.
Verbindungslinien, die zu Spezialisierungsassoziationstypen führen müssen nicht markiert werden.

Das zusätzliche Markieren aller 0:1-Verbindungen führt zu Performanceverbesserungen.
Bei relationalen DBMS, die fehlende Werte (NULL) zulassen führt das Markieren von 0:1-Verbindungen zu einem optimalen relationalen DB-Entwurf. Bei Implementiereungen die auch optionale Schlüsselkandidaten zulassen führt das fortgesetzte Markieren über 0:1 Verbindungen hinweg zu effizienten DB-Strukturen.

Schritt 2

1. Bilde die Zusammenhangskomponenten (bestehend aus Entitäts- und Assoziationsstypen) bezüglich der markierten Verbindungslinien.
2. Übrigbleibende (d.h. noch außerhalb von Zusammenhangskomponenten plaziert Entitäts- und Assoziationstypen bilden jeweils eine eigene Zusammenhangskomponenten, wenn gilt:
    - Falls ausschließlich alle Mitgliedschaftsintervalle an der Verbindung zwischen einem solchen Entitätstyp und irgendeinem Assoziationstyp den minimalen Wert 0 haben, ist aus diesem Entitätstyp eine eigenständige Zusammenhangskomponente zu bilden, sofern der Entitätstyp kein Repräsentationstyp oder die entsprechenden Entitäten nicht schon in einem anderen Assoziationstyp definiert sind (Kardinalitätsintervall 1:z; z>=1).
    - Repräsentationstypen werden nicht berücksichtigt.
    - Assoziationstypen bilden jeweils (als einziger Inhalt) eine eigene Zusammenhangskomponente.

Schritt 3

Treten innerhalb einer Zusammenhangskomponente Zyklen auf, so sind diese wie folgt zu behandeln:

Anmerkung: Ein Zyklus in einer Zusammenhangskomponente ist eine Folge $\{ET_1, AT_{1,2}, ET_2, AT_{2,3}, ..., ET_n, AT_{n,1}\}$ mit den Eigenschaften:

- In allen Assoziationstypen$_{i,k}$ ($1<=i, k<=n$) wird die eine Rolle von ET_i und die andere Rolle von ET_k gespielt.
  Anmerkung:Zu Untertypen führende Kanten werden behandelt wie gewöhnliche Beziehungen zu Assoziationstypen.
- Alle Verbindungslinien einer Folge sind markiert.

Zur Ableitung von Relationen müssen Zyklen aufgelöst werden:

- Falls innerhalb eines Zyklus 0:1-Markierungen vorhanden sind, werden diese gelöscht;
- falls nur 1:1-Markierungen vorhanden sind, wird eine beliebig festzusetzende Verbindungslinie gelöscht.

Schritt 4

Aus jeder Zusammenhangskomponente wird eine Relation nach folgenden Regeln:

1. Namensgebend für eine Relation ist genau einer der innen liegenden Entitätstypen. Enthält eine Zusammenhangskomponente nur einen Assoziationstyp, so bekommt die abgeleitete Relation dessen Namen.
2. Die Relation enthält je ein Attribut für jeden direkt mit einer Repräsentation versehenen Entitätstypen im Inneren der Zusammenhangskomponente. Ein Entitätstyp wird zusammen mit seinen sämtlichen Untertypen als ein Entitätstyp betrachtet, sofern die Untertypen über keine eigene Repräsentation verfügen.
3. Zusätzlich enthält die Relation für jeden Assoziationstyp im Inneren einer Zusammenhangskomponente noch je ein Attribut für jede Rolle die zu einem innenliegenden Assoziationstyp beiträgt, welche ein Entitätstyp spielt, der außerhalb der Zusammenhangskomponente liegt.
   Für einen im Inneren der Zusammenhangskomponente liegenden Assoziationstyp sind alle

Entitätstypen, zu denen eine nicht markierte Verbindungslinie führt, außerhalb.

4. Zusammenhangskomponenten, die ausschließlich aus genau einem Assoziationstyp bestehen werden in eine eigenständige Relation überführt, die für jede zum Assoziationstyp beitragende Rolle ein Attribut enthält.
   Dieses Attribut wird mit der Repräsentation des rollenspielenden Entitätstypen typisiert.
5. Jedes aus einem Entitätstyp im Inneren einer Zusammenhangskomponente abgeleitete Attribut ist Schlüsselkandidat.
   Außerdem sind alle diejenigen Attribute Schlüsselkandidaten, deren entsprechende Kardinalitätsintervalle das Maximum 1 besitzen.
6. In den restlichen Fällen sind alle Attribute zusammen Schlüsselkandidat.
7. Existiert in einer Relation mehr als genau ein Schlüsselkandidat, so ist einer unter diesen als Primärschlüssel auszuzeichnen.
8. Jeder Untertyp erbt alle Rollen, die sein Obertyp innerhalb von Assoziationstypen spielt. Zusätzlich erbt er auch die vorhandene Repräsentation des Obertyps.
9. Relationen, deren Attribute sich aus jeweils gleichen Rollen ableiten, werden durch eine einzige Relation dargestellt.
   Treten in einer Zusammenhangskomponente gleiche Rollen eines Entitätstyps mehrfach auf, so werden sie in einer entsprechenden Relational als genau ein Attribut übernommen.
10. Manuelles Eingreifen:
    bei `0:1`-Markierung ist u.U. eine Entscheidung, orientiert an der modellierten Semantik, zu treffen:
    Folgende Konstellationen können das Rückgängigmachen von Markierungen innerhalb einer Zusammenhangskomponente notwendig werden lassen:
    mehrere Schlüsselkandidaten und:
    - alle Schlüsselkandidaten sind optional
    - nicht alle verpflichtend und die Notwendigkeit vorhanden, einen bestimmten als Primärschlüssel festzulegen.

Schritt 5

1. Leiten sich aus einem Entitätstyp mehrere sich entsprechende Attribute ab, so sind die folgenden Abhängigkeiten (Fremdschlüsselbeziehungen) zu berücksichtigen:
   - Ist eine Attributkombination Schlüsselkandidat, so sind zu den entsprechenden Attributkombinationen, die als Primärschlüssel ausgewählt wurden, Fremdschlüsselbeziehungen vorzusehen.
     Hinweis: Fremdschlüsselbeziehungen bedeuten zusätzliche Zugriffe und sollten daher in der Datenbank entsprechend durch Indexstrukturen unterstützt werden.
   - Beim Auftreten identischer Schlüsselkandidaten sind ebenfalls Fremdschlüsselbeziehungen vorzusehen.
2. Metainformationen, die nicht die DB-Strukturebene betreffen, sondern Ausprägungen einschränken, werden den entsprechenden DB-Strukturelementen zugeordnet (z.B. Domäneneinschränkungen bei Attributen).

Schritt 6

Ist ein Entitätstyp Spezialisierung (d.h. Untertyp) eines anderen, so wird in die Relation die aus der Zusammenhangskomponente gebildet wurde, welche den Untertypen beinhaltet die Primärschlüsselattribute derjenigen Relation übernommen, die den Obertypen beinhaltet.

Anmerkung: Schlüsselkandidaten dieser Relation werden identisch zu den anderen Relationen ermittelt.

Schritt 7

1. Systemunabhängig
   - Alle Attribute bekommen als Datentyp den Entitätstyp, durch den sie repräsentiert werden.
   - Bei `1:1`-Markierung wird für alle Attribute der Relation ein `NOT NULL` vergeben.
   - Ein Primärschlüssel muß stets mit `NOT NULL` vereinbart werden.
   - Bei markierten `0:1`-Beziehungen: Alle aus über `0:1`-Beziehungen angebundenen Entitätstypen entstehenden

Attribute werden auf NULL gesetzt.
- Schlüsselkandidaten und Zugriffspfade werden als Indexe angelegt.
- Soweit für Metainformation formale Umsetzungsmöglichkeiten existieren, werden die entsprechenden konsistenzgarantierenden Einschränkungen formuliert.

2. Systemabhängig
   Die Syntax für die physische Realisierung der Relationen (Tabellen) und der Indexstrukturen sowie der Datentypen und Einschränkungen (soweit unterstützt) müssen dem jeweiligen DBMS angepaßt werden.
   Evtl. durch das DBMS automatisch angelegte Indexstrukturen müssen nicht mehr explizit formuliert werden.

2.3 Algebraischer Entwurf mit der Normalformentheorie

Neben dem graphischen Entwurf logischer DB-Strukturen genießt der algebraische Entwurf auf Basis der sog. *Normalformentheorie* in Theorie und Praxis große Bedeutung.
Historisch gesehen stellt die Betrachtung von relationalen Strukturen mit Hilfe mathematischer Methoden die älteste Disziplin dar und findet sich heute in allen bedeutenden Lehrbüchern.
Dieser Abschnitt führt in die sechs verschiedenen Normalformen hinsichtlich ihrer Definition sowie ihrer Implikationen auf die Struktur des logischen Modells ein und zieht Parallelen zur Vorgehensweise des eingeführten Algorithmus zur Umsetzung konzeptueller Strukturen des E^3R-Modells.

Ebenso wie der Algorithmus zur Transformation eines E^3R-Schemas führt auch die Normalisierung zu einem anomalienfreien relationalen Datenbankentwurf. Voraussetzung der Anomaliefreiheit ist die konsequente Ermittlung und Eliminierung von Redundanz, d.h. keine Information darf in der Datenbank mehrfach vorhanden sein.

Insgesamt verfolgt der Normalisierungsprozeß folgende Ziele:

- Redundanzvermeidung als Basis der Anomaliefreiheit
- Vermeidung unnötiger Abhängigkeiten, die Performanceeinbußen bei Einfüge-, Lösch- und Änderungsoperationen nach sich ziehen
- Senkung der Anzahl beteiligter Tabellen bei der Modifikation der Datenbank
- Erhöhung des Dokumentationsgrades des entstehenden Datenmodells (Ziel: Verständlichkeit)

Die Anomalienfreiheit ist die zentrale Basisforderung und Zielsetzung des Normalisierungsprozesses. Im Detail werden drei Ausprägungen unterschiedlicher Anomalien unterschieden:

- **Einfügeanomalie: Durch das Hinzufügen eines korrekten neuen Tupels werden konsistent vorliegende Daten in einen inkonsistenten Zustand überführt.**
- **Löschanomalie: Durch die Entfernung eines Tupels entsteht ein inkonsistenter Datenbestand.**
- **Aktualisierungsanomalie: Durch die Änderung eines vorhandenen Tupels entsteht ein inkonsistenter Datenbestand.**

Alle Arten von Anomalien gehen auf das Vorhandensein von Redundanz, mithin einem Verstoß gegen die Grundregel jede im konzeptuellen Schema modellierte Information an nur genau einer Stelle abzuspeichern, zurück.

Ausgangssituation des Normalisierungsvorganges ist die *Urrelation* die alle Attribute in genau einer Relation zusammenfaßt.

Erste Normalform

Definition 32: *Erste Normalform (1NF)*
Eine Relation ist dann in erster Normalform, wenn ihre Domänen (=Wertausprägungen der Attribute) nur einfache (atomare) Werte besitzen.

Atomarer Wert bedeutet hierbei, daß kein Attributinhalt strukturiert sein darf, d.h. durch mögliche Zerlegungsoperationen in kleine eigenständige Informationseinheiten zerlegt werden kann.

Beispiel einer Relation die **nicht in 1NF** ist:

FNAME	LNAME	ADDRESS	BDATE
John	Smith	731 Fondren, Houston, TX	1965-01-09
Franklin	Wong	638 Voss, Houston, TX	1955-12-08
Joyce			1972-07-31
Polly Esther	Wallace	291 Berry, Bellaire, TX	1941-06-20, 1952-09-04

Beispiel 19: Relation, die nicht in 1NF ist

Ziel der Überführung in 1NF ist es, Relationen zu erhalten, die in gängigen RDBMS abspeicherbar sind. Diese bieten zwar heute technische Mechanismen (wie Array- und Referenztypen) an, die Strukturen ähnlich den dargestellten verwaltbar werden lassen. Voraussetzung ihrer konzeptionellen Beherrschung ist jedoch die vorherige Normalisierung.

Im Beispiel befinden sich die Zeilen von `Franklin` und `Joyce Wong` nicht in 1NF, da sie nicht für jedes Attribut einen Wert besitzen, sondern sich eine Wertausprägung (`Wong` und `638 Voss, Houston, TX`) teilen.
Ebenso befindet sich der Eintrag von `Polly Esther Wallace` nicht in 1NF, da hier für das Geburtsdatum unerlaubterweise zwei Einträge auftreten.
Die beiden Vornamen sind im Rahmen der Semantik des Attributes `FNAME` zugelassen und daher im Normalisierungsprozeß nicht zu beanstanden.

Die Relation aus Beispiel 19 in erster Normalform:

FNAME	LNAME	ADDRESS	BDATE
John	Smith	731 Fondren, Houston, TX	1965-01-09
Franklin	Wong	638 Voss, Houston, TX	1955-12-08
Joyce	Wong	638 Voss, Houston, TX	1972-07-31
Polly Esther	Wallace	291 Berry, Bellaire, TX	1941-06-20
Polly Esther	Wallace	291 Berry, Bellaire, TX	1952-09-04

Beispiel 20: Relation, die in 1NF ist

Zur Umformung der Relation in eine Relation in erster Normalform wurden die „gemeinsamen“ Attribute aufgelöst, so das nunmehr jedes Attribut genau einem Tabelleintrag (Tupel) zugeordnet ist.
Zusätzlich wurde für jedes Attribut die atomare Belegung sichergestellt.

Die Einführung der ersten Normalform verhindert damit die Bildung *geschachtelter Relationen*, die entstünden, jedes Attribut eine Menge anderer Attribute, mithin wiederum eine vollständige Relation, enthalten könnte.
Anmerkung: Diese Forderung wird durch die in SQL:1999 definierten `ARRAY`-Typen aufgeweicht und für postrelationale und objektorientierte Datenbanken vollständig aufgegeben, weshalb diese Strukturen auch als *Non-First-Normal-Form* (kurz: NFNF, NF2 oder NF^2) bezeichnet werden.

Test auf Einhaltung der ersten Normalform:
Die Relation sollte keine nicht-atomaren Attribute oder verschachtelte Relationen enthalten.

Der algorithmische Ableitungsprozeß aus dem konzeptuellen Schema stellt durch die Organisation der Repräsentationstypen sicher, daß Attribute ausschließlich durch atomare Werte repräsentiert werden.

Zweite Normalform

Grundlegende Voraussetzung zum Verständnis der zweiten Normalform ist das Konzept der *vollen funktionalen Abhängigkeit*.

Definition 33: *Volle funktionale Abhängigkeit*
Ein Attribut *y* einer Relation ist vollfunktional abhängig von einem Attribut *x* wenn gilt, daß jede Ausprägung von *x* genau eine Ausprägung von *y* bestimmt und *y* nicht abhängig von Teilattributen von *x* ist.
Im Zeichen: *x->y*.
Die vollfunktionale Abhängigkeit wird häufig als *FD* (*functional dependency*) abgekürzt.

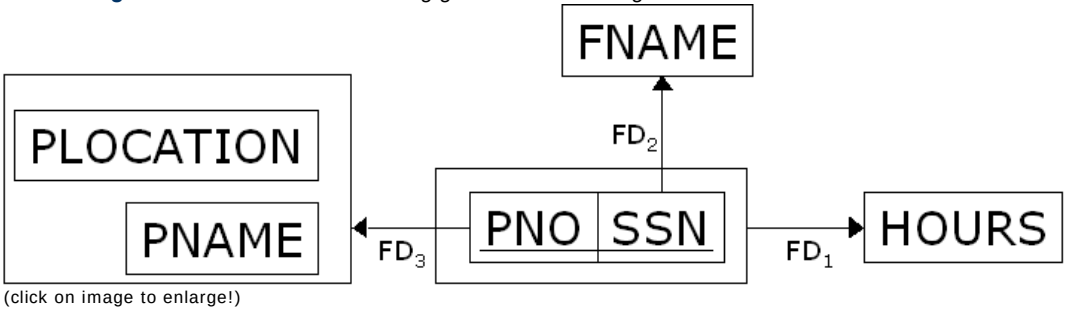
Bemerkung: Man beachte, daß der Begriff *Attribut* in Definition 33 eine nichtleere Menge von Attributen bezeichnet.

Beispiel aus der Demodatenbank:

SSN	PNO	HOURS	FNAME	DNAME	PNAME	PLOCATION
123456789	1	32.5	John	Research	ProductX	Bellaire
123456789	2	7.5	John	Research	ProductY	Sugarland
333445555	2	10.0	Franklin	Research	ProductY	Sugarland
333445555	3	10.0	Franklin	Research	ProductZ	Houston
333445555	10	10.0	Franklin	Research	Computerization	Stafford
333445555	20	10.0	Franklin	Research	Reorganization	Houston
453453453	1	20.0	Joyce	Research	ProductX	Bellaire
453453453	2	20.0	Joyce	Research	ProductY	Sugarland
666884444	3	40.0	Ramesh	Research	ProductZ	Houston
888665555	20	NULL	James	Headquarters	Reorganization	Houston
987654321	20	15.0	Jennifer	Administration	Reorganization	Houston
987654321	30	20.0	Jennifer	Administration	Newbenefits	Stafford
987987987	10	35.0	Ahmad	Administration	Computerization	Stafford
987987987	30	5.0	Ahmad	Administration	Newbenefits	Stafford
999887777	10	10.0	Alicia	Administration	Computerization	Stafford
999887777	30	30.0	Alicia	Administration	Newbenefits	Stafford

In der Relation existieren folgende voll funktionale Abhängigkeiten:

Abbildung 17: *Voll funktionale Abhängigkeiten in der dargestellten Relation*



(click on image to enlarge!)

Es ist offensichtlich, daß zwar HOURS vom vollständigen Primärschlüssel (gebildet aus PNO gemeinsam mit SSN) abhängen (FD_1), aber der Name (FNAME) und die Kombination aus PLOCATION und PNAME nur von SSN bzw. PNO und damit Teilen des Primärschlüssels abhängen (FD_2 bzw. FD_3).

Inhaltlich manifestieren sich diese Probleme im Zwang verschiedene Daten (etwa SSN und FNAME) wiederholt (redundant) abspeichern zu müssen. Ändert sich eine dieser Angaben, so muß potentiell eine große Anzahl Tupel in der Datenbank aktualisiert werden. Werden hierbei nicht

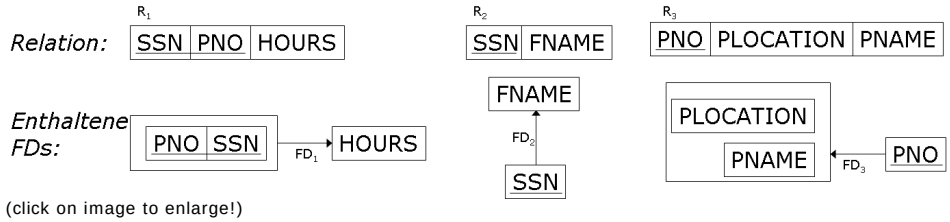
alle Datensätze aktualisiert so entsteht ein inkonsistenter Datenbestand.
Zusätzlich ist es nicht möglich bestimmte Informationszusammenhänge abzubilden. Hierunter fällt beispielsweise der Wunsch Projekte (etwa: PLOCATION und PNAME) zur verwalten, denen noch keinen Mitarbeiter zugeordnet ist.

Um eine Relation in 2NF zu überführen muß sie so zerlegt werden, daß alle Attribute der neu entstehenden Relation vom selben Schlüsselkandidaten abhängen.

Definition 34: *Zweite Normalform (2NF)*
Eine Relation ist in 2NF genau dann, wenn sie in 1NF ist und jedes Nichtschlüsselattribut voll funktional abhängig von einem Schlüsselkandidaten ist.

Entstehende Relationen:

Abbildung 18: *Relationen in 2NF*

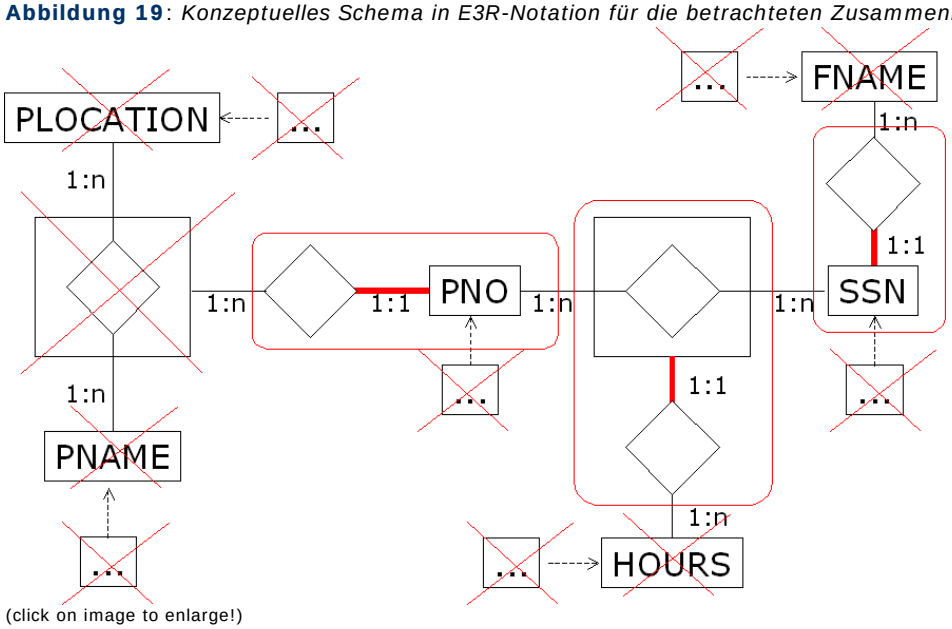


(click on image to enlarge!)

Test auf Einhaltung der zweiten Normalform:
In Relationen deren Primärschlüssel mehrere Attribute enthalten, sollte kein Nichtschlüsselattribut voll funktional von einem Teil des Primärschlüssels abhängen.

Der Ableitungsprozeß aus dem konzeptuellen Schema in E^{3}R-Notation gewährleistet automatisch die Erzeugung von Relationen in 2NF:

Abbildung 19: *Konzeptuelles Schema in E3R-Notation für die betrachteten Zusammenhänge*



(click on image to enlarge!)

Dritte Normalform (3NF)

Die dritte Normalform erweitert die für die Zweite getroffenen Aussagen dahingehend, daß zusätzlich zur voll funktionalen Abhängigkeit die *transitive Abhängigkeit* eingeführt und betrachtet wird.

Definition 35: *Transitive Abhängigkeit*
In einer Relation *R* ist ein Attribut *z* transitiv von einem Attribut *x* abhängig dann und nur dann, wenn *z* voll funktional von *y* und *y* voll funktional von *x* abhängig ist.
Im Zeichen: x->->z

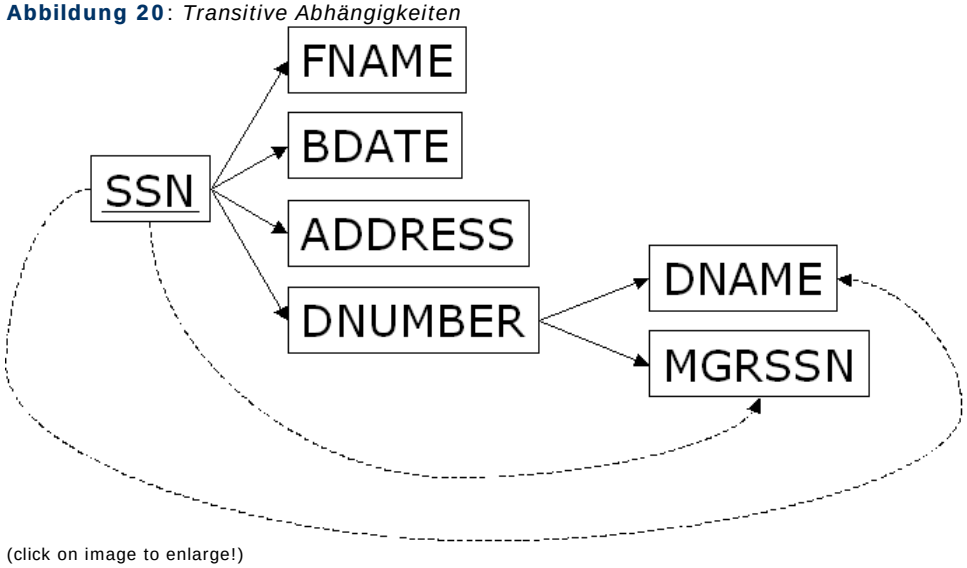
Bemerkung: Man beachte, daß der Begriff *Attribut* in Definition 35 eine nichtleere Menge von Attributen bezeichnet.

Im Beispiel der Demodatenbank:

```
+----------+-----------+------------+--------------------------+---------
+---------------+-----------+
| FNAME    | SSN       | BDATE      | ADDRESS                  | DNUMBER |
DNAME          | MGRSSN    |
+----------+-----------+------------+--------------------------+---------
+---------------+-----------+
| James    | 888665555 | 1937-11-10 | 450 Stone, Houston, TX   |       1 |
Headquarters   | 888665555 |
| Jennifer | 987654321 | 1941-06-20 | 291 Berry, Bellaire, TX  |       4 |
Administation | 987654321 |
| Ahmad    | 987987987 | 1969-03-29 | 980 Dallas, Houston, TX  |       4 |
Administation | 987654321 |
| Alicia   | 999887777 | 1968-07-19 | 3321 Castle, Spring, TX  |       4 |
Administation | 987654321 |
| John     | 123456789 | 1965-01-09 | 731 Fondren, Houston, TX |       5 |
Research       | 333445555 |
| Franklin | 333445555 | 1955-12-08 | 638 Voss, Houston, TX    |       5 |
Research       | 333445555 |
| Joyce    | 453453453 | 1972-07-31 | 5631 Rice, Houston, TX   |       5 |
Research       | 333445555 |
| Ramesh   | 666884444 | 1962-09-15 | 975 Fire Oak, Humble, TX |       5 |
Research       | 333445555 |
+----------+-----------+------------+--------------------------+---------
+---------------+-----------+
```

In der Relation existieren folgende direkten und transitive Abhängigkeiten (die direkten funktionalen Abhängigkeiten sind durch gerichtete Kanten mit durchgezogener Linienführung, die Transitiven durch unterbroche Linienführung dargestellt):

Abbildung 20: *Transitive Abhängigkeiten*



(click on image to enlarge!)

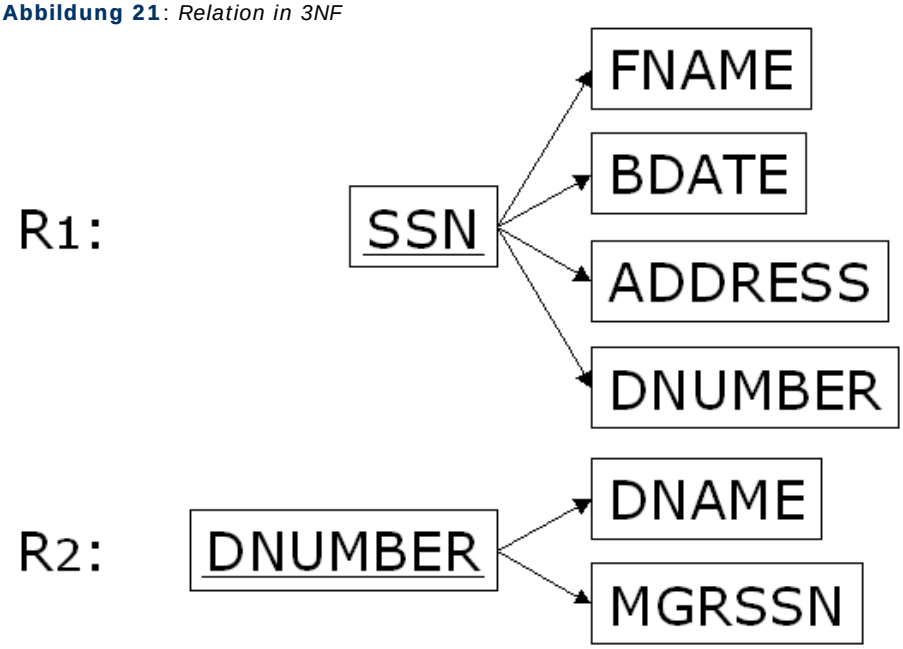
In dieser Organisationsform tritt eine Einfügeanomalie auf, wenn ein Mitarbeiter neu eingefügt wird und dabei „falsche“ (d.h. inkonsistente) Werte für die Abteilung angelegt werden. Zusätzlich fällt das Einfügen neuer Abteilungen, zu denen (noch) kein Mitarbeiter abgespeichert wird, schwer, da SSN zwingend anzugeben ist (NULL-Wert ist wegen der Definition als Primärschlüssel nicht zugelassen!).
Daneben existiert eine Löschanomalie dahingehend, daß wenn der letzte Mitarbeiter einer Abteilung aus der Datenbank entfernt wird auch alle Informationen über diese Abteilung verloren gehen.
Werden Abteilungsdaten verändert, so kann es zu einer Modifikationsanomalie kommen, da nicht in jedem Falle eindeutig klärbar ist ob nur die Abteilungsdaten dieses Mitarbeiters verändert werden sollen (beispielsweise im Falle eines Abteilungswechsels) oder die Daten aller abgespeicherten Abteilungen (beispielsweise im Falle der Umbenennung einer Abteilung).

Die dritte Normalform setzt sich zum Ziel die Ursachen dieser Anomalien zu beseitigen:

Definition 36: *Dritte Normalform (3NF)*
Eine Relation ist dann und nur dann in 3NF, wenn sie in 2NF ist und jedes Nicht-Schlüsselattribut nicht-transitiv abhängig ist von einem Schlüsselkandidaten; sie ist auch in 3NF, wenn sich eine transitive Abhängigkeit ausschließlich über Bestandteile des Schlüsselkandidaten herleiten läßt.

Gemäß dieser Definition ist die Beispielrelation zu zerlegen in:
R_1(SSN, FNAME, BDATE, ADDRESS, DNUMBER) und
R_2(DNUMBER, DNAME, MGRSSN).

Abbildung 21: *Relation in 3NF*

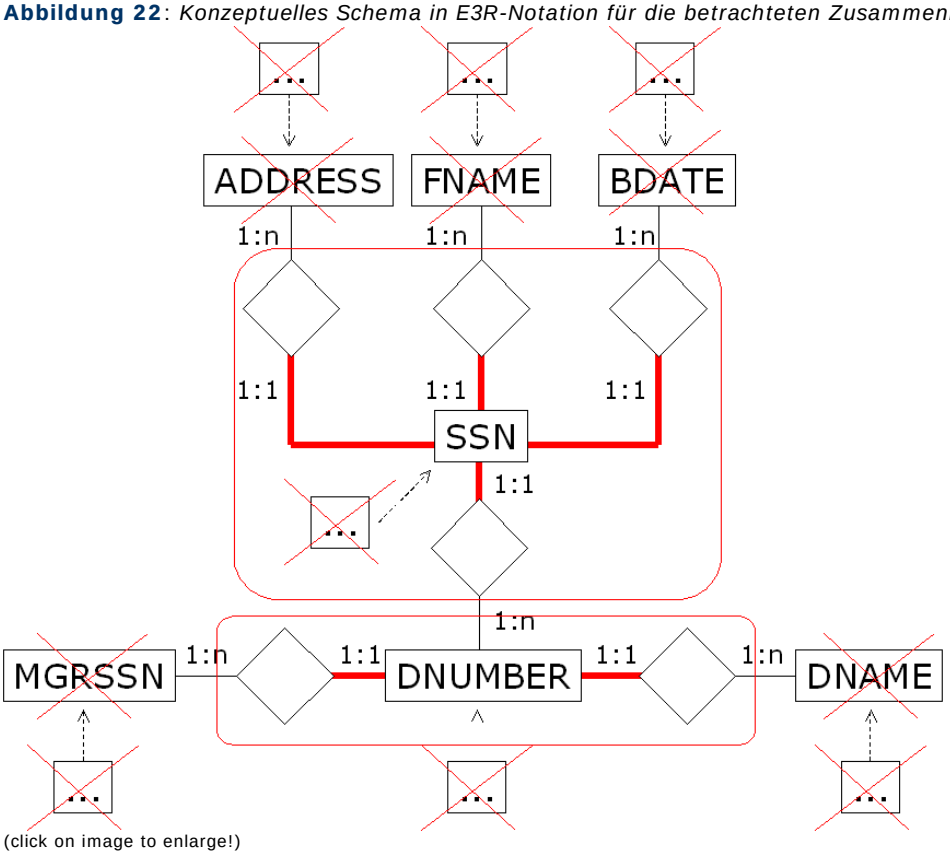


(click on image to enlarge!)

Test auf Einhaltung der dritten Normalform:
Eine Relation sollte kein Nicht-Schlüsselattribut enthalten, das voll funktional von einem anderen Nicht-Schlüsselattribut (oder von einer Menge von Nichtschlüsselattributen) abhängig ist. Das heißt, es sollte keine transitive Abhängigkeit eines Nichtschlüsselattributs vom Primärschlüssel bestehen.

Der Ableitungsprozeß aus dem konzeptuellen Schema in E^{3}R-Notation gewährleistet automatisch die Erzeugung von Relationen in 3NF:

Abbildung 22: *Konzeptuelles Schema in E3R-Notation für die betrachteten Zusammenhänge*



(click on image to enlarge!)

Das Rissanen-Theorem klärt die sich prinzipiell ergebene Fragestellung warum die in Abbildung 21 dargestellte Zerlegung gewählt wurde, da sich prinzipiell unter Ausnutzung der transitiven Abhängigkeiten auch eine (Alternativ-)Zerlegung in R_1(SSN, FNAME, BDATE, ADDRESS, DNUMBER) und R_2(SSN, DNAME, MGRSSN) angeboten hätte.

Nach dem Theorem von Heath und Rissanen ist jedoch die in Abbildung 21 gewählte Zerlegung die einzig korrekt mögliche, da sie die tatsächlich vorhandenen funktionalen Abhängigkeiten erhält, was bei obiger Alternative nicht der Fall wäre.

Angewendet auf unser Beispiel bedeutet dies, daß für obige (nach Heath/Rissenan fehlerhafte) Zerlegung beispielsweise der Anwendungsfall nach Anlage einer Abteilung (DNUMER gemeinsam mit ihrem Namen (DNAME) und der SSN ihres Abteilungsleiters (MGRSSN) nicht möglich wäre, da DNAME und MGRSSN nur verwaltet werden können, wenn gleichzeitig die als Primärschlüssel zwingend anzugebende SSN eines Mitarbeiters dieser Abteilung existiert. Mithin wäre die Speicherung einer Abteilung die nur über ihren Leider aber (noch) nicht über Mitarbeiter verfügt nicht möglich.
Die gewählte Zerlegung vermeidet jedoch diese Einschränkung und liefert, ebenso wie der Abteilungsalgorithmus aus dem konzeptuellen Schema, das korrekte Resultat.

Boyce/Codd-Normalform (BCNF)

Ursprünglich wurde die *Boyce/Codd Normalform* (BCNF) als Vereinfachung der dritten Normalform vorgeschlagen. Jedoch faßt sie diese schärfer und führt so zu einem neuen Typ von Normalform, der nach ihren Schöpfern Boyce und Codd benannt wurde.
Inhaltlich räumt sie mögliche Anomalien aus, die in Relationen, welche sich bereits in 3NF befinden, noch auftreten können.

Definition 37: *Boyce/Codd-Normalform*
In einer Relation ist dann und nur dann in BCNF, wenn sie in 3NF ist und gleichzeitig jede Determinante Schlüsselkandidat ist.

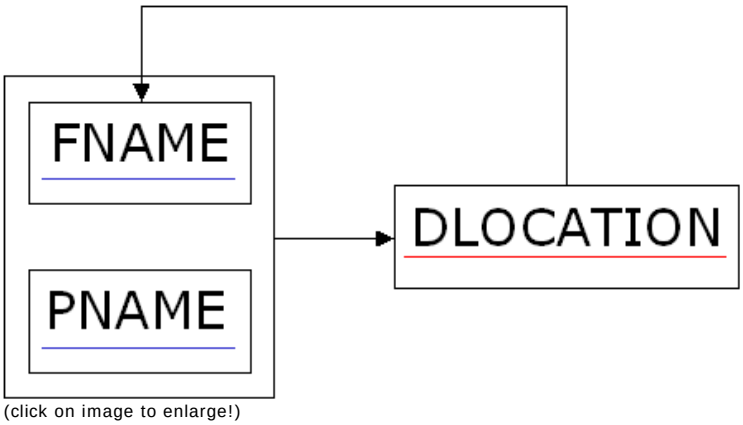
Hierbei definiert die BCNF den Begriff der *Determinante* als den Ausgangspunkt einer funktionalen Abhängigkeit.

Im Beispiel der Demodatenbank:

FNAME	PNAME	DLOCATION
Franklin	ProductY	Sugarland
Franklin	ProductZ	Houston
Jennifer	Newbenefits	Stafford
...	...	...

In der Relation existieren folgende funktionale Abhängigkeiten:

Abbildung 23: *Funktionale Abhängigkeiten*



(click on image to enlarge!)

Die Schlüsselkandidaten sind durch farbliche Unterstreichung hervorgehoben. Hierbei bestimmt FNAME gemeinsam mit PNAME eindeutig einen Wert für DLOCATION (blau) und DLOCATION (rot) bestimmt eindeutig einen Wert für FNAME.

Da sich die transitive Abhängigkeit PNAME->DLOCATION->FNAME ausschließlich über Schlüsselkandidaten herleitet ist die Relation in 3NF.
Dennoch ist sie nicht anomalienfrei, da sie beispielsweise die Ablage von Abteilungsstandorten (DLOCATION) und den Namen der Abteilungsleiter (FNAME) verbietet, wenn kein Projektname (PNAME) zusätzlich abgespeichert wird.

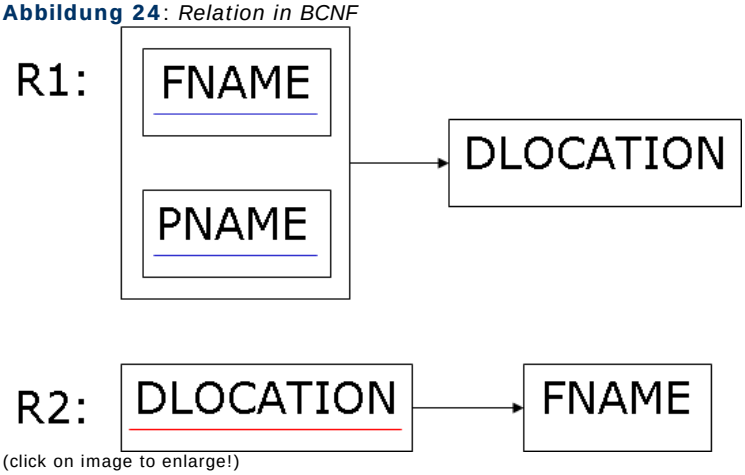
Ebenso ist die Ablage von Projekten und den zuständigen Abteilungen erst dann möglich, wenn zusätzlich der Name des Abteilungsleiters bekannt ist.

Dieses Manko wird durch die Forderung der BCNF behoben. Sie erzwingt die Zerlegung der abgebildeten Ausgangsrelation in zwei Eigenständige. Hierbei wird jede Determinante der Ausgangsrelation zum Schlüsselkandidaten in den neu gebildeten Relationen.

Gemäß Definition 37 ist die Beispielrelation zu zerlegen in:

R_1(FNAME, PNAME, DLOCATION)
R_2(DLOCATION, FNAME).

Abbildung 24: *Relation in BCNF*

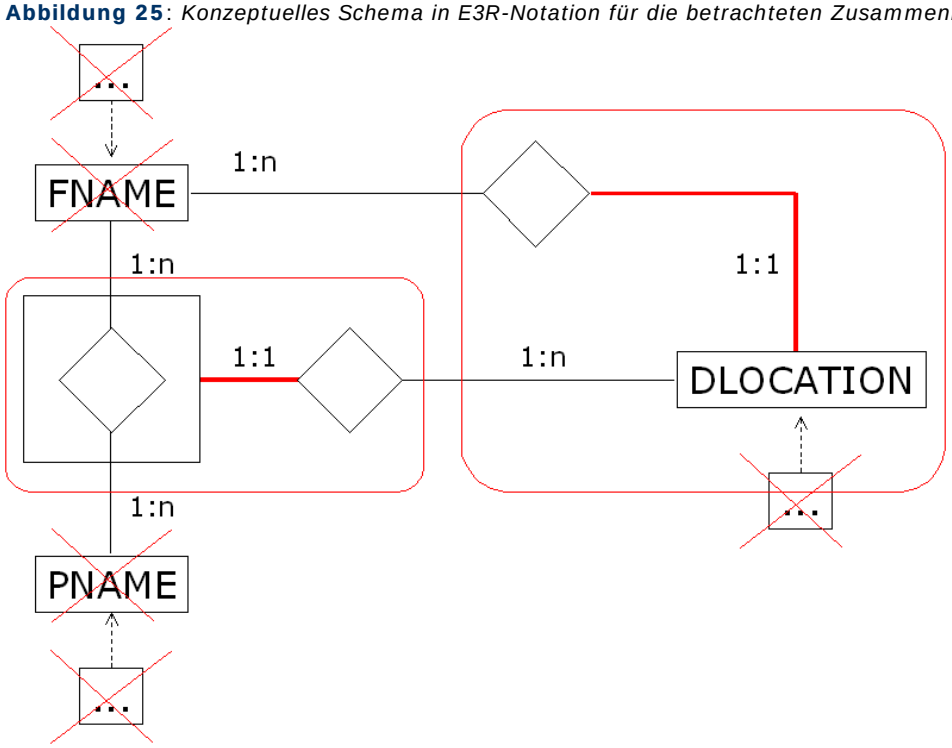


(click on image to enlarge!)

Test auf Einhaltung der Boyce/Codd-Normalform:
Eine Relation sollte lediglich direkt vom Schlüssel abhängende Attribute enthalten.

Der Ableitungsprozeß aus dem konzeptuellen Schema in E^{3}R-Notation gewährleistet automatisch die Erzeugung von Relationen in BCNF:

Abbildung 25: *Konzeptuelles Schema in E3R-Notation für die betrachteten Zusammenhänge*



(click on image to enlarge!)

Vierte Normalform (4NF)

Die bisher betrachteten Normalformen nutzen alle das Vorhandensein funktionaler Abhängigkeiten aus. Jedoch existieren neben diesem Beziehungstyp auch noch andere, im vorhergehenden nicht berücksichtige, Abhängigkeit.

Definition 38: *Mehrwertige Abhängigkeit*
Eine mehrwertige Abhängigkeit (*multivalue dependency*, MVD) ist eine Abhängigkeit, die einem

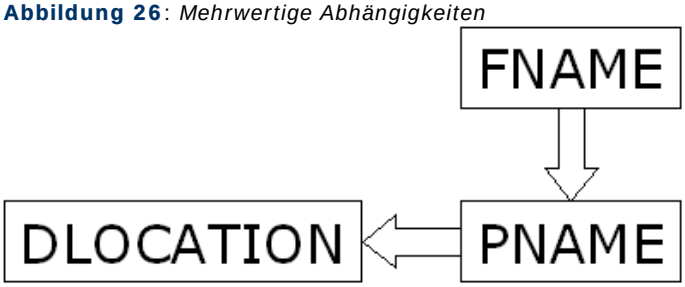
einem Attribut eine Menge verschiedener Werte zuordnet.

Das Vorhandensein mehrwertiger Abhängigkeiten in einer Relation kann zu Aktualisierungsanomalien führen, wie die Betrachtung des bekannten Beispiels aus der Demodatenbank:

FNAME	PNAME	DLOCATION
Franklin	ProductY	Sugarland
Franklin	ProductZ	Houston
Jennifer	Newbenefits	Stafford
...	...	...

In der Relation existieren folgende mehrwertige Abhängigkeiten:

Abbildung 26: *Mehrwertige Abhängigkeiten*



(click on image to enlarge!)

Die Existenz der dargestellten mehrwertigen Abhängigkeiten in der Datenbank führt dazu, daß dieselbe Information mehrfach abgespeichert werden muß. So enthält die Beispieldatenbank mehrfach denselben FNAME sowie wiederholt denselben Produktnamen (PNAME).
Als Konsequenz dieser Wiederholung müssen bei jedem Aktualisierungsvorgang, der die Zuordnung des Produktes zum FNAME betrifft alle FNAME-Einträge geändert werden, jedoch bei der Zuständigkeitsänderung eines Produktverantwortlichen nur der betroffene FNAME.

Ziel der vierten Normalform ist die Elimination von Redundanzen, die aus mehrwertigen Abhängigkeiten herrühren. Allerdings können mehrwertige Abhängigkeiten nicht vollständig entfernt werden, daher ist für die Normalisierung gemäß 4NF die Eliminierung aller *nicht trivialen* mehrwertigen Abhängigkeiten als Zielsetzung definiert.

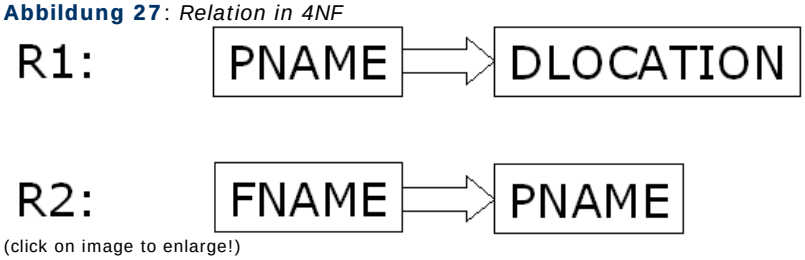
Eine *triviale mehrwertige Abhängigkeit* ist hierbei festgelegt als:

Definition 39: *Triviale mehrwertige Abhängigkeit*
Eine triviale mehrwertige Abhängigkeit ist eine mehrwertige Abhängigkeit zwischen Attributmengen X und Y der Relation R für die gilt: Y ist eine Teilmenge von X oder die Vereinigung von X und Y bildet R.

Definition 40: *Vierte Normalform*
Eine Relation ist dann in vierter Normalform, wenn sie nur noch triviale mehrwertige Abhängigkeiten enthält.

Gemäß dieser Definition ist die Beispielrelation zu zerlegen in:
R_1(DLOCATION, PNAME) und
R_2(FNAME, PNAME).

Abbildung 27: *Relation in 4NF*

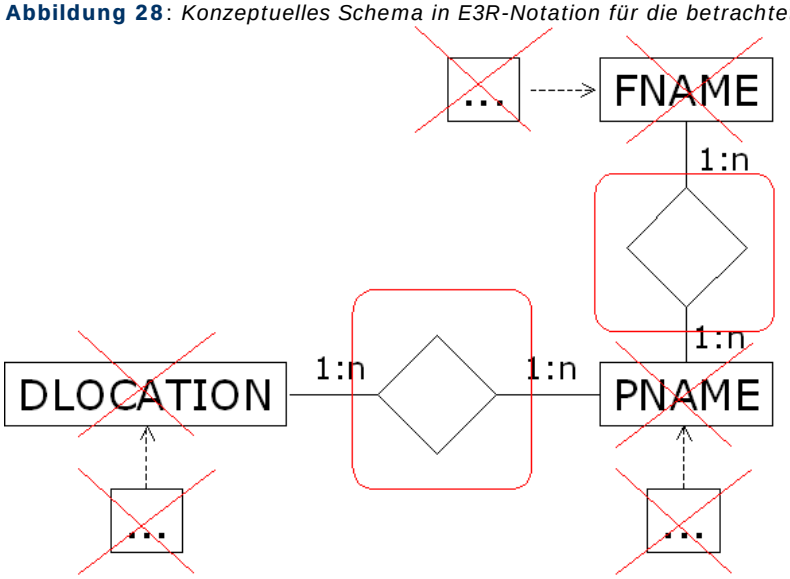


(click on image to enlarge!)

Test auf Einhaltung der vierten Normalform:
Eine Relation sollte keine nicht trivialen mehrwertigen Abhängigkeiten enthalten.

Der Ableitungsprozeß aus dem konzeptuellen Schema in E^{3}R-Notation gewährleistet automatisch die Erzeugung von Relationen in 4NF:

Abbildung 28: *Konzeptuelles Schema in E3R-Notation für die betrachteten Zusammenhänge*



(click on image to enlarge!)

Fünfte Normalform (5NF)

Die fünfte Normalform greift anders als alle vorhergehenden nicht auf die Zusammenhänge der Typebene zurück, sondern benötigt zu ihrer Untersuchung die Betrachtung von tatsächlichen Datenbankinhalten.
Ausgehend von diesen definiert sie die fünfte Normalform als:

Definition 41: *Fünfte Normalform*
Eine Relation ist dann in fünfter Normalform (5NF), bei ihrer Zerlegung durch Projektionen und deren anschließender Kombination durch Verbundoperationen keine Tupel gebildet werden, die nicht Bestandteil der Ausgangsrelation waren.

Aufgrund ihrer Abstützung auf die Projektions- und Verbundoperation (engl. *join*) wird die 5NF auch als *Project Join Normalform* (PJNF) bezeichnet.

Im Beispiel der Demodatenbank:

Relation `Ur`:

FNAME	DNUMBER	DLOCATION
John	5	Bellaire
John	5	Houston
James	1	Houston

Durch Projektion ergeben sich die drei möglichen Relationen:

```
SELECT Ur.FNAME, Ur.DNUMBER INTO R11 FROM Ur
```

R_{11}:

FNAME	DNUMBER
John	5
John	5
James	1

```
SELECT Ur.FNAME, Ur.DLOCATION INTO R12 FROM Ur
```

R_{12}:

FNAME	DLOCATION
John	Bellaire
John	Houston

James	Houston

```
SELECT Ur.DNUMBER, Ur.DLOCATION INTO R13 FROM Ur
```

R_{13}:

DNUMBER	DLOCATION
5	Bellaire
5	Houston
1	Houston

Durch Verbundoperationen entstehen die Relationen:

```
SELECT R11.FNAME, R11.DNUMBER, R12.DLOCATION INTO R21 FROM R11 INNER JOIN R12 ON
R11.FNAME=R12.FNAME
```

R_{21}:

FNAME	DNUMBER	DLOCATION
John	5	Bellaire
John	5	Houston
James	1	Houston

```
SELECT R11.FNAME, R11.DNUMBER, R13.DLOCATION INTO R22 FROM R11 INNER JOIN R13 ON
R11.DNUMBER = R13.DNUMBER
```

R_{22}:

FNAME	DNUMBER	DLOCATION
John	5	Bellaire
John	5	Houston
James	1	Houston

```
SELECT R12.FNAME, R12.DLOCATION, R13.DNUMBER INTO R23 FROM R12 INNER JOIN R13 ON
R12.DLOCATION=R13.DLOCATION
```

R_{23}:

FNAME	DNUMBER	DLOCATION
John	5	Bellaire
John	5	Houston
James	1	Houston
James	5	Houston
John	1	Houston

Trotz der ausschließlich durch Rekombination der zuvor aus der Urrelation gebildeten Projektionen (d.h. ohne Hinzunahme neuer) Daten „entstehten“ in Relation R_{23} (rot hervorgehobene) Tupel (sog. *spurious tupel*), die in dieser Form nicht in der Ausgangsrelation präsent waren.

Erst das Zusammenfügen aller aus Verbundoperationen erzeugten Relationen durch einen erneuten Verbund eliminiert diesen *unechten Tupel*:

```
SELECT R21.FNAME, R22.DNUMBER, R23.DLOCATION INTO RESULT FROM (R21 INNER JOIN R22
ON (R21.FNAME = R22.FNAME) AND (R21.DNUMBER = R22.DNUMBER) AND (R21.DLOCATION =
R22.DLOCATION)) INNER JOIN R23 ON (R22.FNAME = R23.FNAME) AND (R22.DNUMBER = R23.
DNUMBER) AND (R22.DLOCATION = R23.DLOCATION) AND (R21.FNAME = R23.FNAME) AND (R21.
DNUMBER = R23.DNUMBER) AND (R21.DLOCATION = R23.DLOCATION)
```

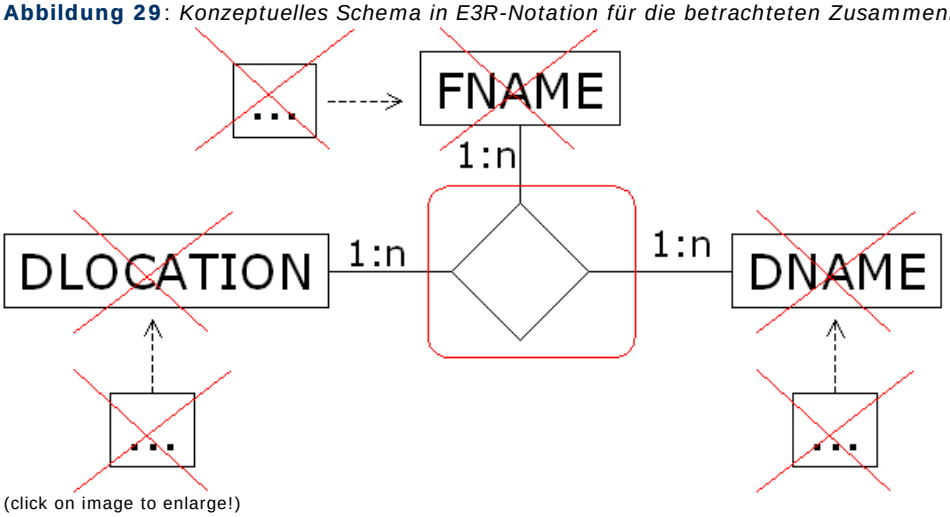
FNAME	DNUMBER	DLOCATION
John	5	Bellaire
John	5	Houston
James	1	Houston

Das Auftreten unechter Tupel ist ein Indiz für eine Relation in der Verbundabhängigkeiten existieren. Eine solche Relation darf nicht weiter zerlegt werden, da durch die Entfernung von Attributen Information verloren ginge.

Test auf Einhaltung der fünften Normalform:
Wenn durch Projektions- und Verbundoperationen keine unechten Tupel entstehen, dann ist die Relation in 5NF.

Der Ableitungsprozeß aus dem konzeptuellen Schema in E^3R-Notation gewährleistet automatisch die Erzeugung von Relationen in 5NF:

Abbildung 29: *Konzeptuelles Schema in E3R-Notation für die betrachteten Zusammenhänge*



(click on image to enlarge!)

Die 5NF ist die höchstmögliche über den Normalisierungsprozeß erreichbare Normalform.
Dies bedeutet jedoch nicht, daß jede Relation bis in 5NF gebracht werden kann. Der Normalisierungsprozeß endet generell mit der höchstmöglichen Normalform, dies muß jedoch nicht immer 5NF sein, sondern orientiert sich an den modellierten Informationszusammenhängen.

Weitere Normalisierungsaspekte

Über die im vorhergehedenden betrachteten formalisierten Normalformen hinaus existieren noch weitere Abhängigkeiten und Normalformen-ähnliche Güteaussagen für Datenbanken.

Inklusionsabhängigkeit:
Die Inklusionsabhängigkeit (engl. *inclusion dependence*, ID) beschreibt die Beziehungen zwischen Super- und Subtypen, insbesondere die *Attributentsprechung*, d.h. die Tatsache, daß der Subtype mindestens alle Attribute des Supertypen aufweist, sowie die *Kompatibilitätsrelation* worunter die Austauschbarkeit von Ausprägungen der beiden Typen verstanden wird für Anwendungsbereiche die lediglich auf die gemeinsam vorhandenen Attribute zugreifen.
aus der Inklusionsabhängigkeit folgen u.a. die drei Inferenzregeln:
$IDIR_1$: (Reflexivität) Die Inklusionsabhängigkeit ist reflexiv.
$IDIR_2$: (Attributentsprechung) Verfügen zwei Typen über dieselben Attribute, dann entsprechen sie sich.
$IDIR_3$: (Transitivität) Ist *B* Untertyp von *A* und *C* Untertyp von *B*, dann ist auch *C* Untertyp von *A*.
Trotz dieser formalisierbaren Aussagen und der breiten Verwendung von Modellierungskonstrukten zur Darstellung von Spezialisierungsbeziehungen wurden auf Basis der Inklusionsabhängigkeit bisher noch keine Normalformen vorgeschlagen.

Template-Abhängigkeiten:
Die Idee der Template-Abhängigkeit fußt auf der Definition einer Reihe von *Hypothesetupeln* und daraus abgeleiteten *Konklusionstupeln*, welche gültige Ausprägungen der Datenbank abstrahiert beispielhaft aufzeigen.
Template-Abhängigkeiten gestatten die einfach Formulierung von Intrarelationsabhängigkeiten die sich auf konkrete Wertausprägungen einzelner Attribute beziehen.
Das Beispiel zeigt die Abhängigkeit, daß kein Angestellter mit mehr Einkommen ausgestattet sein darf als sein Vorgesetzter:

```
EMPLOYEE(FNAME, SSN ..., SALARY, SUPERSSN)
             a     b ...    c     NULL
Hypothese    e     f ...    g     b
-------------------------------------
Konklusion       c < g
```

Domain-Key-Normalform (DKNF):
Grundannahme der Domain-Key-Normalform (DKNF) ist es, daß wenn für jedes Attribut einer Relation eine Domäne (d.h. die Menge der zugelassenen Wertbelegungen) angegeben wird, alle Änderungsanomalien verschwinden.
Gleichzeitig fordert die DKNF die eineindeutige Identifikation jedes Attributs einer Relation durch einen Schlüssel.
In der Praxis kann jedoch die Angabe einer allgemein formulierten eindeutigen Domäne mit unter zu Schwierigkeiten führen, weshalb die Prüfung auf Einhaltung dieser Normalform mit erheblichen technsichen Umsetzungsschwierigkeiten verbunden ist.

3 Arbeiten mit einer Datenbank

3.1 Codd'sche Regeln und Eigenschaften relationaler Systeme

Trotz des in dieser Hinsicht sehr eindeutigen grundlegenden Papiers von E. F. Codd über die Relationenstruktur (*A Relational Model of Data for Large Shared Data Banks*) existierte lange Zeit keine Übereinkunft darüber welche Eigenschaften ein relationales DBMS mindestens aufweisen muß um dieser Systemklasse zugerechnet werden zu können.
Daher definiert Codd 1986 in einem zweiteiligen Artikeln für die Zeitschrift *Computer World* 12 strenge Regeln die ein RDBMS aus seiner Sicht zwingend erfüllen muß um als solches eingestuft werden zu können.
Diese hierin erhobenen Forderungen sind jedoch so streng, daß sie bis heute kein System vollständig erfüllt.
Die Regeln sind nachfolgend mit ihren englischsprachigen Originalbezeichnungen wiedergegeben, da sich für sie bisher keine eindeutige und allgemeinverständliche deutsche Übersetzung etablieren konnte.

Regel 1: **The Information Rule**:
Alle Daten, die in einer Datenbank gespeichert werden sind auf dieselbe Art dargestellt, nämlich durch Werte in Tabellen.
Anmerkung: In dieser Definition wurde bewußt der Begriff der Tabelle gegenüber dem der Relation bevorzugt.

Regel 2: **Guaranteed Access Rule**:
Jeder gespeicherte Wert muß über Tabellenname, Spaltenname und Wert des Primärschlüssels zugreifbar sein, wenn der zugreifende Anwender über hinreichende Zugriffsrechte verfügt.

Regel 3: **Systematic Treatment of Null Values**:
Nullwerte müssen datentypunabhängig zur Darstellung fehlender Werte unterstützt werden.
Systematisch drückt hierbei aus, daß Nullwerte unabhängig von denjenigem Datentyp für den sie auftreten gleich behandelt werden.

Regel 4: **Dynamic On-line Catalog Based on the Relational Model**: Forderung nach einem Online-Datenkatalog (*data dictionary*) in Form von Tabellen.
Dieser Katalog beschreibt die in der Datenbank abgelegten Tabellen hinsichtlich ihrer Struktur und zugelassenen Inhaltsbelegungen.

Regel 5: **Comprehensive Data Sublanguage Rule**:
Für das DBMS muß mindestens eine Sprache existieren durch die sich die verschiedenen Inhaltstypen (Tabelleninhalte, Sichten, Integritätsstrukturen (Schlüsselbeziehungen, Wertebereichseinschränkungen, Aufzählungstypen) sowie Zugriffsrechte) definieren lassen.

Regel 6: **View Updating Rule**:
Sofern theoretisch möglich, müssen Inhalte von Basistabellen auch über deren Sichten änderbar sein.

Regel 7: **High-level Insert, Update, and Delete**:
Innerhalb einer Operation können beliebig viele Tupel bearbeitet werden, d.h. die Operationen werden grundsätzlich mengenorientiert ausgeführt. Hierfür ist eine so abstrahierte Sicht dieser Operationen notwendig, daß keinerlei Information über die syteminterne Darstellung der Tupel notwendig ist.

Regel 8: **Physical Data Independence**:
Änderungen an der internen Ebene dürfen keine Auswirkungen auf die auf den abgespeicherten Daten operierenden Anwendungsprogramme besitzen.
Werden Daten demnach reorganisiert oder beispielsweise durch Indexe zugriffsbeschleunigt, so darf eine solche Änderung die auf die Datenbank zugreifenden Anwendungsprogramme nicht beeinträchtigen.

Regel 9: **Logical Data Independence**:
Änderungen des konzeptuellen Schemas dürfen keine Auswirkung auf die Anwedungsprogramme besitzen, solange diese nicht direkt von der Änderung betroffen sind.

Regel 10: **Integrity Independence**:
In Verfeinerung der fünften Regel wird gefordert, daß alle Integritätsbedingungen ausschließlich durch die Sprache des DBMS definieren lassen können müssen. Definierte Integritätsbedingungen müssen in Tabellen abgespeichert werden und durch das DBMS zur Laufzeit abgeprüft werden.

Im Mindesten werden folgende Forderungen durch verfügbare Systeme unterstützt:

- Kein Attribut welches Teil eines Primärschlüssels ist darf NULL sein.
- Ein Fremdschlüsselattribut muß als Wert des zugehörigen Primärschlüssels existieren.

Regel 11: **Distribution Independence**:
Die Anfragesprache muß so ausgelegt sein, daß Zugriffe auf lokal gehaltene Daten identisch denen auf verteilt gespeicherte Daten formuliert werden können.
Hieraus läßt sich auch die Ausdehnung der Forderungen nach logischer und physischer Datenunabhängigkeit für verteilte Datenbanken ableiten.

Regel 12: **Nonsubversion Rule**:
Definiert ein DBMS neben der High-level Zugriffssprache auch eine Schnittstelle mit niedrigerem Abstraktionsniveau, dann darf durch diese keinesfalls eine Umgehung der definierten Integritätsregeln möglich sein.

Zusätzlich faßt Codd in **Regel 0** nochmals die Anforderungen dahingehend zusammen, daß er postuliert, alle Operationen für Zugriff, Verwaltung und Wartung der Daten ausschließlich mittels relationaler Fähigkeiten abzuwickeln.

Derzeit existiert kein am Markt verfügbares kommerzielles System welches alle zwölf Regeln vollständig umsetzt. Insbesondere sind die Regeln 6, 9, 10, 11 und 12 in der Praxis schwer umzusetzen.

Darüber hinaus greifen die Coddschen Regeln nicht alle Gesichtspunkte des praktischen Datenbankeinsatzes auf. So bleiben Fragestellungen des Betriebs (wie Sicherungs-, Wiederherstellungs- und Sicherheitsaspekte) eines DBMS völlig ausgeklammert.

3.2 Implementierung des logischen Modells mit SQL-DDL

Die SQL-DDL dient allgemein der Definition und Verwaltung von Tabellen- und Indexdefinitionen innerhalb einer relationalen Datenbank entlang ihres gesamten Lebenszyklus, d.h. von ihrer Erstellung über alle Wartungsstadien bis hin zur Entfernung.

Nicht normiert durch den SQL-Standard sind die notwendigen Schritte zur Erzeugung einer Datenbank innerhalb eines Datenbankmanagementsystems. Überdies variieren die hierfür abzusetzenden Kommandos von Hersteller zu Hersteller und müssen der spezifischen Dokumentation entnommen werden.

Hinweis: Zwar läßt SQL inzwischen die beliebige Schreibung der Schlüsselworte (groß, klein oder gemischt) zu, zur bessern Hervorhebung und Kompatibilität mit existierender Literatur werden sie jedoch in den nachfolgenden Syntaxübersichten und Beispielen durchgehend in Großschreibung wiedergegeben.

Innerhalb der Syntaxbeschreibungen gelten folgende Konventionen:

- Schlüsselworte, die direkt wie abgedruckt eingegeben werden müssen sind großgeschrieben.
- Optionale Bestandteile, die weggelassen werden können sind in eckigen Klammern („[]“) dargestellt.

Die Klammern selbst sind nicht Bestandteil der Syntax und müssen nicht eingegeben werden.

- Dargestellte runde Klammern („()“) sind Syntaxbestandteil und müssen unverändert eingegeben werden.
- Senkrechte Striche („|“) trennen Alternativen von denen jeweils eine ausgewählt werden kann, nicht jedoch mehrere.
- Kommentare, die auf die Ausführung keinen Einfluß haben werden durch zwei Minuszeichen („--“) eingeleitet und enden mit dem Zeilenende.
- Alle SQL-Befehle werden generell durch ein Semikolon abgeschlossen. Dieses ist aus Übersichtlichkeitsgründen in den Syntaxübersichten weggelassen.

Erzeugen von Tabellen

Die SQL-Anweisung `CREATE TABLE` dient der Erzeugung neuer Tabellen innerhalb einer bestehenden Datenbank. Sie legt die Struktur und die zugelassenen Typausprägungen, sowie Einschränkungen hinsichtlich der erlaubten Werte fest.

Die vereinfachte Syntax der `CREATE TABLE`-Anweisung lautet:

```
CREATE [TEMPORARY] TABLE tbl_name [(create_definition,...)]
[table_options] [select_statement]

create_definition:
  col_name type [NOT NULL | NULL] [DEFAULT default_value] [AUTO_INCREMENT]
            [PRIMARY KEY] [reference_definition]
  or    PRIMARY KEY (index_col_name,...)
  or    KEY [index_name] (index_col_name,...)
  or    INDEX [index_name] (index_col_name,...)
  or    UNIQUE [INDEX] [index_name] (index_col_name,...)
  or    [CONSTRAINT symbol] FOREIGN KEY [index_name] (index_col_name,...)
            [reference_definition]

type:
        TINYINT[(length)] [UNSIGNED] [ZEROFILL]
  or    SMALLINT[(length)] [UNSIGNED] [ZEROFILL]
  or    MEDIUMINT[(length)] [UNSIGNED] [ZEROFILL]
  or    INT[(length)] [UNSIGNED] [ZEROFILL]
  or    INTEGER[(length)] [UNSIGNED] [ZEROFILL]
  or    BIGINT[(length)] [UNSIGNED] [ZEROFILL]
  or    REAL[(length,decimals)] [UNSIGNED] [ZEROFILL]
  or    DOUBLE[(length,decimals)] [UNSIGNED] [ZEROFILL]
  or    FLOAT[(length,decimals)] [UNSIGNED] [ZEROFILL]
  or    DECIMAL(length,decimals) [UNSIGNED] [ZEROFILL]
  or    NUMERIC(length,decimals) [UNSIGNED] [ZEROFILL]
  or    CHAR(length) [BINARY]
  or    VARCHAR(length) [BINARY]
  or    DATE
  or    TIME
  or    TIMESTAMP
  or    DATETIME
  or    TINYBLOB
  or    BLOB
  or    MEDIUMBLOB
  or    LONGBLOB
  or    TINYTEXT
  or    TEXT
  or    MEDIUMTEXT
  or    LONGTEXT
  or    ENUM(value1,value2,value3,...)
  or    SET(value1,value2,value3,...)

index_col_name:
        col_name [(length)]

reference_definition:
        REFERENCES tbl_name [(index_col_name,...)]
                   [MATCH FULL | MATCH PARTIAL]
                   [ON DELETE reference_option]
                   [ON UPDATE reference_option]
```

```
reference_option:
        RESTRICT | CASCADE | SET NULL | NO ACTION | SET DEFAULT
```

```
mysql> CREATE TABLE Person(
        Name VARCHAR(25)
);
```

Beispiel 21: Erzeugung einer Tabelle

Die Anweisung aus Beispiel 21 stellt den einfachsten Fall ein Tabellenerzeugungsanweisung dar. Es wird die Tabelle `Person`, die mit `Name` nur über eine Spalte verfügt erzeugt.
Eine „kleinere“ Fassung ist nicht möglich, da spaltenlose Tabellen nicht erstellt werden können.

Informationen über eine angelegte Tabelle können durch den Befehl `DESCRIBE` gefolgt vom Namen der abzufragenden Tabelle erlangt werden:

```
mysql> DESCRIBE Person;
+-------+-------------+-------------------+------+-----+---------+-------+
| Field | Type        | Collation         | Null | Key | Default | Extra |
+-------+-------------+-------------------+------+-----+---------+-------+
| Name  | varchar(25) | latin1_swedish_ci | YES  |     | NULL    |       |
+-------+-------------+-------------------+------+-----+---------+-------+
```

Beispiel 22: Ermittlung von Tabelleninformation

Im Beispiel werden die zuvor festgelegten Daten wie Spaltenname (`Name`) und Datentyp (`VARCHAR(25)`) ermittelt, sowie die Belegung einiger Vorgabewerte (u.a. `NULL` und `Default`).

Durch Angabe des Schlüsselwortes `TEMPORARY` können Tabellen erstellt werden, die während der Arbeit mit der Datenbank den herkömmlichen gleichgestellt behandelt werden, jedoch dem Ende der Datenbankverbindung automatisch inclusive aller darin abgelegten Daten aus der Datenbank entfernt werden.

```
mysql> CREATE TEMPORARY TABLE Person2(
        Name VARCHAR(25)
);

--Verbindungsende
--Aufbau einer neuen Verbindung

mysql> DESCRIBE Person2;
ERROR 1146: Table 'SQLTest.Person2' doesn't exist
```

Beispiel 23: Erzeugung einer temporären Tabelle

Wird die Datenbankverbindung getrennt und neu aufgebaut, so ist die zuvor temporär erstellte Tabelle Peson2 nicht mehr vorhanden und zugreifbar. Das `DESCRIBE`-Kommando liefert daher einen Fehler.

Wie bereits in Beispiel 21 gezeigt muß jede Spalte einer Tabelle einen Datentyp besitzen. Dieser definiert die zugelassenen Wertbelegungen und wird durch das DBMS bei jeder schreibenden Operation (d.h. Einfügen, Ändern und Leeren) geprüft.
Wird versucht ein ungültiger Wert zu setzen, so erfolgt eine Fehlermeldung und die Ablehnung des Eintragungs- oder Änderungswunsches. Im Falle eines zeichenkettenwertigen Datentyps (z. B. `VARCHAR`) erfolgt keine Fehlermeldung, sondern die Werte werden nur abgeschnitten eingefügt.

```
mysql> CREATE TABLE Person(
        Name VARCHAR(25)
);

mysql> INSERT INTO Person values("Max Mustermann");
--ok

mysql> INSERT INTO Person values("Franz Obermüller-Hinterhuber-Niedermayer");
--keine Fehlermeldung
--Wert wird jedoch nur abgeschnitten eingefügt:

mysql> SELECT * FROM Person;
+---------------------------+
| Name                      |
+---------------------------+
| Max Mustermann            |
| Franz Obermüller-Hinterhu |
+---------------------------+
```

Beispiel 24: Auswirkung von Datentypen I

Beispiel 24 zeigt die Auswirkung eines gesetzten Datentypen VARCHAR. Werden, wie im Beispiel Werte eingefügt, die die zulässige maximale Zeichenkettenlänge überschreiten, so werden die überzähligen Zeichen ohne Fehlermeldung abgeschnitten.

```
mysql> CREATE TABLE Person(GebDat date);

mysql> INSERT INTO Person values ('1970-12-12');
--ok

mysql> insert into Person values ('1970-19-42');
--offensichtlich falsch
--Übernommener Wert: 0000-00-00
```

Beispiel 25: Auswirkung von Datentypen II

Im Beispiel 25 wird ein offensichtlich ungültiges Datum eingefügt. Die Datenbank übernimmt jedoch nicht diesen falschen Wert sondern setzt den Vorgabewert von `0000-00-00`.
Dasselbe geschieht auch bei numerischen Datentypen (etwa: `INTEGER`) wenn versucht wird sie mit einer Zeichenkette zu belegen.

MySQL unterstützt drei Datentypklassen:

- Numerischedatentypen zur Darstellung von Zahlen.
- Zeichenkettendatentypen zur Darstellung von Texten und Binärdaten.
- Datumsdatentypen zur Darstellung von Uhrzeit- und Datumsformaten.

Die derzeit durch MySQL angebotenen Datentypen sind in der nachfolgenden Tabelle zusammengestellt:

Typname	Beispiel	Bemerkung
Numerische Datentypen		
`TINYINT [UNSIGNED]`	(Vorzeichenbehaftete) Ganzzahl der Breite acht Bit.	(2^7, ..., -1) 0, 1, ..., 2^7-1), (2^7, ... 2^8-1) (-128, ..., -1,) 0, 1, ..., 127, (128, ..., 255)
`BOOL`	Boole'scher Wahrheitswert.	Zugelassene Belegungen `0` und `1`. Synonym für `TINYINT(1)`
`SMALLINT [UNSIGNED]`	(Vorzeichenbehaftete) Ganzzahl der Breite 16 Bit.	(2^{15}, ..., -1) 0, 1, ..., 2^{15}-1), (2^{15}, ... 2^{16}-1) (-32.768, ..., -1,) 0, 1, ..., 32.767, (32.768, ..., 65.535)
`MEDIUMINT [UNSIGNED]`	(Vorzeichenbehaftete) Ganzzahl der Breite 24 Bit.	(2^{23}, ..., -1) 0, 1, ..., 2^{23}-1), (2^{23}, ... 2^{24}-1) (-8.388.608, ..., -1) 0, 1, ..., 8.388.607, (8.388.608, ..., 16.777.215)
`INT [UNSIGNED]`	(Vorzeichenbehaftete) Ganzzahl der Breite 32 Bit.	(2^{31}, ..., -1) 0, 1, ..., 2^{31}-1), (2^{31}, ... 2^{32}-1) (-2.147.483.648, ..., -1) 0, 1, ..., 2.147.483.647, (2.147.483.648, ..., 4.294.967.295)

INTEGER [UNSIGNED]		Synonym für INT
BIGINT [UNSIGNED]	(Vorzeichenbehaftete) Ganzzahl der Breite 64 Bit.	(2^{63}, ..., -1) 0, 1, ..., 2^{63}-1), (2^{63}, ... 2^{64}-1) (-9.223.372.036.854.774.808, ..., -1) 0, 1, ..., 9.223.372.036.854.774.807, (9.223.372.036.854.774.808, ..., 18.446.744.073.709.551.615)
FLOAT [UNSIGNED]	(Vorzeichenbehaftete) Fließkommazahl der Breite 32 Bit, gemäß dem Standard IEEE-754	$(-(2-2^{-23})^{127}$, ..., $(2-2^{-23})^{127})$ $-3{,}402...*10^{38}$, ..., $-1{,}175...*10^{-38}$ und $1{,}175...*10^{-38}$ ... $3{,}402...*10^{38}$
DOUBLE [UNSIGNED]	(Vorzeichenbehaftete) Fließkommazahl der Breite 64 Bit, gemäß dem Standard IEEE-754	$(-(2-2^{-52})^{1023}$, ... $(2-2^{-52})^{1023}$ $-1{,}797...*10^{-308}$, ..., $-2{,}225...*10^{308}$ und $1{,}797...^{308}$, ..., $2{,}225...^{-308}$
REAL [UNSIGNED]		Synonym für DOUBLE
DECIMAL [(Genauigkeit, [Nachkommastellen])]	Festkommazahl beliebiger Genauigkeit	DECIMAL(9,2) liefert eine Dezimalzahl mit neun Gesamtziffern, davon zwei Nachkommastellen.
DEC		Synonym für DECIMAL
NUMERIC		Synonym für DECIMAL
Zeichenkettendatentypen		
CHAR [Länge]	Textfeld konstanter Länge bis zu 255 Zeichen, das immer die angegebene Anzahl Speicherstellen benötigt.	Dies ist ein Test0x20;0x20;0x20;
CHARACTER		Synonym zu CHAR
NCHAR		Synonym zu CHAR
NATIONAL CHARACTER		Synonym zu CHAR
CHARACTER VARYING		Synonym zu VARCHAR
NATIONAL VARCHAR		Synonym zu VARCHAR
VARCHAR [Länge]	Textfeld variabler Länge. Überzählige Leerzeichen am Ende einer Zeichenkette werden vor dem Abspeichern entfernt.	Dies ist ein Test
TINYTEXT	Textfeld variabler Länge, bis zu 2^8-1 (=255) Zeichen.	... etwas mehr Text
TEXT	Textfeld variabler Länge, bis zu 2^{16}-1 (=65.535) Zeichen.	Sehr viel ... Text hier ...
MEDIUMTEXT	Textfeld variabler Länge, bis zu 2^{24}-1 (=16.777.215) Zeichen.	... etwas mehr Text hier ...
LONGTEXT	Textfeld variabler Länge, bis zu 2^{32}-1 (=2.294.967.295) Zeichen.	... noch mehr Text hier ...
TINYBLOB	Binäre Form von TINYTEXT.	
MEDIUMBLOB	Binäre Form von MEDIUMTEXT	
	BLOB	Binäre Form von TEXT
Datumstypen		
DATE	Datum in ISO-8601-Schreibweise (JJJJ-MM-TT)	2004-06-08
TIME	Uhrzeit gemäß ISO 8601 (hh:mm:ss)	07:45:00
DATETIME	Datum und Uhrzeit gemäß ISO 8601 (JJJJ-MM-TT hh:mm:ss)	2005-05-27 07:45:00
TIMESTAMP	Sekundengenauer Zeitpunkt zwischen 1970-01-01 und 2037-12-31	Anwenderdarstellung: 2004-06-08 12:52:03
YEAR	Vierstellige Jahreszahl	2004
Komplexe Datentypen		
ENUM	Aufzählungstyp mit bis zu 65.535 Elementen	
SET	Menge mit bis zu 64 Elementen	

Jede Spalte einer Relation muß mit genau einem Datentyp der oben dargestellten Liste versehen werden. Nachfolgend sind einige Definitionen und Besonderheiten zusammengestellt:

```
mysql> CREATE TABLE test(wenigText CHAR(300));
--zulässige Grenze für CHAR überschritten ...

mysql> DESCRIBE test;
+-----------+------+-------------------+------+-----+---------+-------+
| Field     | Type | Collation         | Null | Key | Default | Extra |
+-----------+------+-------------------+------+-----+---------+-------+
| wenigText | text | latin1_swedish_ci | YES  |     | NULL    |       |
+-----------+------+-------------------+------+-----+---------+-------+
```

Beispiel 26: Auswirkung von Datentypen III

Beispiel 26 zeigt die automatische Konversion des Datentypen CHAR in der Datenbank in TEXT sobald bereits bei der Anlage die zulässige Größenbegrenzung für CHAR überschritten wird. Werden zur Laufzeit Texte größer als die im CREATE TABLE-Ausdruck angegebene Maximalkapazität abgespeichert, so werden alle überzähligen Zeichen abgeschnitten.

```
mysql> CREATE TABLE test(ts timestamp, x VARCHAR(10));
Query OK, 0 rows affected (0.35 sec)

mysql> INSERT INTO test values(null, "abc");
Query OK, 1 row affected (0.06 sec)

mysql> INSERT INTO test values(null, "def");
Query OK, 1 row affected (0.05 sec)

mysql> INSERT INTO test values(null, "abc");
Query OK, 1 row affected (0.06 sec)

mysql> SELECT * FROM test;
+---------------------+------+
| ts                  | x    |
+---------------------+------+
| 2003-05-26 23:25:51 | abc  |
| 2003-05-26 23:26:13 | def  |
| 2003-05-26 23:26:28 | abc  |
+---------------------+------+
3 rows in set (0.00 sec)

mysql> UPDATE test SET x="xyz" WHERE x="abc";
Query OK, 2 rows affected (0.05 sec)
Rows matched: 2  Changed: 2  Warnings: 0

mysql> SELECT * FROM test;
+---------------------+------+
| ts                  | x    |
+---------------------+------+
| 2003-05-26 23:27:10 | xyz  |
| 2003-05-26 23:26:13 | def  |
| 2003-05-26 23:27:10 | xyz  |
+---------------------+------+
3 rows in set (0.00 sec)
```

Beispiel 27: Auswirkung von Datentypen IV

Das Beispiel zeigt die Nutzung des Typs TIMESTAMP. Spalten dieses Typs werden automatisch durch das DMBS mit Werten versorgt werden. Daher ist die Belegung mit NULL bei Einfügung der drei Zeichenketten wirkungslos.
Überdies wird der Wert jeder TIMESTAMP-typisierten Spalte (im Beispiel: ts) bei jedem Schreibvorgang aktualisiert. Dies zeigt die nochmalige Ausgabe der Tabelleninhalte nach Aktualisierung der beiden Tupel, die für das Attribut x den Wert abc aufweisen.

```
mysql> create table Ampel(farbe enum('rot','gelb','gruen'));
Query OK, 0 rows affected (0.07 sec)

mysql> INSERT INTO Ampel VALUES('rot');
Query OK, 1 row affected (0.05 sec)

mysql> INSERT into Ampel VALUES('blau');
Query OK, 1 row affected (0.04 sec)

mysql> SELECT * FROM Ampel;
+-------+
| farbe |
+-------+
| rot   |
|       |
+-------+

mysql> INSERT INTO Ampel Values(2);
Query OK, 1 row affected (0.05 sec)

mysql> SELECT * FROM Ampel;
+-------+
| farbe |
+-------+
| rot   |
|       |
| gelb  |
+-------+
3 rows in set (0.00 sec)
```

Beispiel 28: Auswirkung von Datentypen V

Das Beispiel 28 zeigt die Nutzung eines Aufzählungstypen.
Er erlaubt ausschließlich das Einfügen der vordefinierten Werte und legt für alle ungültigen Belegungen (im Beispiel: `blau`) die leeren Zeichenkette ab.
Die Werte können dabei wie in der Aufzählung definiert oder durch ihre Indexposition (beginnend ab 1) gespeichert werden.

```
mysql> CREATE TABLE test(x SET("a","b","c","d"));
Query OK, 0 rows affected (0.07 sec)

mysql> INSERT INTO test VALUES("a");
Query OK, 1 row affected (0.07 sec)

mysql> INSERT INTO test values("a,b");
Query OK, 1 row affected (0.04 sec)

mysql> INSERT INTO test values("a,c");
Query OK, 1 row affected (0.07 sec)

mysql> INSERT INTO test values("b,c");
Query OK, 1 row affected (0.05 sec)

mysql> SELECT * FROM test;
+------+
| x    |
+------+
| a    |
| a,b  |
| a,c  |
| b,c  |
+------+
4 rows in set (0.00 sec)

mysql> select * FROM test WHERE x & 1;
+------+
| x    |
+------+
| a    |
```

```
| a,b  |
| a,c  |
+------+
3 rows in set (0.00 sec)
--liefert alle Tupel, die das zweite Mengenelement (="a") enthalten
```

Beispiel 29: Auswirkung von Datentypen VI

Das Beispiel zeigt die Definition eines mengenwertigen Datentyps, d.h. einer Tabellenspalte, die mehr als einen Wert aufnehmen kann sowie die Abfragemöglichkeiten dafür.
Hinweis: Dieser Datentyp führt bereits in die NF^2-Datenstrukturen über und wird daher nicht im Rahmen des Entwurfsprozesses im konzeptuellen Schema verwendet.

Ergänzend zur Datentypangabe können für jede Spalte weitere einschränkende Angaben zur Spezifikation der erlaubten Werte getroffen werden.

`NOT NULL` legt hierbei fest, daß ein Tupel für eine Spalte zwingend einen von `NULL` verschiedenen Wert besitzen muß.

```
mysql> CREATE TABLE Person(Name VARCHAR(20), PersAuswNr INT NOT NULL);
Query OK, 0 rows affected (0.08 sec)

mysql> INSERT INTO Person VALUES('Max Obermüller', '123456789');
Query OK, 1 row affected (0.11 sec)

mysql> INSERT INTO Person VALUES('Xaver Hinterhuber', NULL);
ERROR 1048: Column 'PersAuswNr' cannot be null

mysql> select * from Person;
+----------------+------------+
| Name           | PersAuswNr |
+----------------+------------+
| Max Obermüller |  123456789 |
+----------------+------------+
1 row in set (0.00 sec)
```

Beispiel 30: Definition einer Spalte als NOT NULL

Das Beispiel zeigt eine Tabelle, bei der das Attribut `PersAuswNr` als `NOT NULL` definiert wurde. Einfüge- oder Aktualisierungsversuche, die zu Nullwerten dieses Attributs führen würden, werden durch das DMBS unterbunden.

Alternativ dazu gestattet die Angabe von `NULL` die Existenz von Nullwerten in der Tabelle. Diese Definition ist optional und wird bei fehlender Angabe automatisch als Vorgabe gesetzt.
Das Beispiel zeigt die Definition der Spalte `Autofahrer` als `NULL`, die neben den beiden vorgegebenen Werten auch keinen Wert enthalten darf.

```
mysql> CREATE TABLE Person(
    -> Name VARCHAR(20),
    -> PersAuswNr INT NOT NULL,
    -> Autofahrer ENUM('J','N') NULL);
Query OK, 0 rows affected (0.07 sec)

mysql> INSERT INTO Person VALUES('Max Obermüller', '123456789', NULL);
Query OK, 1 row affected (0.12 sec)

mysql> INSERT INTO Person VALUES('Xaver Hinterhuber', '234567891', 'J');
Query OK, 1 row affected (0.04 sec)

mysql> select * from Person;
+-------------------+------------+------------+
| Name              | PersAuswNr | Autofahrer |
+-------------------+------------+------------+
| Max Obermüller    |  123456789 | NULL       |
| Xaver Hinterhuber |  234567891 | J          |
+-------------------+------------+------------+
3 rows in set (0.01 sec)
```

Beispiel 31: Definition einer Spalte als NULL

Die Angabe der Klausel `DEFAULT VALUE` gestattet es einen Vorgabewert zu definieren, der gesetzt wird wenn kein Wert für eine Spalte angegeben wird.
Dies ersetzt jedoch nicht automatisch die Möglichkeit des Auftretens von Nullwerten innerhalb einer Spalte. Diese können auch weiterhin auftreten, sofern sie explizit eingefügt werden.
Die unterschiedlichen Wirkungsweisen zeigt Beispiel 32. Dort findet sich die Spalte `Autofahrer` (vorgabegemäß, da keine andere Angabe erfolgte) als nullwertfähig mit Vorgabewert `J`, sowie die Spalte `Hundebesitzer`, die mit Vorgabe `N` und als nicht nullwertfähig deklariert wurde.

```
mysql> CREATE TABLE Person(
    -> Name VARCHAR(20) NOT NULL,
    -> Autofahrer ENUM('J','N') DEFAULT 'J',
    -> Hundebesitzer ENUM('J','N') NOT NULL DEFAULT 'N'
    -> );
Query OK, 0 rows affected (0.07 sec)

mysql> INSERT INTO Person VALUES('Xaver Obermüller', 'J', 'J');
Query OK, 1 row affected (0.10 sec)

mysql> INSERT INTO Person VALUES('Max Hinterhuber', NULL, 'N');
Query OK, 1 row affected (0.05 sec)

mysql> INSERT INTO Person (Name) VALUES('Schorsch Huber');
Query OK, 1 row affected (0.05 sec)

mysql> SELECT * FROM Person;
+------------------+------------+---------------+
| Name             | Autofahrer | Hundebesitzer |
+------------------+------------+---------------+
| Xaver Obermüller | J          | J             |
| Max Hinterhuber  | NULL       | N             |
| Schorsch Huber   | J          | N             |
+------------------+------------+---------------+
3 rows in set (0.00 sec)
```

Beispiel 32: Definition einer Spalte mit Vorgabewerten

Ist die Auszeichnung einer einzelnen Spalte als Primärschlüssel gewünscht, so kann der Definition `(PRIMARY) KEY` nachgestellt werden um dies zu erreichen.
Die Definition eines Primärschlüssels impliziert den Zwang in jedem Tupel einen von `NULL` verschiedenen eineindeutigen Wert dafür abspeichern zu müssen. Primärschlüsselspalten sind damit immer auch `NOT NULL`.
Zusätzlich kann für jede Tabelle höchstens ein Primärschlüsselattribut angegeben werden.
Hinweis: Aus mehr als einer Spalte zusammengesetzte Primärschlüssel können nicht durch diese Syntax gebildet werden.

```
mysql> CREATE TABLE Person(Name VARCHAR(20) PRIMARY KEY, Adresse VARCHAR(50));
Query OK, 0 rows affected (0.06 sec)

mysql> INSERT INTO Person VALUES('Xaver Obermüller', 'Dorfstr. 12');
Query OK, 1 row affected (0.10 sec)

mysql> INSERT INTO Person VALUES('Hans Hintermeier', 'Dorfstr. 13');
Query OK, 1 row affected (0.05 sec)

mysql> INSERT INTO Person (Name) VALUES ('Schorsch Huber');
Query OK, 1 row affected (0.06 sec)

mysql> select * from Person;
+------------------+--------------+
| Name             | Adresse      |
+------------------+--------------+
| Hans Hintermeier | Dorfstr. 13  |
| Schorsch Huber   | NULL         |
| Xaver Obermüller | Dorfstr. 12  |
+------------------+--------------+
```

```
4 rows in set (0.00 sec)

mysql> INSERT INTO Person VALUES(NULL, 'Hauptstr. 11');
ERROR 1048: Column 'Name' cannot be null
```

Beispiel 33: Definition eines Primärschlüssels

Zur Erstellung eines zusammengesetzten Primärschlüssels kann nicht das nachgestellte Schlüsselwort `PRIMARY KEY` verwendet werden, da seine wiederholte Angabe mehrdeutig wäre. Für diesen Fall muß eine gesondert `PRIMARY KEY`-Definition in den `CREATE TABLE`-Ausdruck aufgenommen werden:

```
CREATE TABLE DEPT_LOCATIONS(
   DNUMBER INTEGER(1) NOT NULL,
   DLOCATION VARCHAR(20) NOT NULL,
   PRIMARY KEY (DNUMBER, DLOCATION));
```

Beispiel 34: Definition eines zusammengesetzten Primärschlüssels

Das Beispiel zeigt verschiedene Einfügeoperationen. Bemerkenswert ist die letzte, die das Primärschlüsselattribut leer läßt. Die hierbei erfolgende Belegung mit der leeren Zeichenkette (nicht `NULL`!) ist ein gültiger Eintrag im Sinne des angegebenen Datentypen `VARCHAR` und der Restriktion keine Belegung mit `NULL` vorzunehmen.

Soll ein nicht-sprechender Schlüssel (z.B. eine einfache Zählnummer) zur Identifikation genutzt werden, so kann diese durch das DBMS automatisiert bereitgestellt werden.

```
mysql> CREATE TABLE Person(
    -> LfdNr Int AUTO_INCREMENT PRIMARY KEY,
    -> Name VARCHAR(20) NOT NULL);
Query OK, 0 rows affected (0.07 sec)

mysql> INSERT INTO Person VALUES(1,'Max Obermüller');
Query OK, 1 row affected (0.09 sec)

mysql> INSERT INTO Person VALUES(3,'Schorsch Hinterhuber');
Query OK, 1 row affected (0.05 sec)

mysql> INSERT INTO Person VALUES(3,'Xaver Mayer');
ERROR 1062: Duplicate entry '3' for key 1

mysql> SELECT * FROM Person;
+-------+----------------------+
| LfdNr | Name                 |
+-------+----------------------+
|     1 | Max Obermüller       |
|     3 | Schorsch Hinterhuber |
+-------+----------------------+
2 rows in set (0.00 sec)

mysql> INSERT INTO Person (Name) VALUES('Xaver Mayer');
Query OK, 1 row affected (0.05 sec)

mysql> INSERT INTO Person (Name) VALUES('Hans Huber');
Query OK, 1 row affected (0.04 sec)

mysql> select * from Person;
+-------+----------------------+
| LfdNr | Name                 |
+-------+----------------------+
|     1 | Max Obermüller       |
|     2 | Xaver Mayer          |
|     3 | Schorsch Hinterhuber |
|     4 | Hans Huber           |
+-------+----------------------+
4 rows in set (0.00 sec)
```

Beispiel 35: Definition eines automatisch befüllten Primärschlüssels

Im Beispiel 35 wird der Wert der Spalte LfdNr, sofern nicht durch den Anwender explizit angegeben, automatisch ermittelt und eingefügt.

Zur Zugriffsbeschleunigung dienende Indexstrukturen können bereits zum Tabellenerstellungszeitpunkt durch den Anwender angegeben werden. Diese werden jedoch nicht der Spaltendefinition als nachgestellt, sondern bilden einen eigenen Eintrag innerhalb der Tabellendefinition.
Ein Index kann gleichzeitig eine oder mehrere Spalten umfassen. Beispiel Beispiel 36 zeigt dies:

```
mysql> CREATE TABLE Person(
    -> Name VARCHAR(20) PRIMARY KEY,
    -> GebDat DATE, Str_HsNr VARCHAR(20),
    -> PLZ CHAR(5),
    -> Ort VARCHAR(50),
    -> INDEX GebDatIdx (GebDat),
    -> INDEX AdresseIndex (Str_HsNr, PLZ, Ort));
Query OK, 0 rows affected (0.07 sec)
```

Beispiel 36: Definition von Indexen

Als beschränkende Verschärfung, die sich auch positiv auf die Zugriffsgeschwindigkeit auswirkt, kann ein Index als UNIQUE INDEX definiert werden. Er darf dann ausschließlich eindeutige Werte oder Wertkombinationen aufnehmen.

Erzeugung von Fremdschlüsselbeziehungen

Die Erzeugung von Fremdschlüsselbeziehungen ist das integrale Element zur Wahrung der referentiellen Integrität. Fremdschlüsselbeziehungen können bereits zum Erstellungszeitpunkt einer Tabelle angegeben werden wie Beispiel 37 zeigt oder nachträglich durch einen ALTER TABLE-Ausdruck hinzugefügt werden wie durch Beispiel 38 gezeigt.

```
CREATE TABLE DEPARTMENT(
   DNAME VARCHAR(20) NOT NULL,
   DNUMBER INTEGER(1) PRIMARY KEY,
   MGRSSN INTEGER(9),
   MGRSTARTDATE DATE);

CREATE TABLE EMPLOYEE(
   FNAME VARCHAR(10) NOT NULL,
   MINIT VARCHAR(1),
   LNAME VARCHAR(10) NOT NULL,
   SSN INTEGER(9) PRIMARY KEY,
   BDATE DATE,
   ADDRESS VARCHAR(30),
   SEX ENUM('M','F'),
   SALARY REAL(7,2) UNSIGNED,
   SUPERSSN INTEGER(9),
   DNO INTEGER(1) REFERENCES DEPARTMENT(DNUMBER),
   INDEX DNO_IDX (DNO));
```

Beispiel 37: Erzeugung von Fremdschlüsselbeziehungen zum Tabellenersellungszeitpunkt

Das Beispiel erzeugt neben der angestrebten Fremdschlüsselbeziehungen zwischen dem Attribut DNO der Tabelle EMPLOYEE und dem Primärschlüssel DNUMBER in DEPARTMENT einen Index auf den Fremdschlüssel innerhalb der Tabelle EMPLOYEE.
Dies ist für einige Datenbankmanagementsysteme (darunter MySQL) notwendig, um die Zugriffe auf den Fremdschlüssel zu beschleunigen.

Das nachfolgende Beispiel liefert dasselbe Ergebnis, jedoch unter nachträglicher (d.h. nach dem Erstellungszeitpunkt der Tabellen) Fremdschlüsselerzeugung:

```
CREATE TABLE DEPARTMENT(
   DNAME VARCHAR(20) NOT NULL,
   DNUMBER INTEGER(1) PRIMARY KEY,
   MGRSSN INTEGER(9),
   MGRSTARTDATE DATE);

CREATE TABLE EMPLOYEE(
   FNAME VARCHAR(10) NOT NULL,
   MINIT VARCHAR(1),
   LNAME VARCHAR(10) NOT NULL,
   SSN INTEGER(9) PRIMARY KEY,
   BDATE DATE,
   ADDRESS VARCHAR(30),
   SEX ENUM('M','F'),
   SALARY REAL(7,2) UNSIGNED,
   SUPERSSN INTEGER(9),
   DNO INTEGER(1));

ALTER TABLE EMPLOYEE ADD INDEX DNO_IDX (DNO);
ALTER TABLE EMPLOYEE ADD CONSTRAINT DNO_FK FOREIGN KEY (DNO) REFERENCES DEPARTMENT
(DNUMBER);
```

Beispiel 38: Nachträgliche Erzeugung von Fremdschlüsselbeziehungen

3.3 Der Anfrageteil von SQL

Anfragen zur Ermittlung von Datenbankinhalten stellen den eigentlichen Sprachkern von SQL und zweifellos den in der Praxis bedeutsamsten Anteil der Sprache dar.
Die gesamte Mächtigkeit des Anfrageteils von SQL erschließt sich durch das Schlüsselwort SELECT. Es gestattet Anfragen theoretisch unbegrenzter Komplexität in einer uniformen und leicht zu behaltenden Syntax zu formulieren, deren Mächtigkeit von einfachsten Anfragen bis zu aufwendigen Auswertungen reicht.

Anfragen von Datenbankinhalten

Die SQL-Anweisung SELECT dient der Abfrage von in einer Datenbank abgelegten Inhalten. Sie benötigt Wissen über die angelegten Tabellen sowie deren Struktur hinsichtlich Spalten und deren Typen.
Alle nachfolgenden Beispielanfragen beziehen sich, sofern nicht anders angegeben auf die Demodatenbank.

Die vereinfachte Syntax der SELECT-Anweisung lautet:

```
SELECT [ALL|DISTINCT] select_item,...
FROM table_specification,...
[WHERE search_condition]
[GROUP BY grouping_column,...]
[HAVING search_condition]
[ORDER BY sort_specification,...]
```

```
SELECT FNAME
FROM EMPLOYEE;
```

Beispiel 39: Einfache Anfrage

Die Anfrage liefert die Inhalte der Spalte FNAME aller in der Tabelle EMPLOYEE abgelegten Tupel. Die Werte werden in keiner vorgegebenen Reihenfolge ausgegeben, d.h. eine etwaige Sortierung ist zufallsbedingt und kann durch DBMS interne Reorganisationsprozesse zerstört werden.

Durch diese Anfrage wird als Resultat eine nicht in der Datenbank abgelegte Tabelle erzeugt, welche nur die im SELECT-Ausdruck angegebenen Spalten enthält. Die Ergebnistabelle stimmt zwar in Tupelanzahl mit der Ursprungstabelle überein, blendet jedoch einzelne Attribute aus. Diese Vorgang wird als *Projektion* bezeichnet.

Definition 42: *Projektion*
Die Projektion blendet einzelne Spalten aus.

```
SELECT FNAME, MINIT, LNAME, SSN, BDATE, ADDRESS, SEX, SALARY, SUPERSSN, DNO
FROM EMPLOYEE;
```

Beispiel 40: Anfrage aller Spalten einer Tabelle

Die Abfrage aus Beispiel 40 liefert die Wertinhalte aller Spalten (sie sind explizit nach dem Schlüsselwort `SELECT` angegeben) der Relation `EMPLOYEE`.
Als Besonderheit wird für das Attribut `SUPERSSN` des Mitarbeiters James E. Borg der Wert `NULL` ausgegeben. Diese Zeichenkette gibt an, daß für dieses Attribut der Relation kein Wert abgespeichert wurde.

Die schreibaufwendige und damit fehlerträchtige Explizierung aller Spalten einer Relation ist kaum praktikabel und überdies äußerst änderungssensitiv im Falle der Aufnahme neuer Spalten oder der Lösung Existierender.
Aus diesem Grunde kann statt des Spaltennamens ein Stern „*“ als Jokerzeichen stellvertretend für alle Spalten einer Relation angegeben werden.
Beispiel 41 zeigt dies als Umschreibung der Anfrage aus Beispiel 40:

```
SELECT *
FROM EMPLOYEE;
```

Beispiel 41: Anfrage aller Spalten einer Tabelle mit Jokerzeichen

Enthält die Ausgabe nicht den Primärschlüssel einer Tabelle, so kann es vorkommen, das mehrfach dieselben Werte ausgegeben werden. Dies kann durch Angabe des Schlüsselwortes `DISTINCT` in der `SELECT`-Klausel vermieden werden.
`DISTINCT` überschreibt das vorgegebene Verhalten (`ALL`) alle Einträge auszugeben.

```
SELECT DISTINCT SALARY
FROM EMPLOYEE;
```

Beispiel 42: Duplikatfreie Ausgabe aller verschiedenen Werte

Beispiel 42 liefert alle verschiedenen Werteinträge der Spalte `SALARY` duplikatfrei.

Durch Angabe mehrerer Einträge in der `FROM`-Klausel können Inhalte aus verschiedenen Tabellen innerhalb einer Anfrage extrahiert werden:

```
SELECT DNAME, PNUMBER
FROM DEPARTMENT, PROJECT;
```

Beispiel 43: Anfrage auf zwei Tabellen

Die Anfrage bildet das kartesische Produkt der beiden angefragten Tabellen.

Aliasbildung

Bei Anfragen über mehrere Tabellen kann es zu Problemen hinsichtlich der Eindeutigkeit der Spaltenbezeichner kommen. So würde die Anfrage aus Beispiel 44 nicht die für `ESSN` abgelegten Werte liefern, sondern den Fehler `ERROR 1052: Column: 'ESSN' in field list is ambiguous`, da in jeder der beiden in der Anfrage berücksichtigten Tabellen eine Spalte mit `ESSN` benannt ist.

```
SELECT ESSN, ESSN
FROM WORKS_ON, DEPENDENT;
```

Beispiel 44: Fehlerhafte Anfrage auf zwei Tabellen

Als Lösung bietet SQL die Möglichkeit den Spaltennamen zusätzlich durch Voranstellung des Namens der die Spalte beherbergenden Tabelle zu qualifizieren um die erforderliche Eindeutigkeit herzustellen:

```
SELECT WORKS_ON.ESSN, DEPENDENT.ESSN
FROM WORKS_ON, DEPENDENT;
```

Beispiel 45: Lösung des Mehrdeutigkeitsproblems bei Anfrage auf zwei Tabellen

Unter Nutzung der Möglichkeit Alternativnamen für Tabellen, sog. *Aliasnamen*, anzugeben ergibt sich eine in der Schreibung kompaktere Umsetzung:

```
SELECT w.ESSN, d.ESSN
FROM WORKS_ON AS w, DEPENDENT AS d;
```

Beispiel 46: Lösung des Mehrdeutigkeitsproblems bei Anfrage auf zwei Tabellen

Gleichzeitig kann die Aliasbildung eingesetzt werden, um die Benennung der Spalten bei der Ausgabe zu modifizieren. Auf diesem Wege können wenig sprechende Namen oder Doppelbenennungen umgangen werden.

```
SELECT w.ESSN AS "Mitarbeiter Sozialversicherungsnummer",
d.ESSN AS "Sozialversicherungsnummer des Verwandten"
FROM WORKS_ON AS w, DEPENDENT AS d;
```

Beispiel 47: Umbenennung von Ausgabespalten

Berechnete Ausgaben

Durch die Angabe einfacher arithmetischer Formeln in der SELECT-Klausel können vor ihrer Ausgabe Berechnungen auf den Werten aus der Datenbank angestellt werden.

```
SELECT FNAME, SALARY*12 as Jahreseinkommen
FROM EMPLOYEE;
```

Beispiel 48: Berechnungen I

Beschränkung der Ergebnismenge

Die bisher betrachteten Anfrageformen lieferten immer die gesamten Inhalte der betrachteten Tabellen. Durch Angabe einer einschränkenden Bedingung innerhalb der WHERE-Klausel einer SELECT-Anweisung können die Tabelleninhalte hinsichtlich einer Bedingung gefiltert werden.

Beispiel 49 liefert nur die abgespeicherten Werte für Geburtsdatum (BDATE) und Adresse (ADDRESS) derjenigen Mitarbeiter, deren Vornamen (FNAME) *John* ist.

```
SELECT BDATE, ADDRESS
FROM EMPLOYEE
WHERE FNAME="John";
```

Beispiel 49: Einschränkung der Anfrage

Durch die Filterung der Ergebnistupel werden alle diejenigen Datenbankeinträge, welche die getroffene Bedingung nicht erfüllen ausgeblendet. Das Ergebnis kann (sofern SELECT * ... gewählt wurde) zwar in der Anzahl Spalten mit der Ursprungsrelation übereinstimmen, wird dies jedoch typischerweise (d.h. außer im Falle, daß alle Tupel der Relation die formulierte Bedingung erfüllen) nicht tun.
Dieser Vorgang wird als *Selektion* bezeichnet.

Definition 43: *Selektion*
Die Selektion blendet einzelne Werteinträge aus..

Als relationale Operatoren für Vergleichstests zwischen zwei (möglicherweise einelementigen) Mengen stehen zur Verfügung:

Operator	Funktion	Bemerkung
=	Gleichheitstest	
<>	Test auf Ungleichheit	Teilweise (auch in MySQL!) ist der Standardoperator durch != ersetzt
<	Test auf kleiner	Liefert bei Zeichenkettendatentypen alle lexikalisch „kleineren“, d.h. diejenigen die in der alphabetischen Sortierung früher auftreten. Ebenso liefert der Operator bei der Anwendung auf Datumsdatentypen die kalendarisch früheren.
<=	Kleiner oder gleich	
>	Größer	
>=	Größer oder gleich	
IS NULL	Testet ob eine Spalte NULL enthält	
IS NOT NULL	Testet ob eine Spalte nicht NULL enthält	
BETWEEN	Testet ob ein Wert in vorgegebenen Grenzen liegt	
IN	Testet ob ein Wert innerhalb einer vorgegebenen Menge liegt	

Neben den einfachen Vergleichsoperationen können durch den LIKE-Operator unscharfe musterbasierte Suchen ausgedrückt werden.
Die Musterausdrücke werden dabei aus den tatsächlich in der Ergebnisemenge erwarteten Zeichen ergänzt um Metazeichen mit besonderer Bedeutung zusammengesetzt. Hierbei stehen „%“ zur Stellvertretung einer (möglicherweise leeren) Menge beliebiger Zeichen und „_“ zur Stellvertretung genau eines Zeichens zur Verfügung.

```
SELECT BDATE, ADDRESS
FROM EMPLOYEE
WHERE FNAME LIKE "J%";
```

Beispiel 50: Musterbasierte Anfrage I

Die Anfrage aus Beispiel 50 liefert die Werte der Spalten BDATE und ADDRESS aller Mitarbeiter deren Name (FNAME) mit einem „J“ beginnt.
Die Anfrage aus Beispiel 51 beschränkt die Suche zusätzlich auf diejenigen Namen, deren vorletztes Zeichen ein „e“ ist, d.h. diejenigen Einträge für die nach dem „e“ nur noch genau ein beliebiges Zeichen auftritt.

```
SELECT BDATE, ADDRESS
FROM EMPLOYEE
WHERE FNAME LIKE "J%e_";
```

Beispiel 51: Musterbasierte Anfrage II

Die Variante aus Beispiel 52 extrahiert alle Spalteninhalte der Mitarbeitertabelle deren Name aus genau fünf Zeichen besteht.

```
SELECT *
FROM EMPLOYEE
WHERE FNAME LIKE "____";
```

Beispiel 52: Musterbasierte Anfrage III

Kombination von Einzelbedingungen

Zur Selektion nach mehreren Bedingungen können diese mit den logischen Operationen AND, OR und NOT kombiniert werden.

Die Anfrage aus Beispiel 53 liefert die Namen aller Mitarbeiter, die in „Houston“ wohnen und weniger als 50000 verdienen.

```
SELECT FNAME
FROM EMPLOYEE
WHERE ADDRESS LIKE "%Houston%" AND SALARY < 50000;
```

Beispiel 53: Kombination von Bedingungen

Mittels der Verknüpfungsoperatoren können auch einige der zuvor gezeigten Vergleichsoperatoren ausgedrückt werden.

Vergleichsoperator	Alternative Schreibweise mit Bedingungsverknüpfung
a BETWEEN b and c	(a >= b) AND (a <= c)
x IN (a, b, c)	(x = a) OR (x = b) OR (x = c)

Für die Kombinationsoperatoren gilt, aufgrund der Möglichkeit des Auftretens von NULL-Werten, die dreiwertige Logik:

AND	TRUE	FALSE	NULL
TRUE	TRUE	FALSE	NULL
FALSE	FALSE	FALSE	FALSE
NULL	NULL	FALSE	NULL

OR	TRUE	FALSE	NULL
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	NULL
NULL	TRUE	NULL	NULL

NOT	TRUE	FALSE	NULL
	FALSE	TRUE	NULL

Kombination von Anfrageergebnissen

In manchen Fällen ist es gewünscht das Ergebnis eigenständiger Anfragen zu einem Ergebnis zu kombinieren. Hierzu kann das UNION-Schlüsselwort zur Verbindung der Einzelanfragen.

```
CREATE TABLE Artikel(
        ArtNo           VARCHAR(4) PRIMARY KEY,
        Bezeichnung VARCHAR(10) NOT NULL,
        Preis                   DECIMAL(5,2) NOT NULL);
CREATE TABLE Sonderpreise(
        ArtikelNo       VARCHAR(4) PRIMARY KEY,
        Bezeichnung     VARCHAR(10) NOT NULL,
        Sonderverkaufsgrund VARCHAR(20));

INSERT INTO Artikel VALUES("2222", "Blitz Superrein", 19.99);
INSERT INTO Artikel VALUES("1111", "Wusch Superfein", 99.95);

INSERT INTO Sonderpreise VALUES("4444", "Kratzweich", "Wasserschaden");
INSERT INTO Sonderpreise VALUES("3333", "Glanz Extraweiss", NULL);

SELECT ArtNo
FROM Artikel
UNION
SELECT ArtikelNo
FROM Sonderpreise;
```

Beispiel 54: Kombination mittels UNION

Die Anfrage aus Beispiel 54 erstellt zunächst zwei Tabellen und fügt einige Daten ein. Anschließend werden die Artikelnummern (Spalte ArtNo bzw. ArtikelNo) beider Tabellen angefragt und das Ergebnis mittels UNION zu einer Resultattabelle kombiniert.

Voraussetzung der Kombinierbarkeit ist die Typgleichheit der selektierten und zu vereinigenden

Attribute (im Beispiel beide vom Typ VARCHAR(4)).

Implizit führt die Verwendung von UNION sowohl die Sortierung der Ergebnissmenge als auch die Duplikatentfernung daraus herbei.

Verbünde

Häufig besteht der Wunsch Werte aus verschiedenen Tabellen nicht nur gemeinsam abzufragen und anzuzeigen, sondern auch inhaltlich in Beziehung zu setzen.

Die Anfrage aus Beispiel 55 versucht durch Abfrage der Tabellen EMPLOYEE und DEPARTMENT die Namen der Mitarbeiter und die (in der anderen Tabelle abgelegte) Bezeichnung Abteilung zu ermitteln die sie beschäftigen.
Aufgrund der Bildung des kartesischen Produkts werden jedoch alle theoretisch möglichen Kombinationen geliefert und nicht die Untermenge der tatsächlich existierenden Paarungen.

```
SELECT FNAME, DNAME
FROM DEPARTMENT, EMPLOYEE;
```

Beispiel 55: Fehlerhafte Verbundbildung

Durch (geschickte) Nutzung der WHERE-Bedingung, die Werte aus beiden Tabellen miteinander in Beziehung setzt, gelingt jedoch die gewünschte Ermittlung:

```
SELECT FNAME, DNAME
FROM DEPARTMENT AS d, EMPLOYEE as e
WHERE d.DNUMBER = e.DNO;
```

Beispiel 56: Innerer Verbund

Das Beispiel 56 liefert lediglich diejenigen Tupel, für die das in EMPLOYEE abgespeicherte DNO-Attribut einen Wert enthält, der auch in der Spalte DNUMBER der Tabelle DEPARTMENT auftritt.

Definition 44: *Innerer Verbund*
Ein Innerer Verbund enthält die selektierten Daten aller beteiligten Tabellen, welche die formulierte Einschränkungsbedingung erfüllen.

Der SQL-Standard gibt für diese besondere Anfrageform eine eigene Syntax vor:

```
SELECT FNAME, DNAME
FROM DEPARTMENT AS d INNER JOIN EMPLOYEE AS e
ON d.DNUMBER = e.DNO;
```

Beispiel 57: Innerer Verbund in Standardnotation

Die Bildung von Verbünden ist nicht auf die Angabe verschiedener Tabellen beschränkt, sondern kann auch durch mehrfache Bezugnahme auf dieselbe Tabelle geschehen:

```
SELECT e1.FNAME as Chef, e2.FNAME as Mitarbeiter
FROM EMPLOYEE AS e1, EMPLOYEE AS e2
WHERE e1.SSN = e2.SUPERSSN;
```

Beispiel 58: Innerer Verbund unter mehrfacher Nutzung derselben Tabelle

Beispiel 59 zeigt ein Beispiel der Bildung eines inneren Verbundes unter Zugriff auf drei Tabellen. Die Anfrage liefert die Familiennamen (Tabelle EMPLOYEE) sowie die Abteilungen denen der Mitarbeiter zugeordnet ist (aus Tabelle DEPARTMENT) sowie die durch den Mitarbeiter bearbeiteten Projekte (Tabelle PROJECT).
Die Tabelle PROJECT kann jedoch nicht direkt in den Verbund einbezogen werden, da sie über keine geeigneten Attribute (d.h. Attribute die mit derselben Semantik in einer der beiden anderen Tabellen auftreten) verfügt. Daher wird zusätzlich die Tabelle WORKS_ON in die Anfrage miteinbezogen, weil sie mit dem Attribut ESSN ein Attribut bietet, welches die in EMPLOYEE enthaltene Attribut SSN als Fremdschlüssel beinhaltet. Ausgehend hiervon kann eine Bedingung

unter Einbezug von PROJECT formuliert werden.

```
SELECT FNAME, DNAME, PNAME
FROM EMPLOYEE AS e, DEPARTMENT AS d, PROJECT AS p, WORKS_ON AS w
WHERE d.DNUMBER = e.DNO AND e.SSN = w.ESSN AND w.PNO = p.PNUMBER;
```

Beispiel 59: Innerer Verbund dreier Tabellen

Für Verbünde ist die Bildung durch ausschließliche Nutzung des Gleichheitsoperators innerhalb der WHERE-Klausel keineswegs zwingend, wenngleich diese sog. *Equi Joins* eine häufige Anwendungsform darstellen.
Beispiel 1 zeigt einen durch Nutzung des kleiner-Operators gebildeten Verbund, der alle Abteilungen enthält, in denen ein Mitarbeiter (noch) nicht arbeitet und deren Abteilungsnummer größer ist als die Nummer der Abteilung welcher der Mitarbeiter gegenwärtig zugeordnet ist. (Mögliche semantische Deutung: Liste möglicher Beförderungen, sofern größere Abteilungsnummern einen Aufstieg codieren.

```
SELECT FNAME, DNAME
FROM EMPLOYEE JOIN DEPARTMENT
ON DEPARTMENT.DNUMBER < EMPLOYEE.DNO;
```

Beispiel 60: Non-Equi-Join

Äußere Verbunde

Neben der Möglichkeit durch innere Verbünde Tupel die über Attribute mit übereinstimmenden Wertbelegungen zu selektieren besteht durch *äußere Verbünde* die Möglichkeit neben den Tupeln mit übereinstimmenden Werten alle Tupel einer am Verbund beteiligten Tabelle vollständig zu selektieren.

Definition 45: *Äußerer Verbund*
Ein Äußerer Verbund enthält die selektierten Daten aller beteiligten Tabellen, welche die formulierte Einschränkungsbedingung erfüllen, sowie alle Daten der „äußeren“ Tabelle.
Die nicht mit Werten belegbaren Felder werden durch NULL aufgefüllt.

Konzeptionell wird zwischen *linken* und *rechten Äußeren Verbünden* unterschieden. Die „Seite“ des Verbundes gibt diejenige beteiligte Tabelle an, die im Rahmen der Verbundbildung vollständig ausgegeben wird.
Beispiel 61 zeigt ein Beispiel eines linken Äußeren Verbundes, Beispiel 62 illustriert einen rechten äußeren Verbund.

```
INSERT INTO EMPLOYEE VALUES("John", "X", "Doe", "999999999", "1965-03-04", "42 XYZ
Street", "M", 50000, NULL, NULL);

SELECT FNAME, DNAME
FROM EMPLOYEE LEFT OUTER JOIN DEPARTMENT
ON DEPARTMENT.DNUMBER = EMPLOYEE.DNO;
```

Beispiel 61: Linker Äußerer Verbund

Das Beispiel fügt zunächst einen Tupel zur Tabelle EMPLOYEE hinzu, der keiner Abteilung zugeordnet ist. In einem Inneren Verbund erscheint dieser Tupel daher nicht. Der linke Äußere Verbund des Beispiels hingegen umfaßt alle Tupel aus *EMPLOYEE* sowie die Werte der hinsichtlich der Bedingung DEPARTMENT.DNUMBER = EMPLOYEE.DNO ermittelten Übereinstimmungen in DEPARTMENT.
Für die nicht ermittelbaren Übereinstimmungen werden NULL-Werte erzeugt.

```
INSERT INTO DEPARTMENT VALUES("New Dept.", 0, 888665555, NULL);

SELECT FNAME, DNAME
FROM EMPLOYEE RIGHT OUTER JOIN DEPARTMENT
ON DEPARTMENT.DNUMBER = EMPLOYEE.DNO;
```

Beispiel 62: Rechter Äußerer Verbund

Auch das Beispiel 62 zum rechten Äußeren Verbund fügt zunächst einen Datensatz ein; diesmal in die Tabelle DEPARTMENT, der zu keinem Tupel in EMPLOYEE in Beziehung steht. Analog dem linken Äußeren Verbund liefert der rechte Äußere Verbund alle Tupel der rechtsstehenden Tabelle (DEPARTMENT) sowie die mit *EMPLOYEE* übereinstimmenden.

Kreuzverbund

Der Kreuzverbund liefert alle gemäß den Gesetzen des kartesischen Produkts bildbaren Kombinationen aus Tupeln der beitragenden Relationen:

```
SELECT DNAME, PNUMBER
FROM DEPARTMENT CROSS JOIN PROJECT;
```

Beispiel 63: Kreuzverbund

Das Beispiel entspricht damit im Ergebnis der Anfrage aus Beispiel 43.

Wird beim Kreuzverbund eine Bedingung angegeben, so entspricht er dem Inneren Verbund. Das Ergebnis der Anfrage aus Beispiel 64 ist daher identisch zum inneren Verbund aus Beispiel 56.

```
SELECT FNAME, DNAME
FROM DEPARTMENT CROSS JOIN EMPLOYEE
WHERE DEPARTMENT.DNUMBER = EMPLOYEE.DNO;
```

Beispiel 64: Kreuzverbund mit Bedingung

Artikel von Satya Komatineni: The Effective Use of Joins in Select Statements

Sortierungen

Zur Sortierung hinsichtlich einer oder mehrerer Spalten der als Anfrageergebnis ermittelten Tabelle steht die ORDER BY-Klausel zur Verfügung.
Beispiel 65 zeigt die Anwendung zur lexikalischen Sortierung:

```
CREATE TABLE Person(
        Vorname VARCHAR(10),
        Nachname VARCHAR(10));

INSERT INTO Person VALUES("Adam", "C-Mann");
INSERT INTO Person VALUES("Cesar", "C-Mann");
INSERT INTO Person VALUES("Berta", "C-Mann");
INSERT INTO Person VALUES("Adam", "A-Mann");
INSERT INTO Person VALUES("Cesar", "A-Mann");
INSERT INTO Person VALUES("Berta", "A-Mann");
INSERT INTO Person VALUES("Adam", "B-Mann");
INSERT INTO Person VALUES("Cesar", "B-Mann");
INSERT INTO Person VALUES("Berta", "B-Mann");

SELECT *
FROM Person
ORDER BY Nachname;
```

Beispiel 65: Sortierung

Ist die Sortierung bezüglich mehrerer Attribute, d.h. Sortierung innerhalb eines gleicher Attributwerte hinsichtlich eines anderen Attributs, gewünscht, so können auch mehrere Sortierattribute in der ORDER BY-Klausel versammelt werden.
Zusätzlich zeigt das Beispiel die Kurzschreibweise, welche die zu sortierenden Attribute nicht namentlich benennt, sondern nur hinsichtlich ihrer Position innerhalb der SELECT-Klausel referenziert.

```
CREATE TABLE Person(
        Vorname VARCHAR(10),
        Nachname VARCHAR(10));

INSERT INTO Person VALUES("Adam", "C-Mann");
INSERT INTO Person VALUES("Cesar", "C-Mann");
INSERT INTO Person VALUES("Berta", "C-Mann");
INSERT INTO Person VALUES("Adam", "A-Mann");
INSERT INTO Person VALUES("Cesar", "A-Mann");
INSERT INTO Person VALUES("Berta", "A-Mann");
INSERT INTO Person VALUES("Adam", "B-Mann");
INSERT INTO Person VALUES("Cesar", "B-Mann");
INSERT INTO Person VALUES("Berta", "B-Mann");

SELECT *
FROM Person
ORDER BY 2,1;
```

Beispiel 66: Sortierung bezüglich mehrerer Attribute

Vorgabegemäß erfolgt die Sortierung aufsteigend (*ascending*). Die Umkehrung der Sortierreihenfolge kann durch nachstellen der Zeichenfolge DESC (für *descending*) nach dem Namen des Sortierattributes erreicht werden.
Die aufsteigende Vorgabesortierung (ASC) wird üblicherweise nicht ausgeschrieben, ist aber im Beispiel 67 zur besseren Verdeutlichung expliziert.

```
CREATE TABLE Person(
        Vorname VARCHAR(10),
        Nachname VARCHAR(10));

INSERT INTO Person VALUES("Adam", "C-Mann");
INSERT INTO Person VALUES("Cesar", "C-Mann");
INSERT INTO Person VALUES("Berta", "C-Mann");
INSERT INTO Person VALUES("Adam", "A-Mann");
INSERT INTO Person VALUES("Cesar", "A-Mann");
INSERT INTO Person VALUES("Berta", "A-Mann");
INSERT INTO Person VALUES("Adam", "B-Mann");
INSERT INTO Person VALUES("Cesar", "B-Mann");
INSERT INTO Person VALUES("Berta", "B-Mann");

SELECT *
FROM Person
ORDER BY 2 ASC,1 DESC;
```

Beispiel 67: Auf- und Absteigende Sortierung bezüglich mehrerer Attribute

Unteranfragen

Bisher wurden Anfragen lediglich auf Tabellen in ihrer Rolle als in der Datenbank abgelegte Eingabemengen betrachtet. Die relationale Sichtweise erfordert jedoch keineswegs, daß die Eingangswerte einer Anfrage direkt aus der Datenbank gelesen werden müssen. Sie können auch Ergebnis einer weiteren Anfrage sein.

Anfragen die vor einer anderen Anfrage ausgeführt werden müssen um für diese Eingangswerte zu liefern werden daher als *Unterabfragen* (*subqueries* oder *nested queries*) bezeichnet.

Das Beispiel 68 zeigt eine solche Unteranfrage die alle Projektnummern liefert welche Projekten zugeordnet sind die in der durch *Smith* gleiteten Abteilung bearbeitet werden. Eine zweite Unterabfrage des Beispiels liefert alle Nummern von Projekten an denen dieser Mitarbeiter selbst arbeitet.
Die durch diese Abfrage gelieferten Daten (Projektnummern) sind Eingangsdaten in die Ermittlung der Projektnamen.

```
SELECT DISTINCT PNAME
FROM PROJECT
WHERE PNUMBER IN (
   SELECT PNUMBER
   FROM PROJECT AS p, DEPARTMENT AS d, EMPLOYEE AS e
   WHERE e.SSN = d.MGRSSN AND
   d.DNUMBER = p.DNUM AND
   e.LNAME="Smith")
      OR
         PNUMBER IN (SELECT PNO
            FROM WORKS_ON AS w, EMPLOYEE AS e
            WHERE w.ESSN = e.SSN AND
            e.LNAME="Smith");
```

Beispiel 68: Unterabfrage I

Beispiel 69 zeigt den Vergleich eines Einzelwertes (SALARY) mit einer Menge gelieferter Werte. Die Anfrage ermittelt diejenigen Mitarbeiter, deren Einkommen höher liegt als das Einkommen aller Mitarbeit in Abteilung Nummer 5. (Hinweis es wird nicht ermittelt ob das Einkommen größer ist als die Summe aller Einkommen der Mitarbeiter aus Abteilung 5, sondern nur ob das Einkommen größer ist als jedes Einzeleinkommen eines Mitarbeiters aus Abteilung 5.)

```
SELECT LNAME, FNAME
FROM EMPLOYEE
WHERE SALARY > ALL (SELECT SALARY FROM EMPLOYEE WHERE DNO=5);
```

Beispiel 69: Unterabfrage II

Korrelierte Unteranfragen

Eine besondere Form der Unteranfragen stellen solche dar, die sich in ihrer WHERE-Klausel auf die äußere Anfrage beziehen.
Diese Form der Anfrageschachtelung wird auch als *korrelierte Unteranfrage* bezeichnet.

Das Beispiel 70 zeigt eine solche Anfrage, die alle Verwandten (DEPENDENT) ermittelt, die das selbe Geschlecht haben wie der in der Tabelle EMPLOYEE erfaßte Mitarbeiter.

```
SELECT e.FNAME, e.LNAME
FROM EMPLOYEE AS e
WHERE e.SSN IN (SELECT ESSN
                                        FROM DEPENDENT
                                        WHERE e.SEX = SEX);
```

Beispiel 70: Korrelierte Unterabfrage

Jede korrelierte Unterabfrage kann durch Umschreibung in eine nicht-korrelierte Fassung überführt werden. So lautet die Formulierung des aus Beispiel 70 ohne geschachtelte Unterabfrage:

```
SELECT e.FNAME, e.LNAME
FROM EMPLOYEE AS e, DEPENDENT AS d
WHERE e.SSN = d.ESSN AND
e.SEX = d.SEX;
```

Beispiel 71: Auflösung der korrelierten Unterabfrage

Die Formulierung als geschachtelte Unterabfrage ist damit nicht zwingend notwendig, kann jedoch aus Gründen der Übersichtlichkeit gewünscht sein.

Die nähere Betrachtung der Anfragen aus Beispiel 70 und Beispiel 71 zeigen, daß die aus der Tabelle DEPENDENT angefragten Daten lediglich zur Formulierung der Bedingung, nicht jedoch zur Ausgabe herangezogen werden. Daher läßt sich die Bedingung unter Verwendung des EXISTS-Operators umschreiben zu:

```
SELECT e.FNAME, e.LNAME
FROM EMPLOYEE AS e
WHERE EXISTS (  SELECT *
                FROM DEPENDENT
                WHERE e.SSN = ESSN AND
                e.SEX = SEX);
```

Beispiel 72: Korrelierte Unterabfrage mit EXISTS

EXISTS liefert den Boole'schen Wahrheitswert immer dann, wenn die (Unter-)Abfrage eine nichtleere Menge ist, d.h. Daten enthält.

Anfragen die EXISTS oder IN beinhalten können auch durch linke Äußere Verbünde ausgedrückt werden, wie Beispiel 73 zeigt:

```
SELECT e.FNAME, e.LNAME
FROM EMPLOYEE AS e LEFT JOIN DEPENDENT AS d
ON e.SSN = d.ESSN AND e.SEX = d.SEX
WHERE d.SEX IS NOT NULL;
```

Beispiel 73: Korrelierte Unterabfrage ausgedrückt als linker äußerer Verbund

Eine ähnliche Funktion wie die EXISTS-Operation stellt ANY bereit, jedoch liefert diese die durch die Unterabfrage angefragten Tupel zurück um sie an eine Bedingung zu knüpfen.
Beispiel 74 zeigt die Ermittlung der Namen derjenigen Mitarbeiter, die mehr als irgendein beliebiger Manager verdienen.

```
SELECT FNAME
FROM EMPLOYEE
WHERE SALARY > ANY (SELECT SALARY
   FROM EMPLOYEE
   WHERE SSN IN (SELECT SUPERSSN
      FROM EMPLOYEE));
```

Beispiel 74: Unterabfrage unter Verwendung von ANY

Aggregatfunktionen und Gruppierung

Über die Sortierung hinausgehend ist oftmals ein bestimmte Anordnung der durch eine Anfrage ermittelten Ergebnistupel gewünscht, etwa als inhaltliche Gruppierung.
Gleichzeitig sind oft quantiative Aussagen über Eigenschaften der Resultatmenge --- wie größter oder kleinster Wert sowie Summen- oder Durchschnittsbildung --- gewünscht.

Beispiel 1 zeigt die Ermittlung der Summe aller Gehälter (SQL-Funktion SUM) sowie des Maximal- (MAX), Minimal- (MIN) und Durchschnittsgehalts (AVG) für die Mitarbeiter der *Research*-Abteilung. Die genannten SQL-Funktionen werden als *Aggregierungsfunktionen* bezeichnet, da sie die durch die Abfrage ermittelten Einzelwerte (d.h. die Einträge der Spalte SALARY) jeweils zu genau einer Aussage verdichten.

```
SELECT SUM(SALARY), MAX(SALARY), MIN(SALARY), AVG(SALARY)
FROM EMPLOYEE, DEPARTMENT
WHERE DNO = DNUMBER AND DNAME="Research";
```

Beispiel 75: Aggregierungsfunktionen

Mit der Funktion COUNT steht eine Möglichkeit zur Ermittlung der Mächtigkeit einer Tupelmenge zur Verfügung. Beispiel Beispiel 76 zeigt ihre Verwendung zur Ermittlung der Anzahl der Mitarbeiter der mit *Research* bezeichneten Abteilung.

```
SELECT COUNT(*)
FROM EMPLOYEE, DEPARTMENT
WHERE DNO = DNUMBER AND DNAME = "Research";
```

Beispiel 76: Zählfunktion I

Als Argument der COUNT-Funktion kann mit DISTINCT ein Schlüsselwort angegeben werden, welches die ausschließliche Zählung verschiedener Werte erwirkt.
Die Anfrage aus Beispiel Beispiel 77 ermittelt durch Nutzung dieses Schlüsselwortes die Anzahl der verschiedenen Werte in der Spalte SALARY.

```
SELECT COUNT(DISTINCT SALARY)
FROM EMPLOYEE;
```

Beispiel 77: Zählfunktion II

Häufig wird, wie in Beispiel 78 gezeigt, eine Anfrage zur Ermittlung der Anzahl als Unterabfrage formuliert und in der umgebenden Hauptabfrage mit einer Bedingung versehen.

```
SELECT LNAME, FNAME
FROM EMPLOYEE
WHERE (SELECT COUNT(*)
                FROM DEPENDENT
                WHERE SSN=ESSN) >= 2;
```

Beispiel 78: Eingebettete Zählfunktion

Neben den bisher gezeigten aggregierten Aussagen über eine Gesamtmenge besteht oftmals der Wunsch nach von Ermittlung Aussagen dieses Stils über bestimmte Werteklassen innerhalb der betrachteten Gesamtmenge. Hierzu dienen Gruppierungen der Ausgangsmenge, auf welche dann die verschiedenen Aggregierungsfunktionen separat angewandt werden können.
Beispiel Beispiel 79 zeigt dies für die Ermittlung der Mitarbeiteranzahl pro Abteilung sowie der Berechnung des abteilungsinternen Durchschnittsgehalts.

```
SELECT d.DNAME AS "Abteilung", COUNT(*) AS "Anzahl Mitarbeiter", AVG(SALARY) AS
"Durchschnittsgehalt"
FROM EMPLOYEE AS e, DEPARTMENT AS d
WHERE e.DNO = d.DNUMBER
GROUP BY DNO;
```

Beispiel 79: Gruppierung

Zur Realisierung wird die GROUP BY-Klausel verwendet, welche die Angabe eines oder mehrerer Attribute zuläßt anhand der die selektierte Menge partitioniert werden soll.

Die Anfrage des Beispiels Beispiel 80 zeigt die Nutzung einer Verbundbedingung innerhalb einer Gruppierungsanfrage, die Projektnummer und -name sowie vermöge der COUNT-Funktion die Anzahl der das Projekt bearbeitenden Mitarbeiter ermittelt.

```
SELECT PNUMBER, PNAME, COUNT(*) AS "Anzahl Mitarbeiter"
FROM PROJECT, WORKS_ON
WHERE PNUMBER = PNO
GROUP BY PNUMBER, PNAME;
```

Beispiel 80: Gruppierung mit Verbundbedingung

Durch zusätzliche Angabe der HAVING-Klausel kann die Menge der Gruppierungsresultate mittels einer Bedingung beschränkt werden.
So ermittelt die Anfrage aus Beispiel 81 dieselben Resultat wie die in Beispiel 80 gezeigte, jedoch nur für Projekte deren Mitarbeiteranzahl größer 2 ist.

```
SELECT PNUMBER, PNAME, COUNT(*)
FROM PROJECT, WORKS_ON
WHERE PNUMBER = PNO
GROUP BY PNUMBER, PNAME
HAVING COUNT(*) > 2;
```

Beispiel 81: Bedingte Gruppierung

Die formulierte Beschränkung wirkt sich nicht auf die zur Berechnung herangezogene Grundgesamtheit, sondern lediglich auf die Ausgabe der Gruppierungsergebnisse aus, die vor der Auswertung der in der HAVING-Klausel formulierten Bedingung berechnet werden müssen.
Zur Beschränkung der zur Berechnung heranzuziehenden Grundgesamtheit steht auch unter Nutzung der GROUP BY-Klausel der durch WHERE formulierte Bedingungsteil der SELECT-Anfrage zur Verfügung.

```
SELECT PNUMBER, PNAME, COUNT(*)
FROM PROJECT, WORKS_ON, EMPLOYEE
WHERE PNUMBER = PNO AND SSN = ESSN AND DNO=5
GROUP BY PNUMBER, PNAME;
```

Beispiel 82: Beschränkung der Gruppierungseingangsdaten

Gruppierungsschritte können auch in Unterabfragen auftreten, wie das Beispiel 83 zur Ermittlung des Abteilungsnamens und der Anteil der darin arbeitenden Personen mit einem Gehalt über 40000 für alle Abteilungen mit mindestens 2 Mitgliedern zeigt:

```
SELECT DNAME, COUNT(*)
FROM DEPARTMENT, EMPLOYEE
WHERE DNUMBER = DNO AND SALARY > 4000 AND DNO IN (
                SELECT DNO
                FROM EMPLOYEE
                GROUP BY DNO
                HAVING COUNT(*) > 2)
GROUP BY DNUMBER;
```

Beispiel 83: Gruppierung in Unterabfrage

Der Datenmanipulationsteil von SQL

Neben den bisher betrachteten Eigenschaften der Sprache SQL zur Definition von Datenbankstrukturen und zur Abfrage von Datenbankinhalten stehen auch Befehle zur Manipulation in Form von Einfüge-, Aktualisierung- und Löschoperationen zur Verfügung.

Der Einfügebefehl *INSERT*

Zum Hinzufügen neuer Tupel in eine bestehende Tabelle durch Angabe von Werten für einen oder mehrere Spalten dieser Tabelle wird der Befehl INSERT anboten.
Typischerweise wird dieser Befehlstyp zum Einfügen neuer Datensätze in bestehende Tabellen laufender Applikationen, ebenso wie zur Übernahme kompletter Datenbestände aus existierenden Datenquellen oder zur Neuladung einer Datenbank im Rahmen der Wiederherstellungsprozesses nach einem Systemausfall mit Datenverlust verwendet.

Die allgemeine Syntax des INSERT-Ausdruckes lautet:

```
INSERT INTO tbl_name (col_name,...)? VALUES(constant|NULL ...)
```

```
INSERT INTO EMPLOYEE VALUES(
        'John',
        'B',
        'Smith',
        123456789,
        '1965-01-09',
        '731 Fondren, Houston, TX',
        'M',
        30000,
        333445555,
        5);
```

Beispiel 84: Einfügen eines vollständigen Tupels

Beispiel 84 zeigt den Befehl zur Erzeugung eines neuen Eintrages in der Tabelle EMPLYOEE der Demodatenbank. Die Aufzählung der einzufügenden Werte ist vollständig, d.h. für jede Spalte der Tabelle wird explizit ein konstanter Wert angegeben. Per Konvention müssen alle nichtnumerischen Werte in einfache oder doppelte Hochkommata eingeschlossen werden. Hierunter fallen neben den Zeichenkettentypen auch alle Datumstypen.

Eine Sonderstellung innerhalb der angebbaren Konstanten zur Eintragung stellt die Zeichenkette NULL dar. Sie repräsentiert explizit fehlende Werte, deren Tabelleneinträge entsprechend gekennzeichnet werden. Zur Abgrenzung von der Zeichenkette *NULL* wird diese Angabe nicht in Anführungzeichen eingeschlossen, selbst wenn es sich um eine Spalte eines Zeichenkettentypen handelt.
Beispiel 85 zeigt eine exemplarische Befehlskonstruktion:

```
INSERT INTO EMPLOYEE VALUES(
        'James',
        'E',
        'Borg',
        888665555,
        '1937-11-10',
        '450 Stone, Houston, TX',
        'M',
        55000,
        NULL,
        1);
```

Beispiel 85: Einfügen eines vollständigen Tupels mit NULL-Wert

Neben der Möglichkeit vollständige Tupel einzufügen, kann durch explizite Angabe der einzufügenden Spalten auch eine partielle Befüllung des neu erzeugten Tupels vorgenommen werden.
Für die im INSERT-Befehl nicht angegebenen Spalten wird der spezifizierte Vorgabewert oder NULL eingefügt.
Beispiel 86 zeigt dies exemplarisch anhand des Einfügens der drei Attribute FNAME, LNAME und SSN. Gleichzeitig stellt das Beispiel auch heraus, daß bei expliziter Angabe der einzufügenden Spalten die gewählte Reihenfolge von der in der Tabelle realisierten abweichen kann.

```
INSERT INTO EMPLOYEE (LNAME, FNAME, SSN) VALUES(
        'Doe',
        'John',
        912873465);
```

Beispiel 86: Einfügen eines unvollständigen Tupels

Prinzipiell kann jedes Element der Potenzmenge der Attribute einer Relation eingefügt werden. Es muß jedoch zwingend einen Wert für das Primärschlüsselattribut enthalten, da hierfür der Wert NULL nicht gesetzt werden darf.

Der Aktualisierungsbefehl *UPDATE*

Zur Aktualisierung von Werten innerhalb bestehender Datenbankeinträge bietet der SQL-Sprachumfang den Befehl UPDATE an, der es gestattet frei wählbare Mengen von Tupeln einer Tabelle zu modifizieren.

Die allgemeine Syntax des Befehls lautet:

```
UPDATE tbl_name SET col_name=expression, ... [WHERE search_condition]
```

Beispiel Beispiel 87 zeigt den Befehl zur null-Setzung aller in der Tabelle EMPLOYEE verwalteten Geburtsdaten (BDATA):

```
UPDATE EMPLOYEE SET BDATE=NULL;
```

Beispiel 87: Modifikation aller Tupel durch Setzen eines konstanten Wertes

Neben der Eintragung von Konstanten können auch neue Inhalte aus den Bisherigen errechnet werden. So zeigt Beispiel 88 eine Aktualisierung, die das Gehalt (SALARY) aller Mitarbeiter um zehn Prozent erhöht:

```
UPDATE EMPLOYEE SET Salary=Salary*1.1;
```

Beispiel 88: Modifikation aller Tupel durch Setzen eines berechneten Wertes

Durch Nutzung der, identisch zum SELECT-Ausdruck aufgebauten, WHERE-Klausel kann die Menge der von der Änderung betroffenen Datensätze eingeschänkt werden.
Das Beispiel Beispiel 89 ändert in allen Einträgen, deren LNAME auf Zelaya lautet den Wert zu Jones.
Die Anzahl der betroffenen Tupel ist durch den UPDATE-Ausdruck nicht festlegbar, sondern richtet sich ausschließlich nach der durch die WHERE-Klausel selektierten Eintragsmenge.

```
UPDATE EMPLOYEE SET LNAME='Jones'
WHERE LNAME='Zelaya';
```

Beispiel 89: Modifikation von Tupeln

Durch die Nutzbarkeit der vollständigen Möglichkeiten der aus dem SELECT-Befehl bekannten Mächtigkeit der WHERE-Klausel lassen sich selbst komplexe Aktualisierungen realiseren.
Beispiel 90 zeigt führt die Erhöhung der Gehälter derjenigen Mitarbeiter durch, die Abteilungen zugewiesen sind, die mehr als zwei Projekte bearbeiten.

```
UPDATE EMPLOYEE SET SALARY=SALARY*1.1 WHERE DNO IN
        (SELECT DNUMBER
        FROM PROJECT AS p, DEPARTMENT AS d
        WHERE d.DNUMBER=p.DNUM
        GROUP BY 1
        HAVING COUNT(*) > 2);
```

Beispiel 90: Modifikation von Tupeln (Ermittlung der betroffenen Tupel durch Subanfrage)

***Der Löschbefehl** DELETE*

Zur Löschung von verwalteten Tupeln aus einer Tabelle existiert der DELETE-Befehl, der die betroffenen Datensätze ohne weite Nachfrage entfernt.

Seine allgemeine Syntax lautet:
`DELTE FROM tbl_name [WHERE search_condition]`

Die einfachste Ausprägung der DELETE-Anweisung löscht alle Tupel einer Tabelle:

```
DELETE FROM EMPLOYEE;
```

Beispiel 91: Löschen aller Tupel einer Tabelle

Durch Angabe der WHERE-Klausel können, wie bereits bei UPDATE für die zu aktualisierenden Tupel gezeigt, die zu löschenden Tupel eingegrenzt werden.
So entfernt der Ausdruck aus Beispiel 92 alle Mitarbeiter die in Houston wohnen.

```
DELETE FROM EMPLOYEE
WHERE ADDRESS LIKE "%Houston%";
```

Beispiel 92: Löschen aller Mitarbeiter, die in Houston wohnhaft sind

Durch die Nutzung der expliziten Mengenangabe innerhalb der WHERE-Klausel läßt sich die Menge der zu entfernenden Datensätze statische eingrenzen wie Beispiel 93 zeigt.

```
DELETE EMPLOYEE
WHERE SSN IN (333445555, 888665555, 987987987);
```

Beispiel 93: Löschen bestimmter Datenstätze

Beispiel 94 zeigt die Nutzung einer Unterabfrage zur Ermittlung aller Abteilungen, die nur genau ein Projekt durchführen und anschließenden Löschung dieser Abteilungen aus der Tabelle DEPARTMENT.

```
DELETE FROM DEPARTMENT
WHERE DNUMER IN
        (SELECT DNUM
        FROM PROJECT
        GROUP BY 1
        HAVING COUNT(*) = 1);
```

Beispiel 94: Löschen aller Abteilungen, die nur genau ein Projekt durchführen

Definitionsverzeichnis

Assoziation
Assoziationstyp
Äußerer Verbund
Boyce/Codd-Normalform
Daten
Datenbank
Datenbankmanagementsystem (DBMS)
Datenbanksprache
Datenunabhängigkeit
Dritte Normalform (3NF)
Entität
Entitätstyp
Erste Normalform (1NF)
Fünfte Normalform
Hybrider Entitäts-Assoziationstyp
Index
Information
Innerer Verbund
Kardinalitätsintervall
Konzeptuelles Schema
Logisches Schema
Mehrwertige Abhängigkeit
Metainformation
Modell
NULL-Wert
Physisches Schema
Primärschüssel
Projektion
Referentielle Integrität
Relation
Relationales DBMS
Repräsentation
Repräsentationstyp
Rolle
Schüssel
Selektion
Spezialisierungsassoziationstyp
Superschlüssel
Tabelle
Transitive Abhängigkeit
Triviale mehrwertige Abhängigkeit
Vierte Normalform
Volle funktionale Abhängigkeit
Vollständiges konzeptuelles Schema
Zweite Normalform (2NF)

Schlagwortverzeichnis

Abbildungsverzeichnis

Verzeichnis der Beispiele

Organisatorisches

Die Vorlesung findet dienstags von 7.45 bis 11 Uhr im Raum A266 statt.
Praktikum wird mittwochs von 7.45 bis 9.15 Uhr im Raum C-101 angeboten.

Die schriftliche Abschlußklausur findet am 2004-07-14 um 9.30 Uhr im Raum A226 statt.

Abstract

Die Vorlesung vermittelt einführend die Grundlagen relationaler Datenbanken und ihres Entwurfs.
Hierzu wird anhand eines Beispiels zunächst ein konkretes Datenbankmanagementsystem betrachtet und erste Erfahrungen durch Arbeiten am System gesammelt.
Darauf aufbauend werden die Themenkomplexe *Datenbankentwurf* und praktische Arbeit mit *SQL* am System vertieft.

Als Entwurfstechnik wird neben dem graphischen Entity-Relationship Modell ein Algorithmus zur automatisierten Ableitung logischer Relationenstrukturen in den Vordergrund gestellt. Vertieft werden die Kenntnisse durch eine Einführung in die Normalformentheorie und ihre Bewertung im Vergleich zum dargestellten leicht zu handhabenden graphischen Entwurfsverfahren.
Der praktische Schwerpunkt zielt auf die Vermittlung von Grundkenntnissen im Umfang mit der Datenbanksprache SQL. Hierbei werden die erzeugten Relationenstrukturen selbst in ein Datenbankschema umgesetzt und verschiedene Anfragen selbst erstellt.

Ziel der Veranstaltung

Der Student soll nach der Veranstaltung in der Lage sein...

- einen Problembereich selbständig in ein konzeptuelles Schema zu überführen
- relationale Datenbanken physisch mit SQL implementieren, befüllen, anfragen und manipulieren zu können
- die Normalformentheorie als alternativen Entwurfsansatz beherrschen und anwenden zu können

Notenbildung

Die Gesamtnote setzt sich hälftig aus den gemittelten Einzelnoten der Praktikumsaufgaben (typischerweise werden sechs (d.h. die zweite bis einschließlich siebte) Übungsaufgaben bewertet) und der Klausurnote zusammen.

Vorlesungsinhalt

1 Motivation und Einführung
1.1 Begriffsbestimmung: Was ist eine *Datenbank*?
1.2 Anforderungen an Datenbanksysteme
1.3 Typen von Datenbankmanagementsystemen
2 Entwurf einer Datenbank
2.1 Graphischer Entwurf des konzeptuellen Schemas mit dem Entity-Relationship Modell
2.2 Ableitung logischer Relationenstrukturen
2.3 Algebraischer Entwurf mit der Normalformentheorie
3 Arbeiten mit einer Datenbank
3.1 Codd'sche Regeln und Eigenschaften relationaler Systeme
3.2 Implementierung des logischen Modells mit SQL-DDL
3.3 Der Anfrageteil von SQL

Download der Beispiele und des Scriptums

- PDF
- PDF (tief verlinkt)
- Beispiele
- Scriptquelle im XML-Format

Empfohlene Literatur

Bücher

- Scriptum zur Vorlesung
- Date: *An Introduction to Database Systems*, 8th ed., Pearson, 2004.
- Elmasri, Navathe: *Grundlagen von Datenbanksystemen*
  Deutsche Übersetzung von: *Fundamentals of Database Systems*
- Groff, Weinberg: *SQL -- The Complete Reference*, 2nd ed., McGraw-Hill, Osborne, 2002.
- Heuer, Saake: *Datenbanken: Konzepte und Sprachen*
- Pernul, Unland: *Datenbanken im Unternehmen*
- Reese, Yager, King: *MySQL: Einsatz und Programmierung*
- Türker: *SQL:1999 & SQL:2003*, dpunkt.verlag, 2003.
- Vossen: *Datenmodelle, Datenbanksprachen und Datenbankmanagementsysteme*
- Jeckle, Rupp, Hahn, Zengler, Queins: UML 2 Glasklar, Hanser, 2003.

Online Quellen

MySQL und SQL

- Teil 1: Grundprinzipien, Tabellenerstellung, Einfügen von Daten und einfache Anfragen
- Teil 2: Mächtigere Anfragen, Aktualisieren und Löschen von Daten
- Teil 3: Ausgabesortierung, Verbünde und Datenaggregation
- Subqueries in MySQL, Part 1
- SQL-Functions (Kapitel aus K. Kline, D. Kline: *SQL in a Nutshell*)
- SQL Under The Covers (Artikel über SQL-Verarbeitung in DB2)
- Hierarchical Queries

Mailingliste

Steht im Wintra-System zur Verfügung.

Service provided by Mario Jeckle

Generated: 2004-06-08T12:52:01+01:00

Feedback SiteMap
This page's original location: http://www.jeckle.de/vorlesung/datenbanken/index.html
RDF description for this page

Die Stiftung für liberale Politik
Friedrich Naumann Stiftung

Willkommen auf der Homepage des

Arbeitskreis Bildung

Ein Zusammenschluß der Stipendiaten der Friedrich Naumann Stiftung

Arbeitskreis
Friedrich Naumann Stiftung
BILDUNG
Theoretisches
Workshop
Unsere Seminare
Ferienakademien

Mitglied werden
Mitgliederbereich
Das Team

Aktuell

Daten - Information - Wissen - Bildung (Sebastian Dreßler)

Einleitung

Wir sprechen gelegentlich vom Informationszeitalter oder davon, in einer Informationsgesellschaft zu leben, ebenso hört man von einer Wissensgesellschaft oder der Lernenden Gesellschaft. Von einer Bildungsgesellschaft ist hingegen nicht die Rede, wohl aber von deren Bildungspolitik. Das Verständnis dieser Begriffe und das Verhältnis unter einander scheint kompliziert zu sein. In der Tat erweisen sich die drei Begriffe in der oben angeführten Aufzählung zunächst als plausibel, bei einer näheren Betrachtung jedoch als unscharf. Mit dem folgende Referat sollen die Begriffe in ihrer Bedeutung klarer dargestellt werden.

Daten

"Daten sind Strukturen physikalischer Wirkungen, die zur Ableitung von Informationen dienen oder diese Informationen darstellen." (Mackeprang 1987)

Unter "Struktur" versteht man in diesem Zusammenhang die Anordnung in Zeit und Raum. Üblicherweise wird die "Struktur" mathematisch durch Funktionen beschrieben. Sie kann aber auch aus Zeichen bestehen. Damit geht diese Definition etwas weiter als die DIN Norm 44300:

"Daten sind Zeichen oder kontinuierliche Funktionen..." (DIN 44300).

Diese Definition ist etwas ungenau, weil "Zeichen" und "Funktionen" damit gleichgesetzt werden, was die Zeichen bzw. Funktionen beschreiben. In Wirklichkeit sind Zeichen und Funktionen nur die Beschreibungen von Strukturen.

Aus der Definition folgt, dass Strukturen physikalischer Wirkungen Daten sein können, aber nicht sein müssen. Daten können durchaus ihre Eigenschaft Daten zu sein verlieren, in dem sie nicht zur Ableitung von Informationen verwendet werden. Ob also nun Strukturen der physischen Welt Daten sind oder nicht, hängt in jedem Fall von der Tätigkeit des Empfängers ab.

Damit ist man nun bei dem Ursprung des Wortes "Daten" ("das Gegebene" und/ oder "gegebenes") angelangt. Daten exstieren als solche also nur bei einem Empfänger.

Weiter folgt aus der Definition, dass was im täglichen unreflektierten Sprachgebrauch im Zusammenhang mit Sachsystemen "Information" genannt wird, in Wirklichkeit Daten sind.

Hierzu ein Beispiel bzw. ein kleiner Versuch, die Übertragung einer sehr einfachen Information.

Hierzu ein Beispiel. Man stelle sich folgende Situation vor. Eine Person ist in Not geraten und versucht über Klopfzeichen um Hilfe zu rufendem. Die Person klopfe drei mal kurz, dann drei mal lang und wiederum drei mal kurz gegen einen Gegenstand. Der hilferufenden Person, dem Sender, ist klar, was das bedeutet. Trifft das aber auch auf alle möglichen Zuhörer zu. Wahrscheinlich nicht ganz. Die übermittelten Klopfzeichen werden nur von der Teilmenge aller potentiellen Adressaten verstanden, nämlich die, welche das Morsealphabet kennen und somit wissen, dass drei mal lang den Buchstaben S und drei mal kurz den Buchstaben O bedeutet und damit die Buchstabenfolge SOS geklopft wurde. Ebenfalls nur eine Teilmenge der Zuhören werden wissen, dass diese Zeichenfolge das internationale Seenotsignal, save our souls, bedeutet. An dieser Stelle sei darauf hingewiesen, dass es hier korrekterweise Seenotsignal heißt und nicht etwa Seenotinformation. Eine Information wird es erst durch die Verknüpfung der erlebten Reize mit dem bei dem Zuhörer bereits vorhandenen

Wissen. Information wird also nicht transportiert, sondern bei dem entsprechenden Adressaten wissensgestützt konstruiert.

Es werden also nur Daten übertragen, die man dann als Nachrichten bezeichnen kann. Ein Signal ist dann eine physikalische Größe, deren Struktur (Werteverlauf) Daten sind.

Information

Der Ursprung des Wortes liegt im lateinischen Wort "Informatio". Es ist seinerseits die Substantivform des Verbs "informare" und besteht aus den Derivaten "in" und "forma". Die Vorsilbe "in" kann mit "ein-" oder "hinein-" übersetzt werden. Dagegen ist die Übersetzung des Wortes "forma" durch "Form" nicht so einfach, da dieser Begriff nicht nur auf gegenständlicher (Anordnung von Materie im Raum), sondern auch auf abstrakter Ebene ("gut in Fotm sein") verwendet werden kann. Im klassischen Latein hatte das Verb drei mögliche Bedeutungen:

1. Einen Gegenstand äußerlich und konkret in eine bestimmte Form bringen.
2. Eine Vorstellung von etwas haben, im Sinne sich etwas Abstraktes vorstellbar zu machen.
3. Eine pädagogische Bedeutung im Sinne einer Tätigkeit die ein Lehrender mit einem Lernenden verrichtet. ("einprägen", "bilden", "den letzten Schliff geben")

Betrachtet man nun das Substantiv "informatio" mit der Endung "-tio" ("-ung"), so lassen zwei Interpretationen ableiten. Das Substantiv bezeichnet entweder einen Prozess oder das Ergebnis eines Prozesses. Somit kann man mit den drei Bedeutungen des Verbs sechs verschiedene Verständnisse des Substantivs beschreiben, die in der folgenden Tabelle in einer Matrix zusammengefaßt sind.

Verwendungs-Zusammenhang	Prozess	Ergebnis des Prozesses
konkret	GestaltungFormgebung	FormGestalt
abstrakt	Sich ein Bild machen	Die Vorstellung,Idee
pädagogisch	Unterweisung,Unterricht	Belehrung Wissen,Kenntnis

Bis Heute hat sich im Wesentlichen nur das Verständnis von Wissen und Kenntnis erhalten. Man spricht umgangssprachlich von "Information" im Sinne von Wissen oder Kenntnis. In diesem Sinne spielt es auch eine große Rolle in der Nachrichtentechnik, in der Kybernetik (griech.: Steuermannskunst, Wissenschaft von dynamischen sich selbst regulierenden Systemen) und in der Informationstheorie.

Mit der "Mathematical Theory of Communication", die von Shannon 1949 postuliert wurde, bricht in der Diskussion um den "Informationsbegriff" eine neuer Ära an. Nach Shannons Veröffentlichungen begann in fast allen Wissenschaftsdisziplinen eine Entwicklung, bei der gewisse Begriffe und Regel der Kybernetik und der Kommunikationstheorie "euphorisch" auf andere Gebiete übertragen wurden, die zumindest Shannon mit seiner Arbeit nicht im Auge hatte. Es ereignete sich häufig, dass Begriffe aus ihrem spezifischen Zusammenhang gerissen und in Zusammenhänge gestellt wurden, in der sie dann meist keine erklärenden Funktionen übernehmen konnten.

Die "stürmische" Entwicklung der Verwendungszusammenhänge für das Wort "Information" hat zum einen zu einer semantischen Aufspaltung des Informationsbegriffs geführt. Dies äußert sich heute darin, dass es viele verschiede Informationsbegriffe gibt. Diese Informationsbegriffe sind in ihren jeweiligen spezifischen Verwendungszusammenhängen durchaus wohl definiert und daher auch zweckmäßig, jedoch gibt es aufgrund dieser Entwicklung den Informationsbegriff nicht.

Zum andern hat diese Entwicklung dazu geführt, dass scheinbar jeder weiß, was "Information" ist, in Wirklichkeit aber eine große Unkenntnis herrscht. Mit der Entwicklung der EDV hat das nun zu einem gesellschaftlichen und politischen Problem geführt. Es wird z. B. postuliert, dass man sich im "Informationszeitalter" befinde, oder in den Schulen die "informationstechnische Bildung" vermittelt werden soll. Aber was nun diese "Information" sein soll, darüber weiß und liest man fast nichts.

Nun lassen sich aber aus der Vielzahl von unterschiedlichen Definitionen bzw. Begriffsbestimmungen 16 verschieden Typen identifizieren und in 7 Hauptgruppen einordnen (Mackeprang 1987).

1. Prozess-Ansatz
2. Nachrichten-Ansatz
3. Struktur-Ansatz
4. Wirkungs-Ansatz
5. Wissens-Ansatz
6. Bedeutungs-Ansatz
7. Steuerungs-Ansatz

Hierbei sei angemerckt, dass Mackeprang nur die Literatur aus den Ingenieurwissenschaften und deren Umfeld berücksichtigte.

Aus diesen sieben Hauptgruppen leitete Mackeprang sieben Informationsbegriffe ab, die er dann in einer Definition zu vereinen versucht.

Information:
Information ist der Prozess oder das Ergebnis des Prozesses, bei dem ein sich selbst organisiertes offenes System seine Struktur aufgrund gegebener Syntax ändert.

Ein System ist eine Menge von Elementen, die bestimmte Attribute und Funktionen haben und die untereinander durch Relationen in bestimmten Zusammenhängen stehen (z. B. Ereignisse). Die Elemente könne dabei ihrerseits Systeme sein.

Ein Prozess wird hier als Folge einzelner Ereignisse verstanden, wobei die Ereignisse Elemente des Systems sind.

"Sich selbst organisierend" ist ein System, wenn es seine Struktur aufgrund eines Erkenntnisprozesses ändern kann.

"Offen" ist ein System, wenn es für Menschen möglich ist, auf Metaebene Einblick in einzelne Stufen und in die Art und Weise des Erkenntnisprozesses zu gewinnen. So sind für Menschen nur Menschen offenen Systeme und für Tiere nur Tiere, wenn sie sich untereinander informieren können.

"Gegebener Syntax" sind die wahrgenommenen Strukturen physikalischer Wirkungen auf das System, welche die unterste Stufe des Erkenntnisprozesses bilden. (siehe Daten)

Wissen

Der Informationsverarbeitungsansatz in der (kognitiven) Psychologie geht von einer Netzstruktur des Wissen aus (Hoffmann 1994). Man geht weiter davon aus, dass die Verarbeitungsprozesse in drei Ebenen mit jeweils charakteristischen Überdauerungszeiten und einer Vielzahl von Wechselwirkungen zwischen den Ebenen ablaufen:

· Sensorische Speicher ca. eine halbe Sekunde
· Kurzzeitgedächtnis bis zu 15 Sekunden
· Langzeitgedächtnis nahezu unbegrenzt

Die periphere Verarbeitung sensorischer Reize wird als eine eigenständige funktionale Einheit verstanden. Es wird davon ausgegangen, dass die zu einem bestimmten Zeitpunkt eintreffenden Reize einer parallelen Verarbeitung unterworfen werden. Das Resultat der Verarbeitung ist die Aktivierung von Strukturen im Zentalnervensystem. Die externen Reize werden so auf interne Informationseinheiten abgebildet.

Die im Zentralnervensystem aktivierten Strukturen (Begriffe) repräsentieren erfahrungsabhängig akkumulierte Informationen über Ausschnitte der Umwelt, über uns selbst und über unsere Möglichkeiten, in der Umwelt zu handeln. Zwischen den Begriffen existieren lernabhängig gebildete Verbindungen, die Aktivierungen zwischen den Begriffen vermitteln können. Von einem durch Reizeinflüsse aktivierten Begriff ausgehend, können so mit ihm verbundenen Begriffe ebenfalls aktiviert werden, ohne dass weitere Reizwirkungen dazu notwendig sind. Die Begriffe und die zwischen ihnen existierenden Verbindungen bilden den Inhalt des Langzeitgedächtnisses, sie repräsentieren unser Wissen.

Nach dieser Auffassung beginnt jede Reizverarbeitung mit der Aktivierung von Wissensstrukturen. Aus den Reizen werden dann sinnvoll interpretierbare Informationen. Somit erfolgt die Verarbeitung eines Reizes stets unter der Berücksichtigung des über diesen Reiz im Gedächtnis gespeicherten Wissens. Wie oben bereits am "SOS-Versuch" illustriert erfolgt die menschliche Informationsverarbeitung immer wissensgestützt.

Bildung

Bildung kann beschrieben werden als die Fähigkeit, mit Kultur umzugehen und die Fähigkeit, Werthaltungen wie Menschlichkeit, Verantwortung, Solidarität etc. im Leben als Individuum und als Mitglied der Gesellschaft umzusetzen (Dahncke 2000). Wichtig für die Einordnung des Bildungsbegriffes ist also der Bezug zur Gemeinschaft/Gesellschaft. Zum einen vollzieht sich Bildung mit dem Individuum und dient dann auch diesem. Zum anderen dient die Bildung der Gemeinschaft, der Gesellschaft und ist unentbehrlich für unser Zusammenleben.

Diese Kurzbeschreibung von Bildung möchte ich so im Raum stehen lassen, aber doch noch ein wenig die Situation um den "Bildungsbegriff" beschreiben.

Ropohl (1979) hat ein wissenschaftstheoretisches Werk über Technik geschrieben, das mit dem bemerkenswerten Satz "Jeder weiß, was Technik ist, und dennoch weiß es niemand" beginnt. Dieser Satz läßt sich auf eine Reihe von anderen Begriffen, insbesondere auf den Bildungsbegriff, übertragen. Also: Jeder

weiß oder glaubt zu wissen was Bildung ist, und dennoch kann niemand dieses Wissen so formulieren, dass eine von allen akzeptierte Definition des Bildungsbegriffs entsteht. Die Situation bei der Definition des Bildungsbegriffs scheint also noch diffiziler zu sein als beim Informationsbegriff. Bei der Behandlung dieses Definitionsproblems fällt zunächst auf, dass das Wort Bildung sehr häufig mit zusätzlichen Adjektiven versehen wird. So kennt man z. B. materielle Bildung, kategoriale Bildung, formale Bildung. Weiter taucht das Wort Bildung zusammengesetzt mit andern Nomen auf, wie z. B. Allgemeinbildung, Schulbildung, Menschenbildung, Computerbildung, Lehrerbildung. Man ist mit einer derartigen Aufzählung im Verständnis zwar einen kleinen Schritt weitergekommen, jedoch von einem wirklichen Bildungsverständnis oder gar von einer Bildungsdefinition weit entfernt. Das Problem einer direkten Definition findet man bei vielen Begriffen. Man hilft sich in solchen Fällen zumeist dadurch, dass man den angesprochenen Begriff in einer Reihe von Aussagen, Geschichten, Episoden, Prinzipien und Gesetzen erscheinen lässt. Auf diese Art und Weise erwächst einem allmählich eine Vorstellung von dem gemeinten Begriff. Im amerikanischen nennt man so gefasste Begriffe gelegentlich "law-cluster-word" ("Gesetzeshaufenwort"). Um das "law-cluster" des Begriffs Bildung wiedergeben zu können, sind also eine große Fülle von Aussagen, Geschichten und Zitaten notwendig, auf die ich im weiteren nicht eingehen möchte.

An dieser Stelle möchte ich mit der Beschreibung: Bildung ist der Prozess sich und sein Menschenbild zu bilden, meine Ausführungen zu dem Begriff "Bildung" schließen.

Es sei hier aber auf einige interessante Arbeiten zu diesem Thema von Dahncke (2000), Schwanitz (1999), Klemm, Rolff und Tillmann (1985), Geissler (1986), Marquard (1985) und Hentig (1997) hingewiesen.

Literatur:

Dahncke, Helmut
"Bildungsverständnis in einer hochtechnisierten Welt", hrsg. in "Zur Didaktik der Physik und Chemie", Brechel, R., Leuchtturm-Verlag, 2000

Hentig, H. v.
Bildung - Ein Essay, Darmstadt, 1997

Hoffmann, Joachim
"Kognitive Psychologie", Handwörterbuch der Psychologie, hrsg. von Asanger R. und Wenninger G., Psychologie Verlags Union, 1988

Klemm, K., Rolff, H.-G., Tillmann, K.-J., Bildung für das Jahr 2000, Hamburg, 1985

Mackeprang, Hartwig
"Zum Informationsbegriff der Allgemeinen Technologie", Diss., Berlin, 1987

Marquard, O.
"Über die Unvermeidlichkeit der Geisteswissenschaften", hrsg. in WRK: "Anspruch und Herausforderung der Geisteswissenschaften", Bonn, 1985

Rophol, Günter
"Eine Systemtheorie der Technik", Hanser Verlag, 1979

Schwanitz, D.
Bildung - Alles, was man wissen muss", Frankfurt a. M., 1999

Shannon, Claude E.
"The Mathematical Theory of Communication" Illinois: Univ. of Illinois Press, 1949

FoeBuD e.V. · Marktstraße 18 · D-33602 Bielefeld
http(s)://www.foebud.org · foebud@bionic.zerberus.de

28. Glossar

Algorithmus
Eine Rechenvorschrift. (Informatikerinnen mögen uns die stark verkürzte Erklärung verzeihen.)

Angriff
Der Versuch, verschlüsselte oder sonstige nicht offen zugängliche Daten zu erhalten, d.h. sie ohne Berechtigung zu lesen oder zu kopieren.

ASCII
"American Standard Code for Information Interchange", ein 7-Bit-Zeichensatz, der heutzutage von praktisch allen Computern verstanden wird. Leider enthält ASCII zwar Steuersequenzen wie z.B. Zeilenende und Wagenrücklauf (deren Interpretation von Betriebssystem zu Betriebssystem unterschiedlich ist) und "Klingelton", aber keine Umlaute und ähnliche Sonderzeichen. Das führt immer noch dazu, daß der Austausch von Texten mit Sonderzeichen zwischen verschiedenen Rechnersystemen mit Problemen behaftet ist.

asymmetrische Verschlüsselung
Vgl. "Verschlüsselung".

Chiffrat
Verschlüsselter Klartext.

CRC
CRC steht für "cyclic redundancy check". CRC-Summen sind Prüfsummen, die schnell zu berechnen sind und sehr zuverlässig auf zufällige Änderungen in den Eingabedaten reagieren. Sie werden beispielsweise benutzt, um bei der Datenübertragung mit Modems Übertragunsgfehler zu entdecken. Für kryptographische Zwecke sind CRC-Prüfsummen leider nicht zu gebrauchen, da es sehr einfach ist, zu einer gegebenen Prüfsumme Nachrichten zu konstruieren.

Daten
Alles, was ein Computer verarbeiten kann, sind Daten. Daten sind (meist formalisierte) Beschreibungen einer (realen oder eingebildeten) Realität. Beispiele sind dieser Text, eine Bilddatei oder die Angaben einer Statistik.

digitale Unterschrift

Eine Abart der asymmetrischen Verschlüsselung, bei der das Chiffrat mit einem geheimen Schlüssel erzeugt wird, so daß mit Hilfe des öffentlichen Schlüssels überprüft werden kann, daß die Nachricht tatsächlich von der autorisierten Person verschlüsselt und somit unterschrieben wurde. Wird i.A. in Verbindung mit digitalen Fingerabdrücken verwendet.

Fingerabdruck

Ein mit Hilfe eines festgelegten Verfahrens aus einem beliebig langen Klartext berechneter Wert einer festen Länge. Bei den in der Kryptographie üblichen Verfahren (wie in PGP eingesetzt) läßt sich aus dem Fingerabdruck der Klartext nicht bestimmen und es ist auch nicht möglich, zwei Klartexte mit demselben Fingerabdruck zu berechnen. Näheres finden Sie in Abschnitt 4.6 auf Seite .

Information

"Information is a difference that makes a difference." (Gregory Bateson) - "Information ist ein Unterschied, der eine Bedeutung hat." oder auch "Information ist ein Unterschied, der einen Unterschied macht." Nach dieser Definition werden Daten dadurch zur Information, daß sie beim Empfänger eine Entscheidung beeinflussen. Das ist konsistent mit der Definition nach Shannon, wonach der Informationsgehalt eines Datums (Datum ist die Einzahl von Daten) sich nach der Anzahl der Ja/Nein-Fragen bestimmen läßt, die der Empfänger erst nach Kenntnis dieses Datums beantworten kann.

Eine andere Definition bezieht sich auf den Bedeutungsinhalt der betrachteten Daten. Hiernach ist Information streng an ein Bewußtsein gekoppelt; Computer können lediglich Daten verarbeiten und ihre menschlichen Benutzer bei der Betrachtung der Daten unterstützen, um Informationen zu gewinnen. Sie können aber selbst keine Informationen verarbeiten.

Auf jeden Fall ist Information immer etwas empfängerabhängiges. Den absoluten Informationsgehalt einer Nachricht bestimmen zu wollen, ist etwa genauso sinnvoll wie allgemein angeben zu wollen, wie lange es dauert, ein bestimmtes Buch zu lesen.

geheimer Schlüssel

Bei einem asymmetrischen Verfahren derjenige Teil des Schlüsselpaares, der für die Entschlüsselung bzw. die Signatur einer Nachricht notwendig ist.

Klartext

Mit Klartext ist im Zusammenhang mit Verschlüsselung von Computerdaten nicht nur Text im Sinne von "für den Menschen lesbar" gemeint, sondern alle nicht verschlüsselten Daten. PGP verschlüsselt im Allgemeinen einzelne Dateien, so daß "Klartext" in diesem Handbuch und allgemein bei der Benutzung PGPs als "unverschlüsselte Datei" gelesen werden kann.

Kryptanalyse

Methoden und Verfahren, um chiffrierte Daten ohne vorherige Kenntnis des Schlüssels zu entschlüsseln.

Kryptologie

Die Lehre von der Verschlüsselung.

Kryptographie

Die praktische Anwendung der Kryptologie.

Mantra

In diesem Buch: Der geheime Text, mit dem Ihr privater Schlüssel geschützt ist und den Sie für die Benutzung Ihres Schlüssels eingeben müssen - gewissermaßen die Langversion eines Paßwortes. Näheres zum Mantra und zur Wahl eines guten Mantras steht auf den Seiten ❑, ❑, ❑ und ❑.

Nachricht

Mit "Nachricht" ist im Rahmen dieses Handbuches eine einzelne - verschlüsselte oder unverschlüsselte - Datei gemeint. (Teilweise sehen Sie keine Datei im üblichen Sinne, aber auch eine verschlüsselte E-Mail stellt im Wesentlichen eine Datei dar.)

öffentlicher Schlüssel

Bei einem asymmetrischen Verfahren das Gegenstück zum geheimen Schlüssel, also der Teil des Schlüsselpaares, der für die Verschlüsselung einer Nachricht bzw. die Überprüfung einer Unterschrift notwendig ist.

privater Schlüssel

Siehe "geheimer Schlüssel".

Prüfsumme

Siehe "Fingerabdruck".

RfC

"Request for Comment", wörtlich "Bitte um Kommentare". Dieser Name ist leicht irreführend: Die RfCs sind die Sammlung der Definitionen und technischen Spezifikationen, nach denen u. a. UseNet und Internet ablaufen. Sie finden die RfCs gesammelt unter `www.ietf.org/rfc.html`

Schlüssel

Ein Verschlüsselungsverfahren hat immer zwei Eingaben: Zum Einen den Klartext (bzw. im Fall der Entschlüsselung das Chiffrat), zum anderen den Schlüssel. Der Schlüssel dient als Geheimnis, ohne das aus dem Chiffrat der Klartext nicht gewonnen werden kann.

Verschlüsselung

Verschlüsselung hat das Ziel, einen Klartext unter Verwendung eines Schlüssels in ein Chiffrat umzuwandeln, so daß es bei Kenntnis des passenden Schlüssels leicht ist, aus dem Chiffrat wieder den Klartext zu gewinnen, es aber nicht möglich (im Sinne von durchführbar) ist, dies ohne den Schlüssel zu tun. Bei den meisten Verfahren ist es darüber hinaus wichtig, daß es auch nicht möglich ist, aus der Kenntnis von Klartext und Chiffrat den Schlüssel zu berechnen. Die für die Ent- und Verschlüsselung verwendeten Schlüssel können identisch sein, dann spricht man von einer *symmetrischen Verschlüsselung,* oder sie können verschieden sein, in diesem Fall liegt eine *asymmetrische Verschlüsselung* vor. Näheres finden Sie in Abschnitt HREF="node6.html#Funktionsweise">2.5.

Next Up Previous Contents Index

Next: Index **Up:** Literaturverzeichnis **Previous:** Literaturverzeichnis

Copyright-Vermerke

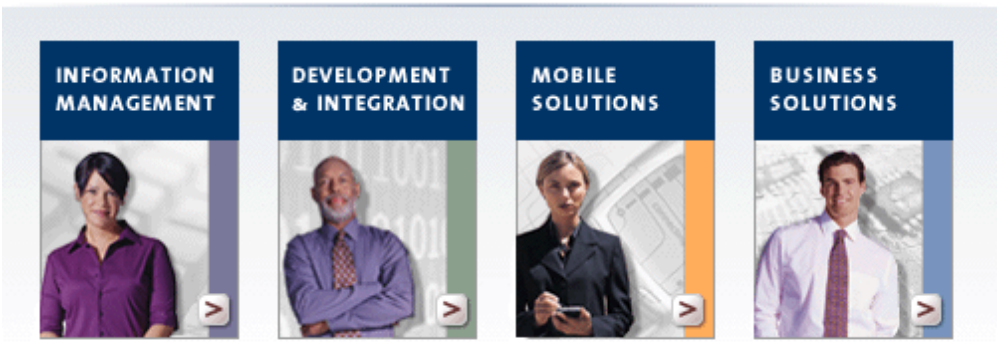
Buch bestellen

Christopher Creutzig

SYBASE

THE ENTERPRISE. UNWIRED. Your customers are here, your partners are there, and your sales people need to be just about everywhere. You need to deliver the right data to the right place at the right time. Quickly. Securely. Easily. You need to 'unwire' your enterprise with Sybase. We deliver enterprise and mobile software solutions for information management, development, and integration. More>

Unwired Campaigns



News

Sybase CEO: Unwired Enterprise Requires Aggressive Apps

'Unwired' Sybase Seeks Better Business Through Channels

Sprint Unveils Plans for Mobile Banking Services Solution

More News...

Events

ISUG Techcast - Easy XML with PowerBuilder Document Object Model - Tuesday, June 8 and Thursday, June 10

Sybase Techwave 2004

Mobilize Your Enterprise for Success

More Events...

Information For

Developers (SDN)
Powerful tools for developers.

Partners
Find a partner. Become a partner. Partner resources.

Press
Sybase news and information.

Industry Solutions

Healthcare
Process improvement, regulatory and mobility solutions.

Financial
Wall Street runs on Sybase.

Government
Delivering critical information securely.

Success Stories

Alvion Technologies — Marketing List Data Warehouse

Statistics Canada — Replication On Demand

MAXIMUS — Mobile Maintenance and Inventory

More Success Stories...

Advantage CA-IDMS

NETWORK AND RELATIONAL DBMS FOR ULTRA-HIGH PERFORMANCE

The Advantage CA-IDMS family provides the leading database management system for the most demanding mainframe environments. Its product mission has always been to serve as the industry-standard, high performance mainframe database management solution for mission critical systems. Advantage CA-IDMS systems that have been developed, debugged, tuned and enhanced over the last three decades are reliably running businesses in over 2,000 sites around the world. More...

have a rep call me

product news

CA's Advantage CA-IDMS Database r16 for z/OS Optimizes Performance, Ease of Use, and Flexibility

Three of World's Top Ten OLTP Databases Run on CA Advantage Technology According to New Winter Corporation Study

sign up for E-News

email this page

products

Advantage CA-IDMS/DB Database
High Performance Mainframe Database Management

Advantage CA-ADS for CA-IDMS
Cooperative Processing For Advantage CA-ADS

information center

> Data Sheets
> White Papers

featured topic

related solutions

Portal and Business Intelligence
Managing Business Intelligence

Application Life Cycle Management
Managing eBusiness Development

featured products

Advantage CA-IDMS/DB Database

Advantage CA-ADS for CA-IDMS

spotlight

Trial & Evaluation Software

Partner Programs

events

Free Seminar
Business Savvy Storage Decisions

LinuxWorld
August 2 – 5

FlexSelect Licensing

Our Software Sold Your Way

headlines

CA Announces 2004 Channel Partner Award Winners

Kumar to Leave Computer Associates

CA Software and Services Help Secure Microsoft Tech-Ed Infrastructure

BrightStor Storage Resource Manager Wins Network Computing Award

CA Reports Q4, Full Fiscal 2004 Results and Provides Guidance for 2005

IMS Family

DB2 Information Management Software

IBM's premier transactional and hierarchical database management system for critical on-line operational and e-business applications and data, enabling Information Integration, Management, and Scalability.

Select a Product Go

Products

IMS
IBM's premier transactional and hierarchical database management system for critical on-line operational and e-business applications and data.

Overview of IMS
Overview presentation of IBM's premier transaction and hierarchical database management system.

IMS Connect
A must-have tool for any IMS shop embarking on e-business, IMS Connect improves TCP/IP access and enables easier access to IMS applications and data from the internet.

IMS Connector for Java
IMS Connector for Java, a component of IMS Connect, allows you to rapidly create and run Java applications to access IMS transactions over the internet through IMS Connect.

IMS Java
IMS Java application support is a function of IMS that enables a programmer with minimal IMS knowledge to write Java application programs that access IMS databases.

SOAP for IMS
The IMS SOAP Gateway is an XML-based connectivity solution that enables existing or new IMS applications to communicate outside of the IMS environment using SOAP.

Tools and components

IBM DB2 and IMS Tools
IBM DB2 and IMS Tools are specifically designed to enhance the performance of IMS and DB2 in an affordable and easy-to-use manner.

IMS Control Center
Using the IMS Control Center with IMS Version 8, you can manage your IMS systems using a graphical user interface from a Windows or UNIX workstation.

Events

IMS Technical Seminar Series
June 15-17, 2004
Detroit, MI; Columbus, OH; Pittsburgh, PA

IMS Technical Conference 2004
June 16-17, 2004
Melbourne, Australia

IMS Fundamentals
June 16-25, 2004
IBM Web Conference

IMS Technical Seminar Series
June 22-24, 2004
Glendale, CA; Costa Mesa, CA; San Francisco, CA

IMS DB Application Programming
July 7-9, 2004
San Francisco, CA

IMS DC Application Programming
July 12-13, 2004
San Francisco, CA

IMS Technical Seminar Series
July 13-15, 2004
Richmond, VA; Washington, DC; Philadelphia, PA

IMS Parallel Sysplex Workshop
August 23-26, 2004
Dallas, TX

IMS Technical Conference
September 27-30, 2004
Orlando, FL

Highlights

Announcement Letters

Presentations/papers

Communities

- Receive Recent IMS news
- IMS Newsletter
- IMS User Groups

Skills

- Documentation (V7 refreshed in April 2004)
- Learn about the IMS Mastery Certification Program
- Want to take an IMS Debug class? Tell us where you'd like it!

DB2 Product Family

DB2 Information Management Software

IBM's DB2 database software is full-featured, robust, scalable and easy to use. As the market share leader, DB2 provides the foundation of information on demand on Linux, UNIX and Windows platforms. DB2 UDB is specially designed and priced to meet your business needs whether large or small.

Select a Product Go

Products

DB2 Universal Database V8.1
IBM's relational database management system for AIX, Linux, HP-UX, Sun, and Windows.

DB2 Universal Database for z/OS and OS/390
Use DB2 on the mainframe to run powerful enterprise applications, and make e-commerce a reality.

DB2 Connect
Make your company's host data directly available to your Personal Computer and LAN-based workstations.

DB2 Everyplace
DB2 relational database and enterprise synchronization architecture for mobile and embedded devices.

DB2 UDB Data Warehouse Editions
Deliver rich BI functionality inside the database.

Tools and components

DB2 and IMS Tools
Data administration, performance, application management, recovery and replication tools.

DB2 Cube Views
Accelerate OLAP queries by using more efficient DB2 materialized query tables.

DB2 Extenders
Manipulate multiple data types with powerful built-in functions

DB2 Intelligent Miner
Tools for data analysis, modeling, scoring, and visualization

DB2 related tools and applications
IBM tools and utilities for developing applications using DB2.

Solutions

Information on Demand
Web-ready solutions.

Industry solutions through Strategic Alliances

Highlights

DB2 UDB Shatters TPC-C Price/Performance Record

E*Trade Financials chooses IBM's DB2 ICE

Oracle, Sybase, MS SQL Server & Informix to DB2 Migration Toolkits

Tuning DB2 databases for Intel Itanium 2-based systems

Communities

Language support for IBM products

developerWorks DB2

DB2 Information Management Resources

Porting to DB2 UDB? Technical Resources

Skills

Training & Certification

IBM Scholars Program

Skills Plus Network

Informix product family

DB2 Information Management Software

IBM Informix® software delivers superior application performance for transaction intensive environments. Our products provide flexible solutions that integrate easily with your IT environment -- ensuring high performance and streamlined, easy administration while managing massive amounts of structured and unstructured data. In addition, other IBM software enhancements assure compatibility with Informix products.

Select a Product Go

Products

Informix Dynamic Server (IDS)
Exceptional online transaction processing (OLTP) database for enterprise and workgroup computing.

Informix OnLine Extended Edition
Easy-to-use, embeddable, relational database server for low-to-medium workloads.

Informix Standard Engine (SE)
Embeddable database server designed for developing small- to medium-sized applications that need the power of SQL without any database administration requirements.

Informix C-ISAM™
Informix C-ISAM is a file management technology that uses a library of C-language functions to efficiently manage indexed sequential access method (ISAM) files.

Informix Extended Parallel Server (XPS)
Comprehensive, flexible data-warehousing features enabling integration of business systems.

IBM Red Brick™ Warehouse
Database server designed to meet the specialized requirements for business-critical, high-demand data analysis.

Tools and components

Informix Tools
A comprehensive array of application development products integrated with Informix database servers.

Informix DataBlade™ modules
Extend the database so that complex objects can be stored in a relational database.

Solutions

IDS and IBM software compatibility guide
By combining the IDS online transaction processing (OLTP) database with IBM software, you can build a resilient on demand business operating environment capable of handling peaks in customer usage with no effect on performance. Get details on the IBM software products that have been updated and tested to complement Informix software.

Informix and WebSphere bundles
Extend the responsiveness, focus and resiliency of your software applications by adding capabilities to your existing IT investment. The advantage is a comprehensive, integrated infrastructure for small, medium and large on demand businesses.

Events

Insight on Demand: Join Janet Perna and others for this executive event on the power of integrated information
June 10, 2004
New York

Join us at worldwide Informix Infobahns! Get details and register.

DB2 Information Management Technical Conference. Register now!
September 20-24
Las Vegas, NV

Webcast on demand: Informix and WebSphere: Integrated for e-business

Webcast on demand: IBM software for zSeries

Highlights

New release: IBM Red Brick Warehouse 6.3

Gold Program enhancements and other great offers

Announcing IBM IDS Workgroup Edition Unlimited

Looking for trial software, fixes, patches?

Communities

Small/Medium businesses

developerWorks Informix

IBM Business Partners

User groups

United States

Search

Home | Products & services | Support & downloads | My account

Select country / region

Select one Go

Resources for:

- Home / home office
- Small & medium business
- Large enterprise
- Government
- Education
- Developers
- IBM Business Partners
- Investors
- Journalists

Jobs at IBM

Training

Administrative support

IBM TV ads

THE PGA TOUR IS ON

The game of golf has entered a whole new era thanks to the power of On Demand Business.

ON DEMAND BUSINESS™

Free Port Replicator II offer

Learn more.

Better business

Build it with Service-Oriented Architecture.

Learn how.

Solutions

Solving business problems:
IBM solutions integrate hardware, software and services to meet the challenges of your industry.

Select an industry Go

Services

- Business and IT services
- Business consulting services
- On demand business
- Infrastructure services
- Financing

Find it fast

- Notebook finder
- Desktop finder
- Intel-based server finder
- Ready to buy?
- Special offers

News

- Engineering and technology focus of camp for 1,000 girls
- IBM gets 'On' new advertising campaign
- Five top innovators named IBM Fellows
- China's auto suppliers could face bumpy road

→ More news and newsletters

ON DEMAND BUSINESS™

Mainframes turn 40

→ From trailblazers to "dinosaurs" to business powerhouses.

The world's most popular open source database

“To provide the best possible service requires maintaining a massive database that can meet extreme performance demands, with very high reliability. The MySQL database has easily met the need, and for a lower total cost of ownership”
Craig Murphy, CTO, Sabre Holdings

BUY MYSQL ONLINE TODAY!

Quick orders:

- MySQL Pro License, $495
- Entry Level Technical Support, $1500
- Zend Studio Plus at 15% discount

Visit the Online Shop »

The MySQL database server is the world's most popular open source database. With more than five million active installations, MySQL has quickly become the core of many **high-volume, business-critical applications**.

Customers such as **Yahoo!, Google, Cisco, Sabre Holdings, HP** and **NASA** are realizing significant cost savings by using MySQL's high performance, reliable database management software to power large Web sites, business-critical enterprise applications and packaged software applications.

MYSQL NEWS:

- 2004.05.24 Matanuska Telephone Association Relies on MySQL for its Most Critical Data
- 2004.05.24 MySQL AB Awarded "IT Company of the Year" by Sweden's Leading Business Magazine
- 2004.05.20 MySQL Wins 2004 SIIA Codie Award for Best Open Source Product
- 2004.05.19 MySQL Professional Certification is now available.
- 2004.05.19 MySQL Certification Study Guide is now available.
- 2004.05.13 MySQL and REALTECH partner for migration of SAP applications to MaxDB

More News » Downloads » Feedback »

EVALUATE
Database Home
Product Editions
Solutions
Real Application Clusters
Manageability
Performance
Product Literature
Demos
Compare
Internet Seminars
Customer Successes
Expert Opinions
Partners

IMPLEMENT
Tutorial
Documentation
Consulting
On Demand
Migration

USE
Training & Education
Support

Printer View

Oracle Database

10*g* Grids Are Everywhere: Easier to Manage at a Lower Cost

Oracle Database 10*g* is the industry's first designed for grid computing, reducing the cost of IT by automating management and clustering servers to dynamically shift resources between applications. Independent studies show Oracle Database 10*g* delivers a higher quality of service at a lower cost of management:

- Summit Strategies: Oracle Grid Control Reduces Manageability Costs, Complexity
- The Edison Group: Oracle is 30% more cost effective to manage vs. Microsoft SQL Server

Today's new, low pricing makes Oracle Database 10*g* the best choice for large enterprises and small and medium-sized businesses alike. Only Oracle offers the benefits of clustering--high availability and scale out on-demand--with Oracle Real Application Clusters.

Customers Simplify Business With Oracle 10*g* (1 min.)

News & Events

- New, low pricing: Oracle Standard Edition One up to two processors for US$745
- Easy as Dell: Oracle Standard Edition One now available directly from Dell
- Oracle Real Application Clusters now available with Oracle Database 10*g* Standard Edition
- Free 2 Day DBA Tutorial: Learn Oracle Database 10*g* administration in just two days

10*g* Grids Are Everywhere

Higher Service Quality:
New Summit Strategies Report on Oracle 10*g* Grid Control

Lower Management Cost:
Oracle 10*g* Easier to Manage than Microsoft SQL Server 2000

Easy as Dell:
Oracle Standard Edition One Now Available Directly from Dell

MORE GRID

The Oracle ISV Forum

Hosted by Oracle PartnerNetwork:
Attend Sessions and Executive Meetings

Register Today:
June 29-30, San Francisco

MORE ORACLE ISV FORUM

Reduce HR Costs

Register Now to Download:
Articles on Solutions to Today's HR Challenges

Listen to Industry Experts:
Hear the HR Transformation Webcast

MORE ORACLE HUMAN RESOURCES

Product Information
How to Buy
Technical Resources
Downloads
Support
Partners
Technologies
SQL Server Community
Windows Server System

How safe is your data?

Be proactive. Protect your SQL Server databases and keep them up to date.

- **Take 10 steps to help secure SQL Server**
- **Learn how Windows XP SP2 affects SQL Server and MSDE**
- **Find more security resources**

Top Stories

Download SQL Server Best Practices Analyzer
Use this tool to scan your SQL Server 2000 systems to verify common best practices have been implemented and ensure better, more manageable installations.

SQL Server Spotlight: Security and SQL Server
Tom Rizzo discusses how the SQL Server team is enhancing security in SQL Server 2005 and what you can do today to help secure your SQL Server 2000 installations and prepare for SQL Server 2005.

Get the New Excel Add-in for SQL Server Analysis Services
Easily access online analytical processing (OLAP) data, perform in-depth analysis, and create flexible, interactive reports using Microsoft Office Excel.

Highlights

Support Policy for SQL Server 2000 and SQL Server 7.0 Extended
Microsoft announces enhanced support life-cycle policy for business and developer products, including SQL Server. Find more details on the Microsoft PressPass site.

Get Started with Reporting Services
Turn business information into a competitive advantage with the report authoring, management, and delivery capabilities of SQL Server 2000 Reporting Services.

Download the Updated SQL Server 2000 Books Online
Get the complete SQL Server 2000 product documentation, updated in January 2004 with new content and revisions.

Choosing an Edition of SQL Server 2000
Find out which edition of SQL Server is right for your organization.

Take a Closer Look

- Simplify the creation and management of business scorecards
- Attend a Reporting Services LoadFest event near you
- Compare SQL Server with other databases
- Is MSDE a better choice for your database needs?

Top Downloads

- SQL Server 2000 SP3a
- Most popular downloads for SQL Server
- SQL Server trial software
- SQL Server Web Data Administrator
- SQL Server 2000 Driver for JDBC SP2

Quick Links

- Latest security bulletin
- SQL Server "Yukon"
- Operations Guide
- Webcasts & chats
- Training & events

Resource Centers

- BI & Data Warehousing
- High Availability
- Scalability & VLDB
- Security
- Small & Medium Business

On the Job

Mary Kay:
Reducing costs with SQL Server 2000 Reporting Services

Information For...

- Business executives (Executive Circle)
- Developers (MSDN)
- IT professionals (TechNet)

Related Sites

- Windows Server 2003
- Microsoft .NET
- MDAC on MSDN
- Books for SQL Server on Microsoft Press

Last Updated: May 25, 2004

English

Microsoft.com Home | Site Map

Search

Warning: You are viewing this page with an unsupported Web browser. This Web site works best with Microsoft Internet Explorer 5.01 or later or Netscape Navigator 6.0 or later. Click here for more information on supported browsers.

Home
Assistance
Training
Templates
Clip Art and Media
Downloads
Office Marketplace
Product Information

Things To Do
Suggest new content
Get our newsletter
Give us feedback

Access

Help

Read about summing in reports

It all adds up. Learn how to place calculated sums in your reports, and in the headers, footers, and groups in your reports. We provide a database with examples that you can download.

Discover Access and Office 2003

- Product information
- Trial version
- Pricing information
- System requirements
- Frequently asked questions

Security warning FAQ
Learn how to deal with Access security messages when you open files.

Join the new Office Communities
Now it's even easier to get answers from other Office users just like you.

Master database design
Use Excel and proven design methods to plan your Access database.

Top tip
To make a text box change size to fit its contents, set its CanGrow or CanShrink property to Yes.
More tips...

BROWSE ACCESS

- 2003 assistance
- 2002 assistance
- 2000 assistance
- Developer Center
- Product support
- MVP connection
- Office Resource Kit
- 3rd-party solutions
- Discussion groups
- Microsoft Press books
- Marketplace

ACCESS HIGHLIGHTS

You talked, we listened

- Learn about using the Office 2003 customer feedback features
- Updated: Types of reports you can create in Access
- Updated: About Microsoft Jet Expression Service sandbox mode
- Updated: Hide the "file may be unsafe" warning
- Updated: Hide the "unsafe expressions are not blocked" warning

Readers' choice: Top picks

- XML for the uninitiated
- Sample databases you can download and adapt
- Using dates and times in Access
- Obtain and deploy the Access 2003 runtime
- Queries III: Create parameter queries

Office Update

Check for Updates

Clip of the Day

Quick links

- See Office demos
- Protect your PC
- Find solutions
- Office quizzes
- Calendars and planners
- Business photos
- Data analysis tools
- Messaging tools
- Contact live support
- Columnists

Hot downloads

- Visio Viewer 2003
- Office 2000 Service Pack 3
- MSN Money Stocks Add-in

Tools for your job

- Finance
- Human resources
- Operations
- Sales
- All tools

Information for

- Education
- Healthcare
- Small business

Requires Office 2003

Grow your skills

- Create dynamic reports in Access using linked tables
- Visit the Office Marketplace for analysis tools that work with Access
- Protect your PC in 3 easy steps

Training

- Access 2003 training roadmap
- Table that data
- Queries I: Get answers with queries

More...

Downloads

- Sample databases you can download and adapt
- Microsoft XML Spreadsheet Add-in for Access
- Snapshot Viewer for Access 2003, 2002, 2000, and 97

More...

Support corner

- Get the Access 2.0 Converter to open Access 2.0 files in Access 2003
- About installing Jet 4.0 Service Pack 8 or later
- Troubleshoot the security messages that appear when you open a file in Access 2003
- I get the message "ActiveX component can't create object"
- I get the message "No Access license on this machine" when I start Access

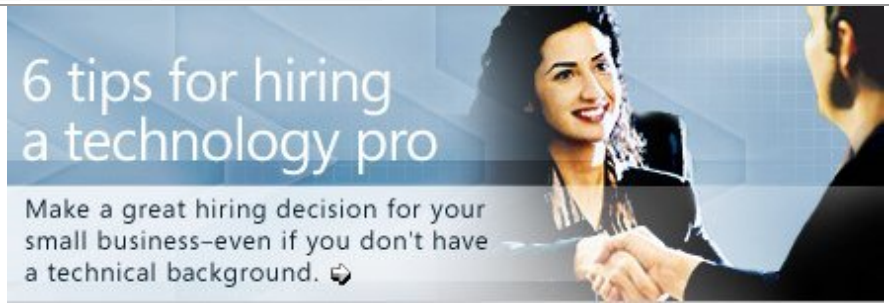
Product Families
Windows
Office
Mobile Devices
Business Solutions
Servers
Developer Tools
Games and Xbox
MSN Services
All Products

Resources
Support
Downloads
Windows Update
Office Update
Learning Tools
Communities
Security

Information For
Home Users
IT Professionals (TechNet)
Developers (MSDN)
Microsoft Partners
Business Professionals
Educators
Journalists

About Microsoft
Corporate Information
Investor Relations
Careers
About this Site

Worldwide
Microsoft Worldwide



home & entertainment

- Find out what your mouse is saying about you
- Let Windows XP back up your data so you don't have to
- Have you done the top 3 things to help protect your PC?

technical resources

- Visual Studio tools for the Microsoft Office System
- Download the SQL Server 2000 Best Practices Analyzer
- Become a Microsoft Certified Desktop Support Technician

business agility

- 18 ways to have fun with Tablet PCs
- Ready for Internet phone service? 5 things to know
- Do enhanced business reporting and data analysis in Excel

Windows

- Internet Explorer: How and why to clear your cache
- How Windows can easily help you fix a mistake
- Choose from 4 editions of Microsoft Windows XP

More Windows ...

Office

- See what programs are in Office 2003 Editions
- 7 mobile services that work with Office
- Access work e-mail messages on any computer with an Internet connection

More Office ...

Windows Server System

- Download Exchange Server 2003 SP1
- What do you want in "Longhorn" Server?
- Free learning plan: Assess your Windows Server System skills

More Windows Server System ...

Visual Studio .NET

- Top 10 MSDN code samples
- Watch online movies to improve your Visual Basic skills
- XML support in SQL Server 2005

Microsoft Business Solutions

- 7 wonders of automated payroll
- Create reports that people really read
- How to figure the total cost of a business solution

MSN

- Use MSN Messenger to keep your life organized
- Download MSN Messenger 6.2
- Get one-click access to e-mail: Download the MSN Toolbar

today's news

- World leaders try out Tablet PCs at G8 Summit

More Microsoft News ...

popular downloads

- Microsoft Office 2000 Service Pack 3
- Internet Explorer 6 Service Pack 1 update
- Windows Media Player 9 Series
- Cumulative security update for Microsoft Outlook Express 6 Service Pack 1

More Downloads ...

popular searches

- Sasser
- Worm
- Movie Maker
- Internet Explorer

More Searches ...

support

- Downloads and updates
- Service packs for Microsoft products
- 3 steps to help ensure your PC is protected

More Visual Studio ...

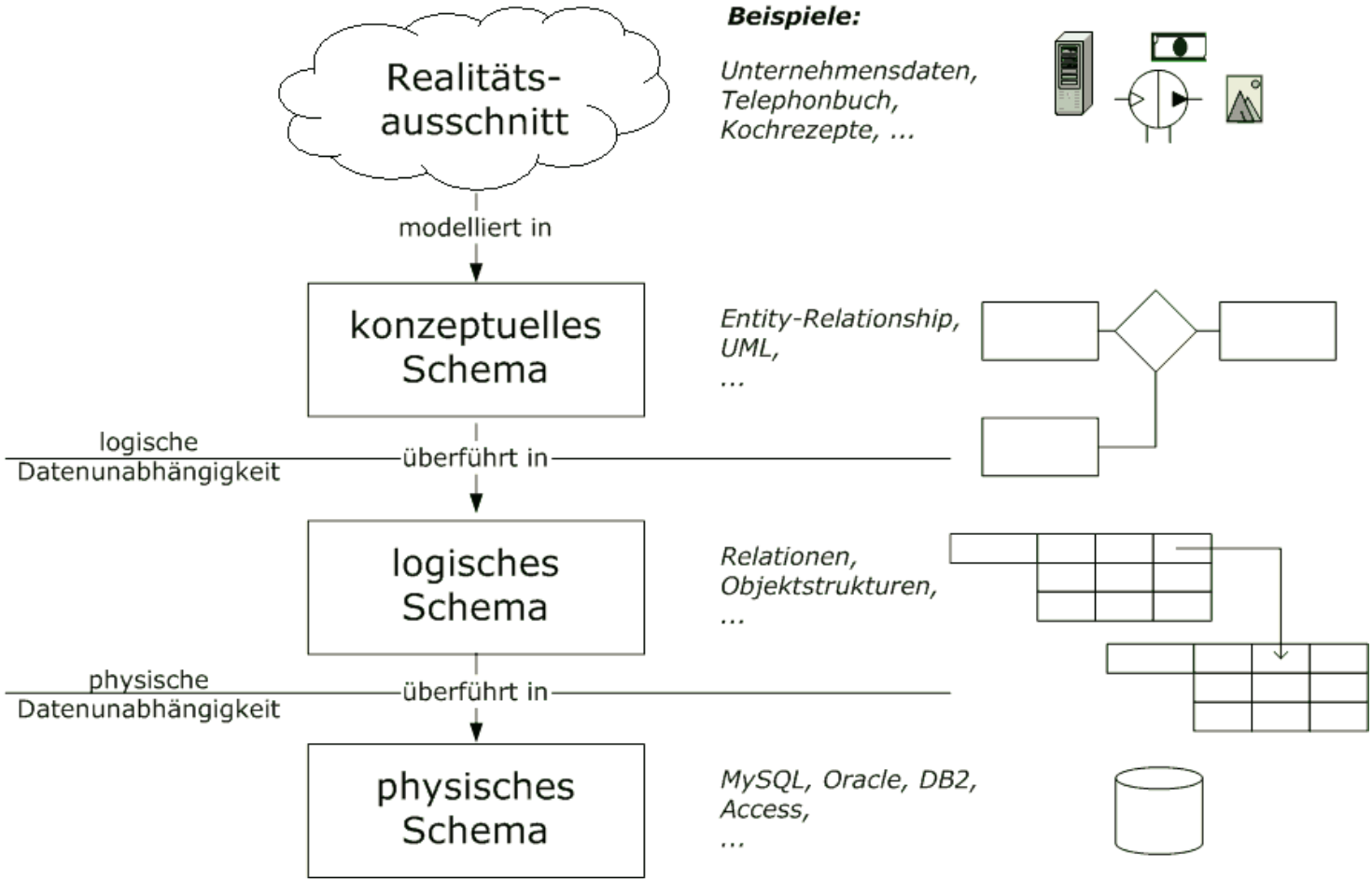
More Microsoft Business Solutions ...

More MSN ...

- Use Office Update to scan your computer and install updates

More Searches ...

Last Updated: Monday, June 7, 2004 - 11:02 a.m. Pacific Time


Realitäts-
ausschnitt
Beispiele:
Unternehmensdaten,
Telephonbuch,
Kochrezepte, ...
modelliert in
konzeptuelles
Schema
Entity-Relationship,
UML,
...
logische
Datenunabhängigkeit
überführt in
logisches
Schema
Relationen,
Objektstrukturen,
...
physische
Datenunabhängigkeit
überführt in
physisches
Schema
MySQL, Oracle, DB2,
Access,
...

Vorlesung Datenbanken
Scriptum zur Vorlesung Datenbanken

Hinweis

Die nachfolgende Darstellung legt eine Installation des Datenbankmanagementsystems MySQL ab Version 4.1 auf einer Linux-Maschine zugrunde. Die gezeigten Aufrufe sollten sich jedoch, nach Anpassung der Pfadangaben und Parametersyntax auch auf die übrigen Plattformen (etwa: Microsoft Windows) übertragen lassen.

1 Vorarbeiten

Laden Sie sich den Quellcode des Datenbankmanagementsystems oder eine der verfügbaren vorübersetzten Binärdistributionen von der Seite des Herstellers.
Achten Sie darauf, mindestens Version 4.1 zu beziehen!

Hinweis: Für den Praktikumsbetrieb genügt der Bezug des vorübersetzten Paketes völlig.
Darüber hinaus ist das DBMS bereits auf den zur Verfügung stehenden Maschinen vorinstalliert und muß nicht mehr durch Sie bezogen werden. Dieser Schritt kann daher entfallen.

2 Installation

Installieren Sie MySQL gemäß der Angaben im Angaben im Installationshandbuch.

Hinweis: Auf den Praktikumsmaschinen ist das DBMS bereits vorinstalliert, dieser Schritt kann daher entfallen.
Nachfolgend wird von einer Installation des DBMS in den Katalog `/usr/local/mysql` ausgegangen, bzw. von der entsprechenden Definition eines symbolischen Links. Sollten Sie MySQL an anderer Stelle im Dateisystem installiert haben, so ändern sich die Pfade entsprechend.

3 Konfiguration

Wir werden später Funktionalitäten (insbesondere die Mechanismen zur Erhaltung der referentiellen Integrität) nutzen, die gesondert aktiviert werden müssen. Modifizieren Sie hierfür die Datei `/etc/my.cnf` folgendermaßen:
Fügen Sie mit einem Texteditor (z.B. vi) im Abschnitt `[mysqld]` der Datei folgende Zeile ein: `innodb_data_file_path = ibdata1:10M:autoextend`.

Sollte die Datei `my.cnf` noch nicht im Katalog `/etc` existieren, so können Sie diese durch kopieren und umbenennen der Datei `/usr/local/mysql/support-files/my-medium.cnf` erzeugen.
Notwendige Schritte:
`cp /usr/local/mysql/support-files/my-medium.cnf /etc/my.cnf`

Hinweis: Aufgrund des Kopiervorganges in den nur für den Benutzer `root` schreibberechtigten Katalog `/etc` muß der Kopiervorgang durch diesen Benutzer erfolgen.

4 Start des DBMS

Der Benutzer `root` startet MySQL durch Ausführung des Befehls `./bin/mysqld_safe --default-table-type=InnoDB` im Katalog `/usr/local/mysql`.
Danach steht das DBMS anderen Systembenutzern zur Verfügung.

Hinweis: Im Praktikumsbetrieb ist das DBMS bereits gestartet und muß nicht mehr durch Sie erfolgen.

5 Erstellung der Beispieldatenbank

Melden Sie sich hierzu mir Ihrer persönlichen Benutzerkennung am System an!

5.1 Erzeugen der Datenbank

Durch den Datenbankadministrator (standardmäßig als `root` benannt

und durch kein Paßwort gesichert.

Hinweis: Im Praktikumsbetrieb ist Ihre Datenbank bereits unter dem Namen Ihrer Praktikumsgruppe erzeugt und muß nicht mehr durch Sie erstellt werden, dieser Teilschritt kann daher entfallen.

Ausführen: `mysqladmin -u root create DBName`
Achten Sie darauf, daß sich `mysqladmin` im Pfad befindet!

5.2 Vergabe der notwendigen Zugriffsberechtigungen für die Datenbank

Modifizieren Sie zunächst das Berechtigungsskript so, daß statt *IHR_NAME* Ihre Benutzerkennung dort eingetragen ist.

Führen Sie dieses Skript mittels `mysql -u root < grants.sql` aus.

Hinweis: Für das Praktikum wurde das Skript bereits für alle Mitglieder Ihrer Praktikumsgruppe zur Ausführung gebracht und muß daher nicht mehr durch Sie aufgerufen werden!

5.3 Laden der Datenbankinhalte

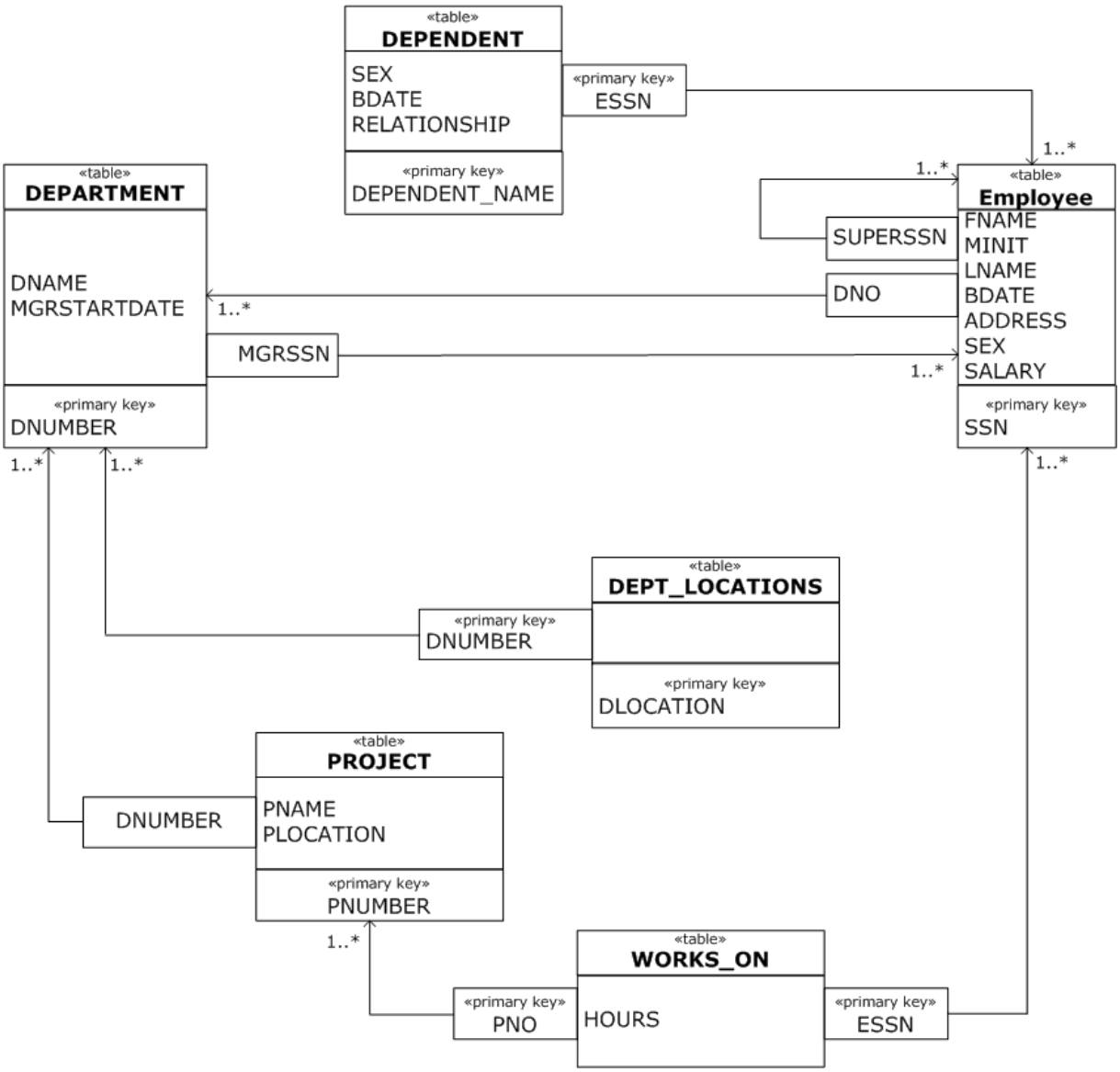
Laden Sie den Datenbankinhalt (Relationen und darin abgelegte Beispieldaten) mit `mysql Datenbankname < demo-db.sql` in Ihre Datenbank.

Hinweis: Ersetzen Sie *Datenbankname* durch den Namen Ihrer Praktikumsgruppe!

6 Die Demo-DB

6.1 Struktur

Abbildung 1: *Struktur der Demo-DB*



(click on image to enlarge!)

6.2 Dateninhalte

Die Relation EMPLOYEE

```
+----------+-------+---------+-----------+-----------
+--------------------------+------+----------+-----------
+------+
| FNAME    | MINIT | LNAME   | SSN       | BDATE      |
ADDRESS                  | SEX  | SALARY   | SUPERSSN  |
DNO  |
+----------+-------+---------+-----------+-----------
+--------------------------+------+----------+-----------
+------+
| John     | B     | Smith   | 123456789 | 1965-01-09 |
731 Fondren, Houston, TX | M    | 30000.00 | 333445555
|    5 |
| Franklin | T     | Wong    | 333445555 | 1955-12-08 |
638 Voss, Houston, TX    | M    | 40000.00 | 888665555
|    5 |
| Joyce    | A     | English | 453453453 | 1972-07-31 |
```

```
5631 Rice, Houston, TX   | F    | 25000.00 | 333445555
|    5 |
| Ramesh   | K     | Narayan | 666884444 | 1962-09-15 |
975 Fire Oak, Humble, TX | M    | 38000.00 | 333445555
|    5 |
| James    | E     | Borg    | 888665555 | 1937-11-10 |
450 Stone, Houston, TX   | M    | 55000.00 |      NULL
|    1 |
| Jennifer | S     | Wallace | 987654321 | 1941-06-20 |
291 Berry, Bellaire, TX  | F    | 43000.00 | 888665555
|    4 |
| Ahmad    | V     | Jabbar  | 987987987 | 1969-03-29 |
980 Dallas, Houston, TX  | M    | 25000.00 | 987654321
|    4 |
| Alicia   | J     | Zelaya  | 999887777 | 1968-07-19 |
3321 Castle, Spring, TX  | F    | 25000.00 | 987654321
|    4 |
+----------+-------+---------+-----------+------------
+--------------------------+------+----------+-----------
+------+
```

Die Relation DEPT_LOCATIONS

```
+---------+-----------+
| DNUMBER | DLOCATION |
+---------+-----------+
|       1 | Houston   |
|       4 | Stafford  |
|       5 | Bellaire  |
|       5 | Houston   |
|       5 | Sugarland |
+---------+-----------+
```

Die Relation DEPARTMENT

```
+---------------+---------+-----------+--------------+
| DNAME         | DNUMBER | MGRSSN    | MGRSTARTDATE |
+---------------+---------+-----------+--------------+
| Headquarters  |       1 | 888665555 | 1981-06-19   |
| Administation |       4 | 987654321 | 1995-01-01   |
| Research      |       5 | 333445555 | 1988-05-22   |
+---------------+---------+-----------+--------------+
```

Die Relation WORKS_ON

```
+-----------+-----+-------+
| ESSN      | PNO | HOURS |
+-----------+-----+-------+
| 123456789 |   1 |  32.5 |
```

```
| 123456789 |   2 |   7.5 |
| 333445555 |   2 |  10.0 |
| 333445555 |   3 |  10.0 |
| 333445555 |  10 |  10.0 |
| 333445555 |  20 |  10.0 |
| 453453453 |   1 |  20.0 |
| 453453453 |   2 |  20.0 |
| 666884444 |   3 |  40.0 |
| 888665555 |  20 |  NULL |
| 987654321 |  20 |  15.0 |
| 987654321 |  30 |  20.0 |
| 987987987 |  10 |  35.0 |
| 987987987 |  30 |   5.0 |
| 999887777 |  10 |  10.0 |
| 999887777 |  30 |  30.0 |
+-----------+-----+-------+
```

Die Relation PROJECT

```
+-----------------+---------+-----------+------+
| PNAME           | PNUMBER | PLOCATION | DNUM |
+-----------------+---------+-----------+------+
| ProductX        |       1 | Bellaire  |    5 |
| ProductY        |       2 | Sugarland |    5 |
| ProductZ        |       3 | Houston   |    5 |
| Computerization |      10 | Stafford  |    4 |
| Reorganization  |      20 | Houston   |    1 |
| Newbenefits     |      30 | Stafford  |    4 |
+-----------------+---------+-----------+------+
```

Die Relation DEPENDENT

```
+-----------+----------------+------+------------
+--------------+
| ESSN      | DEPENDENT_NAME | SEX  | BDATE      |
RELATIONSHIP |
+-----------+----------------+------+------------
+--------------+
| 333445555 | Alice          | F    | 1986-04-05
|              |
| 333445555 | Theodore       | M    | 1983-10-25 |
SON          |
| 333445555 | Joy            | F    | 1958-05-03 |
SPOUSE       |
| 987654321 | Abner          | M    | 1942-02-28 |
SPOUSE       |
| 123456789 | Michael        | M    | 1988-01-04 |
SON          |
| 123456789 | Alice          | F    | 1988-12-30
|              |
| 123456789 | Elizabeth      | F    | 1967-05-05 |
SPOUSE       |
```

```
+-----------+----------------+------+------------
+--------------+
```

7 Arbeiten mit der Beispieldatenbank --- Einige erste SQL-Anfragen

Alle nachfolgend aufgeführten Anfragen werden in der interaktiven MySQL-Clientumgebung unter der persönlichen Systemkennung ausgeführt.
Die Clientumgebung wird mit `mysql` *`Datenbankname`* gestartet und fordert durch den Prompt `mysql>` zur Eingabe auf.
Beendet wird der Client das Kommando `exit`.

7.1 Anzeigen von Relationsbeschreibungen

Anfrage: `DESCRIBE EMPLOYEE;`

```
+----------+----------------------+------+-----+---------
+-------+
| Field    | Type                 | Null | Key | Default |
Extra |
+----------+----------------------+------+-----+---------
+-------+
| FNAME    | varchar(10)          |      |     |
|       |
| MINIT    | char(1)              | YES  |     | NULL
|       |
| LNAME    | varchar(10)          |      |     |
|       |
| SSN      | int(9)               |      | PRI | 0
|       |
| BDATE    | date                 | YES  |     | NULL
|       |
| ADDRESS  | varchar(20)          | YES  |     | NULL
|       |
| SEX      | enum('M','F')        | YES  |     | NULL
|       |
| SALARY   | double(7,2) unsigned | YES  |     | NULL
|       |
| SUPERSSN | int(9)               | YES  | MUL | NULL
|       |
| DNO      | int(1)               | YES  | MUL | NULL
|       |
+----------+----------------------+------+-----+---------
+-------+
```

7.2 Anzeigen aller Tupel der Relation EMPLOYEE

Anfrage:SELECT * FROM EMPLOYEE;

Ausgabe:

```
+----------+-------+---------+-----------+------------
+--------------------------+------+----------+-----------
+------+
| FNAME    | MINIT | LNAME   | SSN       | BDATE      |
ADDRESS                  | SEX  | SALARY   | SUPERSSN  |
DNO  |
+----------+-------+---------+-----------+------------
+--------------------------+------+----------+-----------
+------+
| John     | B     | Smith   | 123456789 | 1965-01-09 |
731 Fondren, Houston, TX | M    | 30000.00 | 333445555
|    5 |
| Franklin | T     | Wong    | 333445555 | 1955-12-08 |
638 Voss, Houston, TX    | M    | 40000.00 | 888665555
|    5 |
| Joyce    | A     | English | 453453453 | 1972-07-31 |
5631 Rice, Houston, TX   | F    | 25000.00 | 333445555
|    5 |
| Ramesh   | K     | Narayan | 666884444 | 1962-09-15 |
975 Fire Oak, Humble, TX | M    | 38000.00 | 333445555
|    5 |
| James    | E     | Borg    | 888665555 | 1937-11-10 |
450 Stone, Houston, TX   | M    | 55000.00 |      NULL
|    1 |
| Jennifer | S     | Wallace | 987654321 | 1941-06-20 |
291 Berry, Bellaire, TX  | F    | 43000.00 | 888665555
|    4 |
| Ahmad    | V     | Jabbar  | 987987987 | 1969-03-29 |
980 Dallas, Houston, TX  | M    | 25000.00 | 987654321
|    4 |
| Alicia   | J     | Zelaya  | 999887777 | 1968-07-19 |
3321 Castle, Spring, TX  | F    | 25000.00 | 987654321
|    4 |
+----------+-------+---------+-----------+------------
+--------------------------+------+----------+-----------
+------+
```

7.3 Anzeigen der Werte des Attributs DNAME der Relation DEPARTMENT

Anfrage:SELECT DNAME FROM DEPARTMENT;

Ausgabe:

```
+---------------+
| DNAME         |
+---------------+
| Headquarters  |
| Administation |
| Research      |
+---------------+
```

7.4 Anzeigen des Geburtsdatums und der Adresse des Mitarbeiters *John B. Smith*

Anfrage:

```
SELECT BDATE, ADDRESS
FROM EMPLOYEE
WHERE FNAME='John' AND MINIT='B'AND LNAME='Smith';
```

Ausgabe:

```
+------------+----------------------+
| BDATE      | ADDRESS              |
+------------+----------------------+
| 1965-01-09 | 731 Fondren, Houston |
+------------+----------------------+
```

7.5 Anfrage auf zwei Relationen (kartesisches Produkt)

Anfrage:

```
SELECT FNAME, DNAME
FROM EMPLOYEE, DEPARTMENT;
```

Die Anfrage liefert alle Werte des Attributs `FNAME` in allen denkbaren Kombinationen mit allen Werten des Attributs `DNAME`.

Ausgabe:

```
+----------+---------------+
| FNAME    | DNAME         |
+----------+---------------+
| John     | Headquarters  |
| Franklin | Headquarters  |
| Joyce    | Headquarters  |
| Ramesh   | Headquarters  |
| James    | Headquarters  |
| Jennifer | Headquarters  |
| Ahmad    | Headquarters  |
| Alicia   | Headquarters  |
```

```
| John     | Administation |
| Franklin | Administation |
| Joyce    | Administation |
| Ramesh   | Administation |
| James    | Administation |
| Jennifer | Administation |
| Ahmad    | Administation |
| Alicia   | Administation |
| John     | Research      |
| Franklin | Research      |
| Joyce    | Research      |
| Ramesh   | Research      |
| James    | Research      |
| Jennifer | Research      |
| Ahmad    | Research      |
| Alicia   | Research      |
+----------+---------------+
```

7.6 Verbund zweier Relationen

Anfrage:

```
SELECT FNAME, LNAME, ADDRESS
FROM EMPLOYEE, DEPARTMENT
WHERE DNAME='Research' AND DNUMBER=DNO;
```

Die Anfrage liefert die Vornamen, Nachname und Adressen aller Mitarbeiter, die in der Research-Abteilung arbeiten.

Ausgabe:

```
+----------+---------+----------------------+
| FNAME    | LNAME   | ADDRESS              |
+----------+---------+----------------------+
| John     | Smith   | 731 Fondren, Houston |
| Franklin | Wong    | 638 Voss, Houston, T |
| Joyce    | English | 5631 Rice, Houston,  |
| Ramesh   | Narayan | 975 Fire Oak, Humble |
+----------+---------+----------------------+
```

7.7 Verbund mehrerer Relationen

Anfrage:

```
SELECT PNUMBER, DNUM, LNAME, ADDRESS, BDATE
FROM PROJECT, DEPARTMENT, EMPLOYEE
WHERE DNUM=DNUMBER AND MGRSSN=SSN AND PLOCATION='Stafford';
```

Die Anfrage liefert Projekt-, Abteilungsnummer, Nachname, Adresse und Geburtsdatum aller Mitarbeiter die Abteilungen leiten, welche in `Stafford` niedergelassen sind.

Ausgabe:

```
+---------+------+---------+---------------------
+------------+
| PNUMBER | DNUM | LNAME   | ADDRESS             |
BDATE      |
+---------+------+---------+---------------------
+------------+
|      10 |    4 | Wallace | 291 Berry, Bellaire, | 1941-
06-20 |
|      30 |    4 | Wallace | 291 Berry, Bellaire, | 1941-
06-20 |
+---------+------+---------+---------------------
+------------+
```

7.8 Aliasbildung

Anfrage:

```
SELECT E.FNAME, E.LNAME, S.FNAME, S.LNAME
FROM EMPLOYEE AS E, EMPLOYEE AS S
WHERE E.SUPERSSN=S.SSN;
```

Die Anfrage liefert Vor- und Nachnamen aller Mitarbeiter gemeinsam mit den Vor- und Nachnamen ihrer Vorgesetzten.

Ausgabe:

```
+----------+---------+----------+---------+
| FNAME    | LNAME   | FNAME    | LNAME   |
+----------+---------+----------+---------+
| John     | Smith   | Franklin | Wong    |
| Franklin | Wong    | James    | Borg    |
| Joyce    | English | Franklin | Wong    |
| Ramesh   | Narayan | Franklin | Wong    |
| Jennifer | Wallace | James    | Borg    |
| Ahmad    | Jabbar  | Jennifer | Wallace |
| Alicia   | Zelaya  | Jennifer | Wallace |
+----------+---------+----------+---------+
```

7.9 Duplikatbehaftete Ausgabe

Anfrage:

```
SELECT ALL SALARY
```

```
FROM EMPLOYEE;
```

Man beachte, diese Anfrage liefert dieselbe Ausgabe wie:

```
SELECT SALARY
FROM EMPLOYEE;
```

Die Anfrage liefert alle Werte des Attributs SALARY.

Ausgabe:

```
+----------+
| SALARY   |
+----------+
| 30000.00 |
| 40000.00 |
| 25000.00 |
| 38000.00 |
| 55000.00 |
| 43000.00 |
| 25000.00 |
| 25000.00 |
+----------+
```

7.10 Duplikatfreie

Anfrage:

```
SELECT DISTINCT SALARY
FROM EMPLOYEE;
```

Die Anfrage liefert alle Werte des Attributs SALARY, jedoch jeden einzelnen Wert nur genau einmal.

Ausgabe:

```
+----------+
| SALARY   |
+----------+
| 30000.00 |
| 40000.00 |
| 25000.00 |
| 38000.00 |
| 55000.00 |
| 43000.00 |
+----------+
```

7.11 Vereinigung von Anfrageergebnissen

Anfrage:

```
(SELECT DISTINCT PNUMBER
FROM PROJECT, DEPARTMENT, EMPLOYEE
WHERE DNUM=DNUMBER AND MGRSSN=SSN AND LNAME='Smith')
UNION
(SELECT DISTINCT PNUMBER
FROM PROJECT, WORKS_ON, EMPLOYEE
WHERE PNUMBER=PNO AND ESSN=SSN AND LNAME='Smith');
```

Die erste Anfrage liefert diejenigen Projektnummern, welche einem Projekt zugeordnet sind, daß der von *Smith* geleiteten Abteilung zugeordnet ist.
Die zweite Anfrage liefert alle Projekte an denen *Smith* mitarbeitet.

Ausgabe:

```
+---------+
| PNUMBER |
+---------+
|       1 |
|       2 |
+---------+
```

7.12 Anfrage von textuellen Attributteilen

Anfrage:

```
SELECT FNAME, LNAME
FROM EMPLOYEE
WHERE ADDRESS LIKE '%Houston, TX%';
```

Ausgabe:

Die Anfrage liefert die Vor- und Nachnamen aller Mitarbeiter die in *Houston, TX* wohnen, d.h. in deren Adresse dieser Teilausdruck auftritt.

```
+----------+---------+
| FNAME    | LNAME   |
+----------+---------+
| John     | Smith   |
| Franklin | Wong    |
| Joyce    | English |
| James    | Borg    |
| Ahmad    | Jabbar  |
+----------+---------+
```

7.13 Anfrage von Teilen eines Datumsattributs

Anfrage:

```
SELECT FNAME, LNAME
FROM EMPLOYEE
WHERE BDATE LIKE '__5_______';
```

Die Anfrage liefert alle Mitarbeiter, die in den 50er Jahren geboren wurden.

Ausgabe:

```
+--------+---------+
| FNAME  | LNAME   |
+--------+---------+
| John   | Smith   |
| Ramesh | Narayan |
| Ahmad  | Jabbar  |
| Alicia | Zelaya  |
+--------+---------+
```

7.14 Einfache arithmetische Operationen

Anfrage:

```
SELECT FNAME, LNAME, 1.1*SALARY
FROM EMPLOYEE, WORKS_ON, PROJECT
WHERE SSN=ESSN AND PNO=PNUMBER AND PNAME='ProductX';
```

Ausgabe:

Zeigt die Höhe des Gehalts aller am *ProductX* beteiligten Mitarbeiter nach einer 10% Lohnerhöhung.

```
+-------+---------+------------+
| FNAME | LNAME   | 1.1*SALARY |
+-------+---------+------------+
| John  | Smith   |   33000.00 |
| Joyce | English |   27500.00 |
+-------+---------+------------+
```

7.15 Anfragen nach Wertebereichen

Anfrage:

```
SELECT *
FROM EMPLOYEE
WHERE (SALARY BETWEEN 30000 AND 40000) AND DNO=5;
```

Ausgabe:

Die Anfrage liefert alle in der Relation EMPLOYEE verfübaren Angaben zu denjenigen Mitarbeitern deren Gehalt zwischen 30000 und 40000 liegt und die in Abteilung 5 beschäftigt sind.

```
+----------+-------+---------+-----------+------------
+--------------------------+------+----------+-----------
+------+
| FNAME    | MINIT | LNAME   | SSN       | BDATE      |
ADDRESS                  | SEX  | SALARY   | SUPERSSN  |
DNO  |
+----------+-------+---------+-----------+------------
+--------------------------+------+----------+-----------
+------+
| John     | B     | Smith   | 123456789 | 1965-01-09 |
731 Fondren, Houston, TX | M    | 30000.00 | 333445555
|    5 |
| Franklin | T     | Wong    | 333445555 | 1955-12-08 |
638 Voss, Houston, TX    | M    | 40000.00 | 888665555
|    5 |
| Ramesh   | K     | Narayan | 666884444 | 1962-09-15 |
975 Fire Oak, Humble, TX | M    | 38000.00 | 333445555
|    5 |
+----------+-------+---------+-----------+------------
+--------------------------+------+----------+-----------
+------+
```

7.16 Sortierte Ausgabe

Anfrage:

```
SELECT DNAME, LNAME, FNAME, PNAME
FROM DEPARTMENT, EMPLOYEE, WORKS_ON, PROJECT
WHERE DNUMBER=DNO AND SSN=ESSN AND PNO=PNUMBER
ORDER BY DNAME, LNAME, FNAME;
```

Liefert die Abteilungsnamen sowie die Vor- und Nachnamen ihrer Leiter, gefolgt von den in den Abteilungen durchgeführten Projekten, aufsteigend sortiert nach Abteilungsname, dem Vor- sowie dem Nachnamen des Abteilungsleiters.

Ausgabe:

```
+--------------+---------+----------+-----------------+
```

DNAME	LNAME	FNAME	PNAME
Administation	Jabbar	Ahmad	Computerization
Administation	Jabbar	Ahmad	Newbenefits
Administation	Wallace	Jennifer	Reorganization
Administation	Wallace	Jennifer	Newbenefits
Administation	Zelaya	Alicia	Newbenefits
Administation	Zelaya	Alicia	Computerization
Headquarters	Borg	James	Reorganization
Research	English	Joyce	ProductY
Research	English	Joyce	ProductX
Research	Narayan	Ramesh	ProductZ
Research	Smith	John	ProductX
Research	Smith	John	ProductY
Research	Wong	Franklin	ProductY
Research	Wong	Franklin	Computerization
Research	Wong	Franklin	ProductZ
Research	Wong	Franklin	Reorganization

Anfrage:

```
SELECT DNAME, LNAME, FNAME, PNAME
FROM DEPARTMENT, EMPLOYEE, WORKS_ON, PROJECT
WHERE DNUMBER=DNO AND SSN=ESSN AND PNO=PNUMBER
ORDER BY DNAME DESC, LNAME ASC, FNAME ASC;
```

Liefert die Abteilungsnamen sowie die Vor- und Nachnamen ihrer Leiter, gefolgt von den in den Abteilungen durchgeführten Projekten, absteigend sortiert nach Abteilungsname und aufsteigend sortiert nach dem Vor- sowie dem Nachnamen des Abteilungsleiters.

Ausgabe:

DNAME	LNAME	FNAME	PNAME
Research	English	Joyce	ProductY
Research	English	Joyce	ProductX
Research	Narayan	Ramesh	ProductZ
Research	Smith	John	ProductX
Research	Smith	John	ProductY
Research	Wong	Franklin	ProductY
Research	Wong	Franklin	Computerization
Research	Wong	Franklin	ProductZ
Research	Wong	Franklin	Reorganization
Headquarters	Borg	James	Reorganization
Administation	Jabbar	Ahmad	Computerization
Administation	Jabbar	Ahmad	Newbenefits
Administation	Wallace	Jennifer	Reorganization
Administation	Wallace	Jennifer	Newbenefits
Administation	Zelaya	Alicia	Newbenefits
Administation	Zelaya	Alicia	Computerization

```
+---------------+---------+----------+-----------------+
```

7.17 Mehr zum Thema:

- Teil 1: Grundprinzipien, Tabellenerstellung, Einfügen von Daten und einfache Anfragen
- Teil 2: Mächtigere Anfragen, Aktualisieren und Löschen von Daten
- Teil 3: Ausgabesortierung, Verbünde und Datenaggregation
- Subqueries in MySQL, Part 1
- SQL-Functions (Kapitel aus K. Kline, D. Kline: *SQL in a Nutshell*)
- SQL Under The Covers (Artikel über SQL-Verarbeitung in DB2)
- Hierarchical Queries

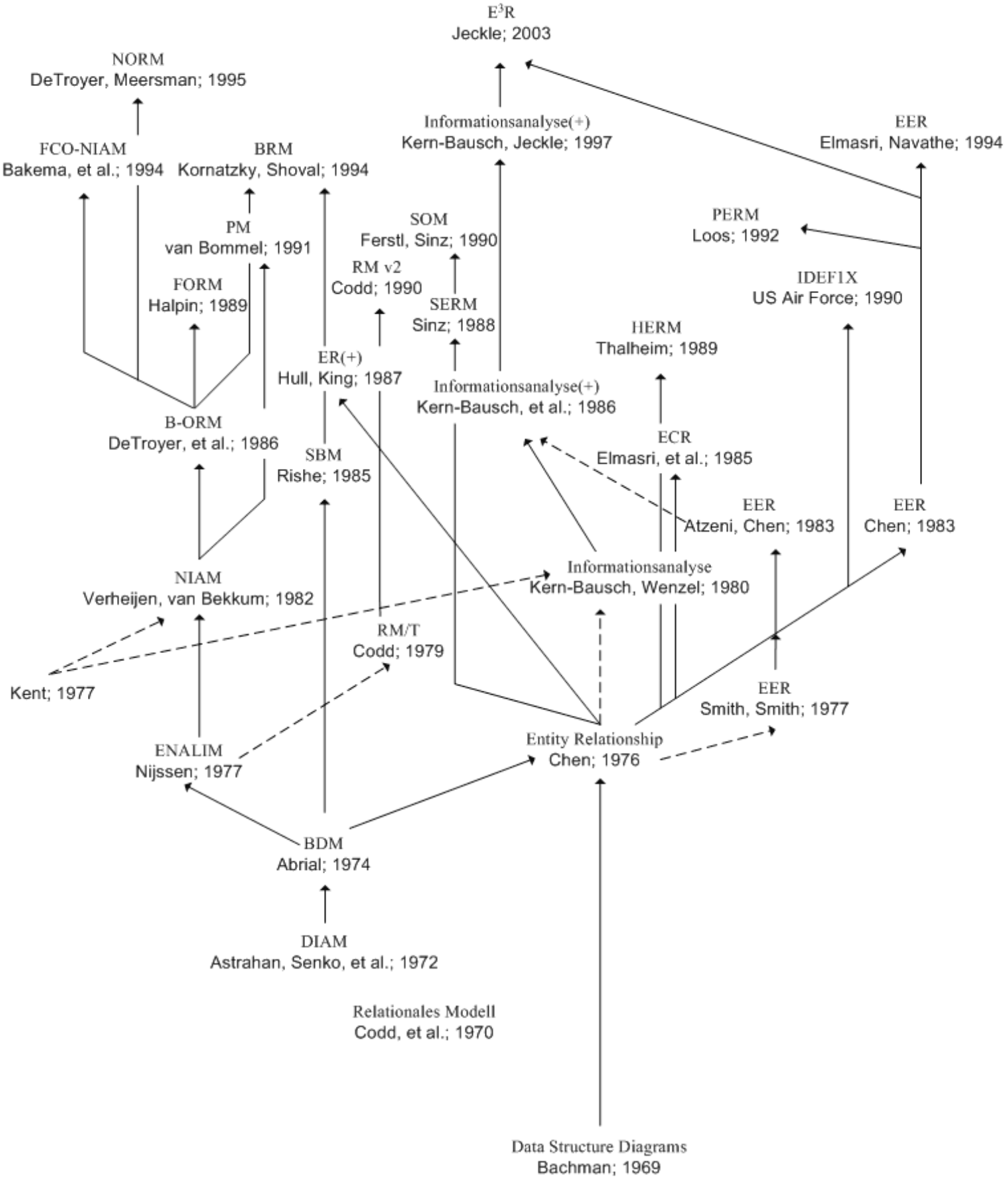
Service provided by Mario Jeckle

Generated: 2004-06-08T12:50:57+01:00

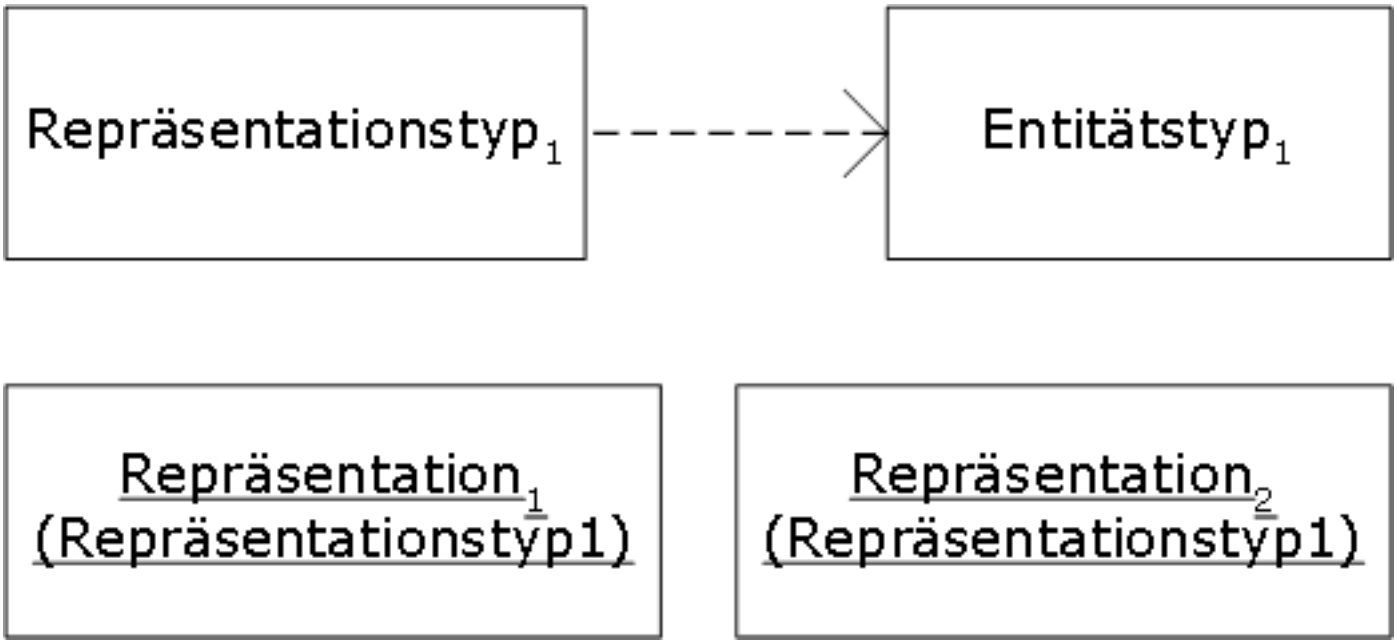
Feedback SiteMap

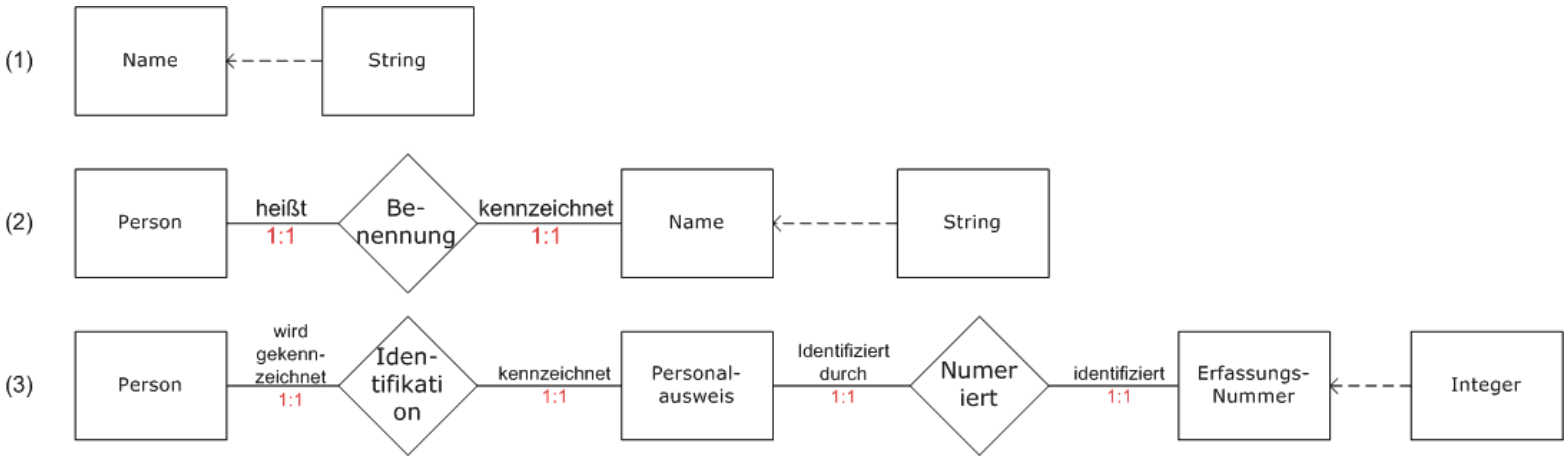
This page's original location: http://www.jeckle.de/vorlesung/datenbanken/MySQLStart.html

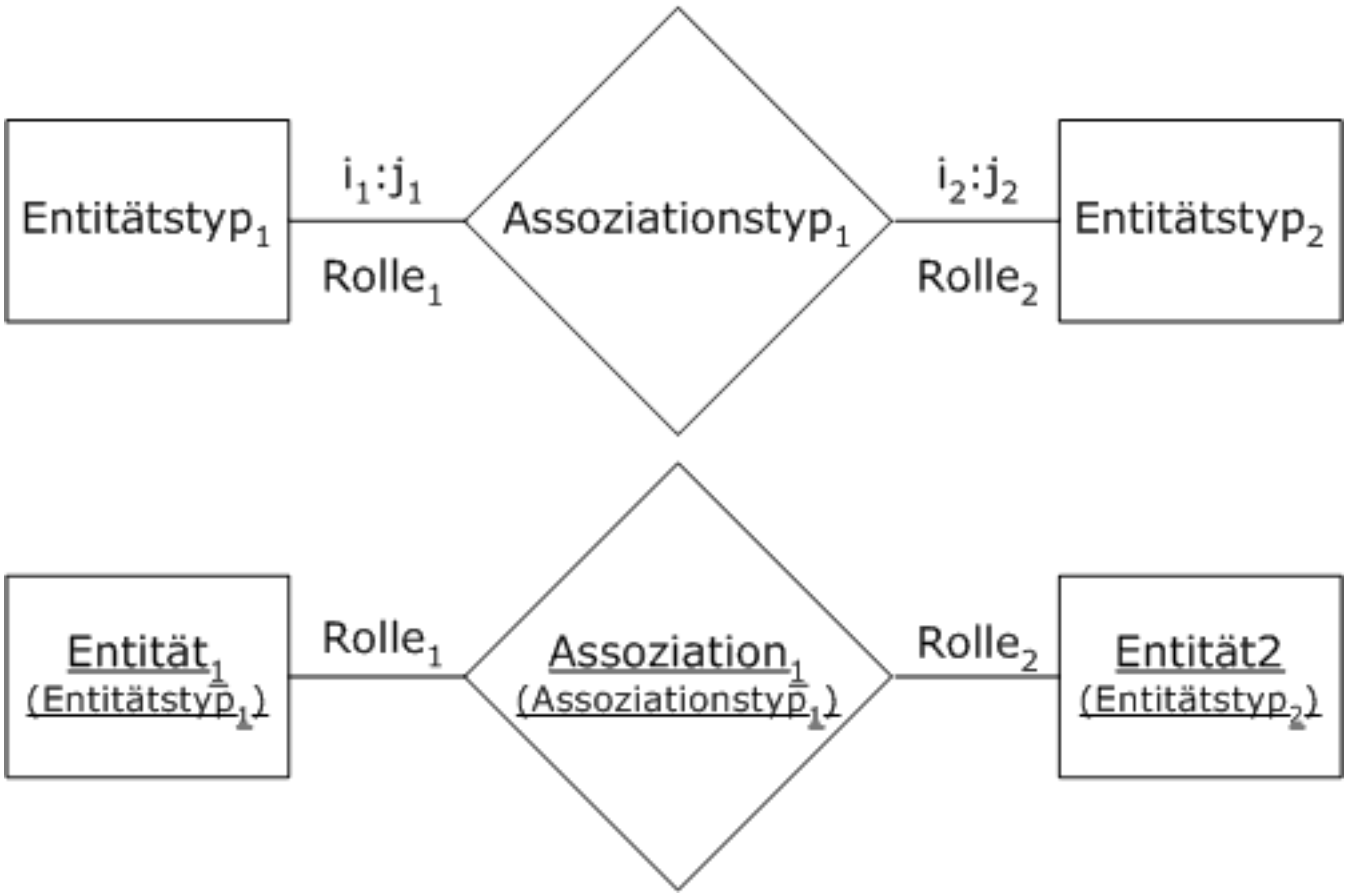
RDF description for this page

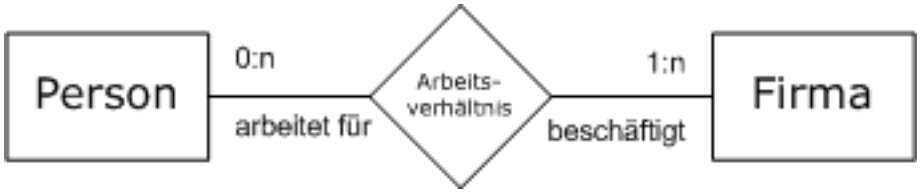

E3R
Jeckle; 2003
NORM
DeTroyer, Meersman; 1995
Informationsanalyse(+)
Kern-Bausch, Jeckle; 1997
EER
Elmasri, Navathe; 1994
FCO-NIAM
Bakema, et al.; 1994
BRM
Kornatzky, Shoval; 1994
PM
van Bommel; 1991
SOM
Ferstl, Sinz; 1990
PERM
Loos; 1992
RM v2
Codd; 1990
FORM
Halpin; 1989
SERM
Sinz; 1988
IDEF1X
US Air Force; 1990
HERM
Thalheim; 1989
ER(+)
Hull, King; 1987
Informationsanalyse(+)
Kern-Bausch, et al.; 1986
B-ORM
DeTroyer, et al.; 1986
ECR
Elmasri, et al.; 1985
SBM
Rishe; 1985
EER
Atzeni, Chen; 1983
EER
Chen; 1983
Informationsanalyse
Kern-Bausch, Wenzel; 1980
NIAM
Verheijen, van Bekkum; 1982
RM/T
Codd; 1979
Kent; 1977
EER
Smith, Smith; 1977
Entity Relationship
Chen; 1976
ENALIM
Nijssen; 1977
BDM
Abrial; 1974
DIAM
Astrahan, Senko, et al.; 1972
Relationales Modell
Codd, et al.; 1970
Data Structure Diagrams
Bachman; 1969

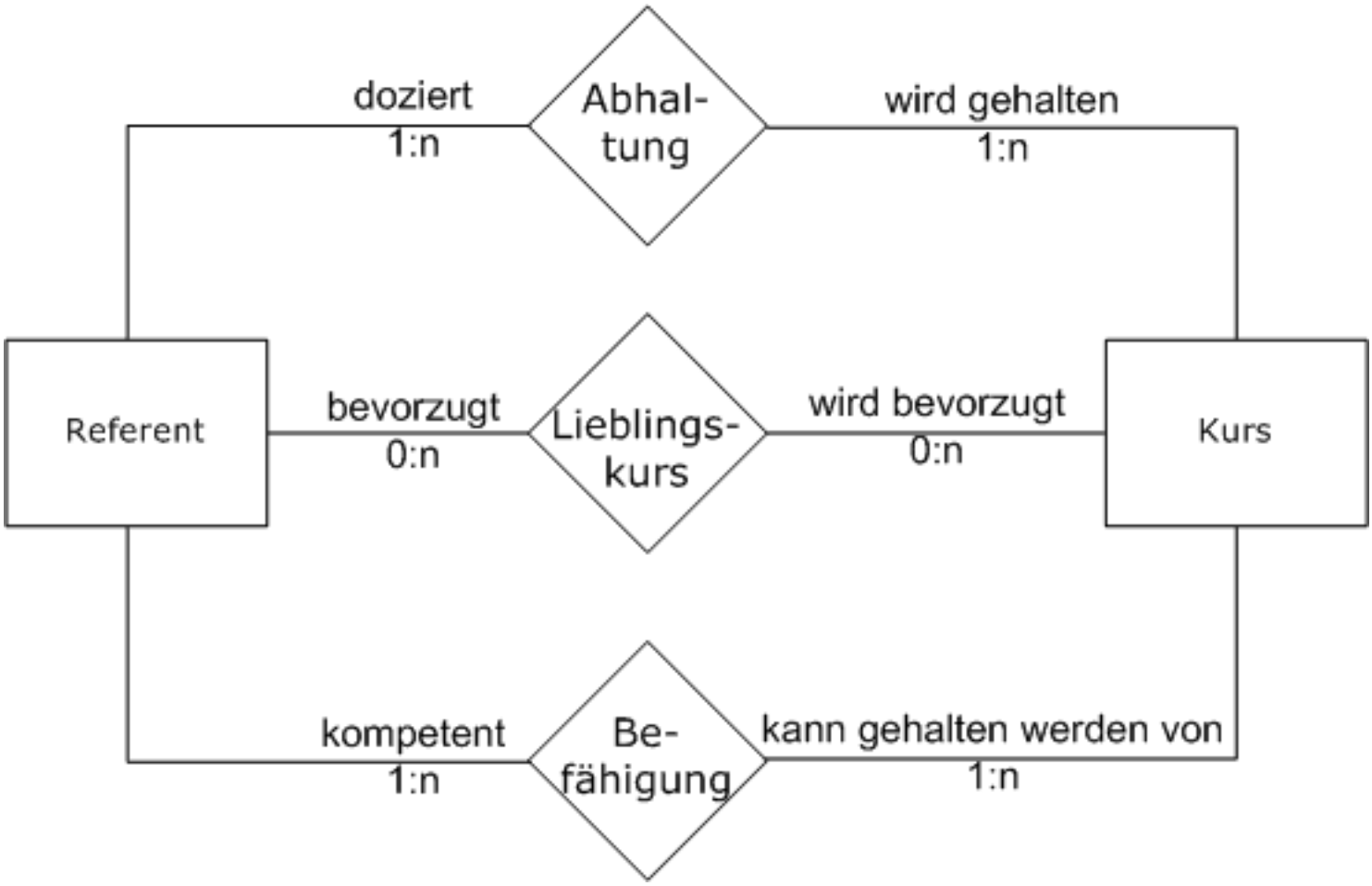

Entitätstyp1
Entität1 (Entitätstyp1)

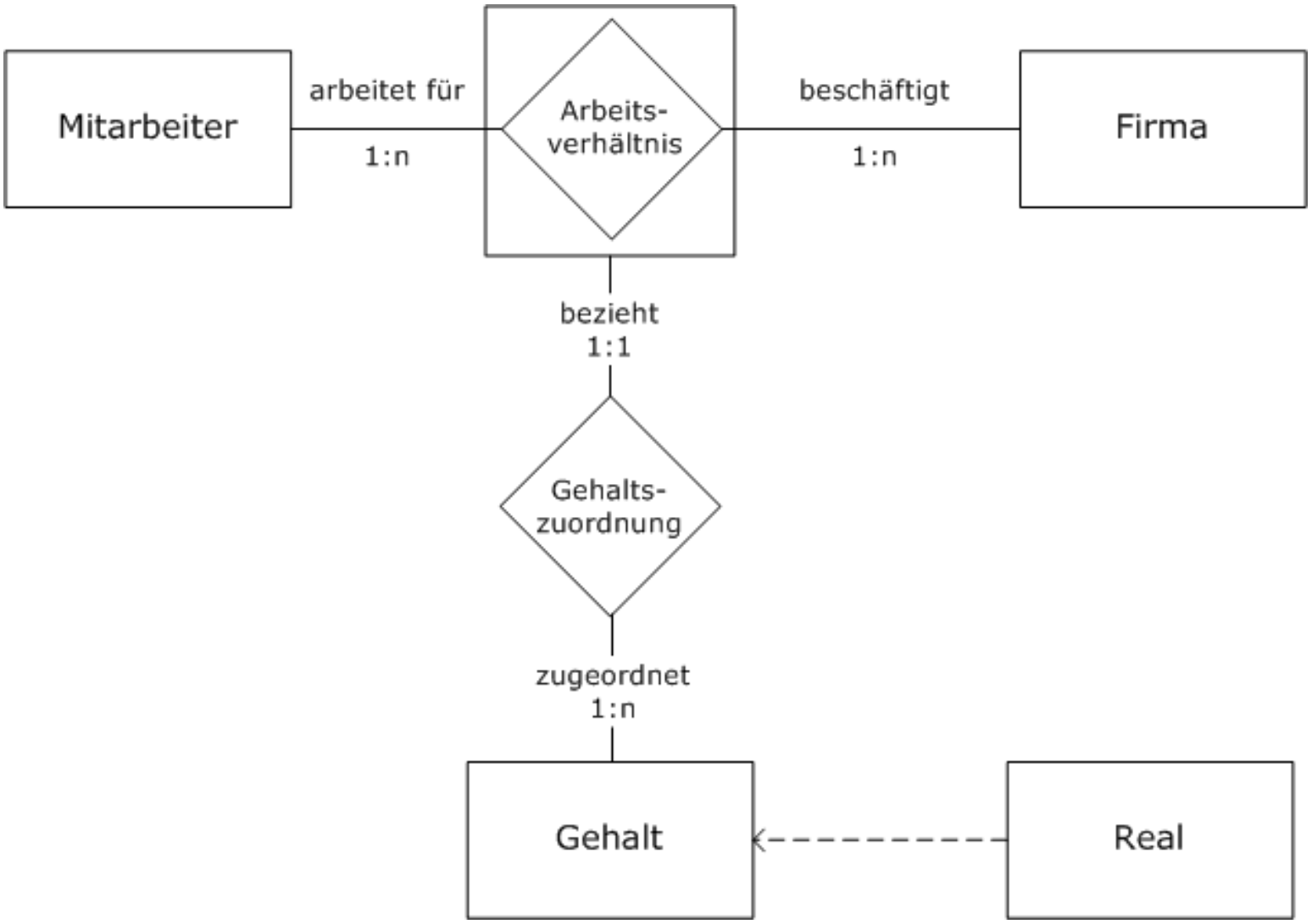

Repräsentationstyp1
Entitätstyp1
Repräsentation1
(Repräsentationstyp1)
Repräsentation2
(Repräsentationstyp1)

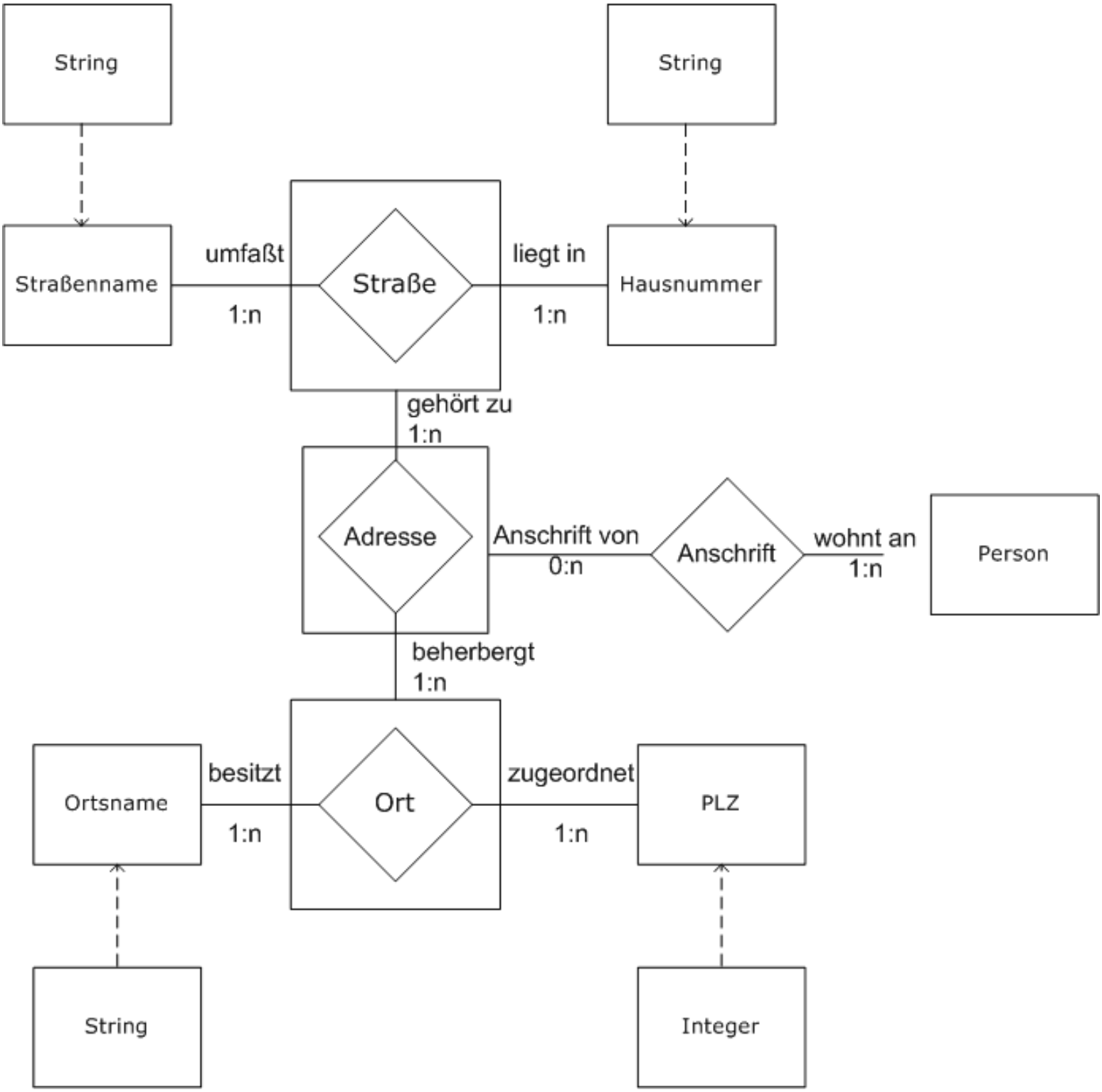

(1)
Name
String
(2)
Person
heißt
1:1
Be-
nennung
kennzeichnet
1:1
Name
String
(3)
Person
wird
gekenn-
zeichnet
1:1
Iden-
tifikati
on
kennzeichnet
1:1
Personal-
ausweis
Identifiziert
durch
1:1
Numer
iert
identifiziert
1:1
Erfassungs-
Nummer
Integer

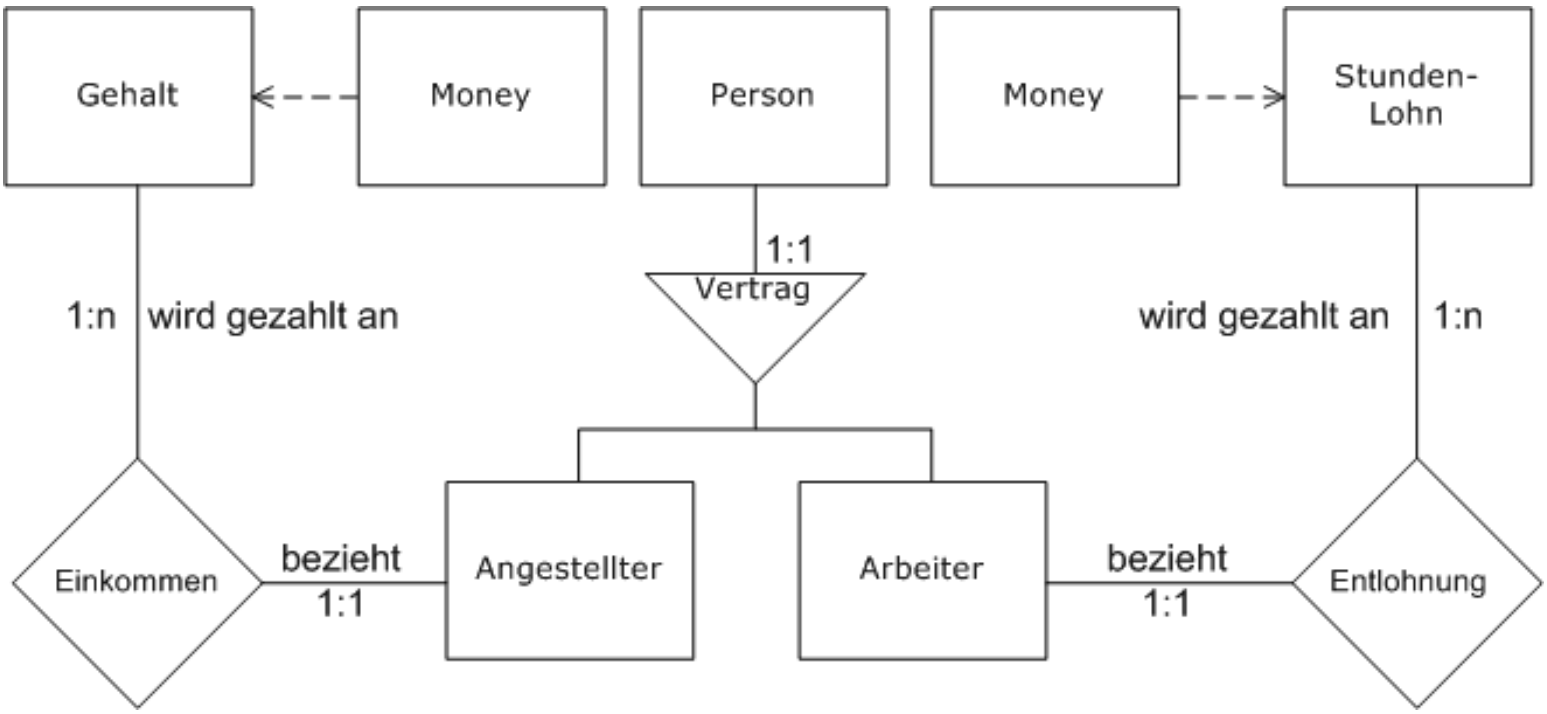

Entitätstyp1
i1:j1
Rolle1
Assoziationstyp1
i2:j2
Rolle2
Entitätstyp2
Entität1
(Entitätstyp1)
Rolle1
Assoziation1
(Assoziationstyp1)
Rolle2
Entität2
(Entitätstyp2)

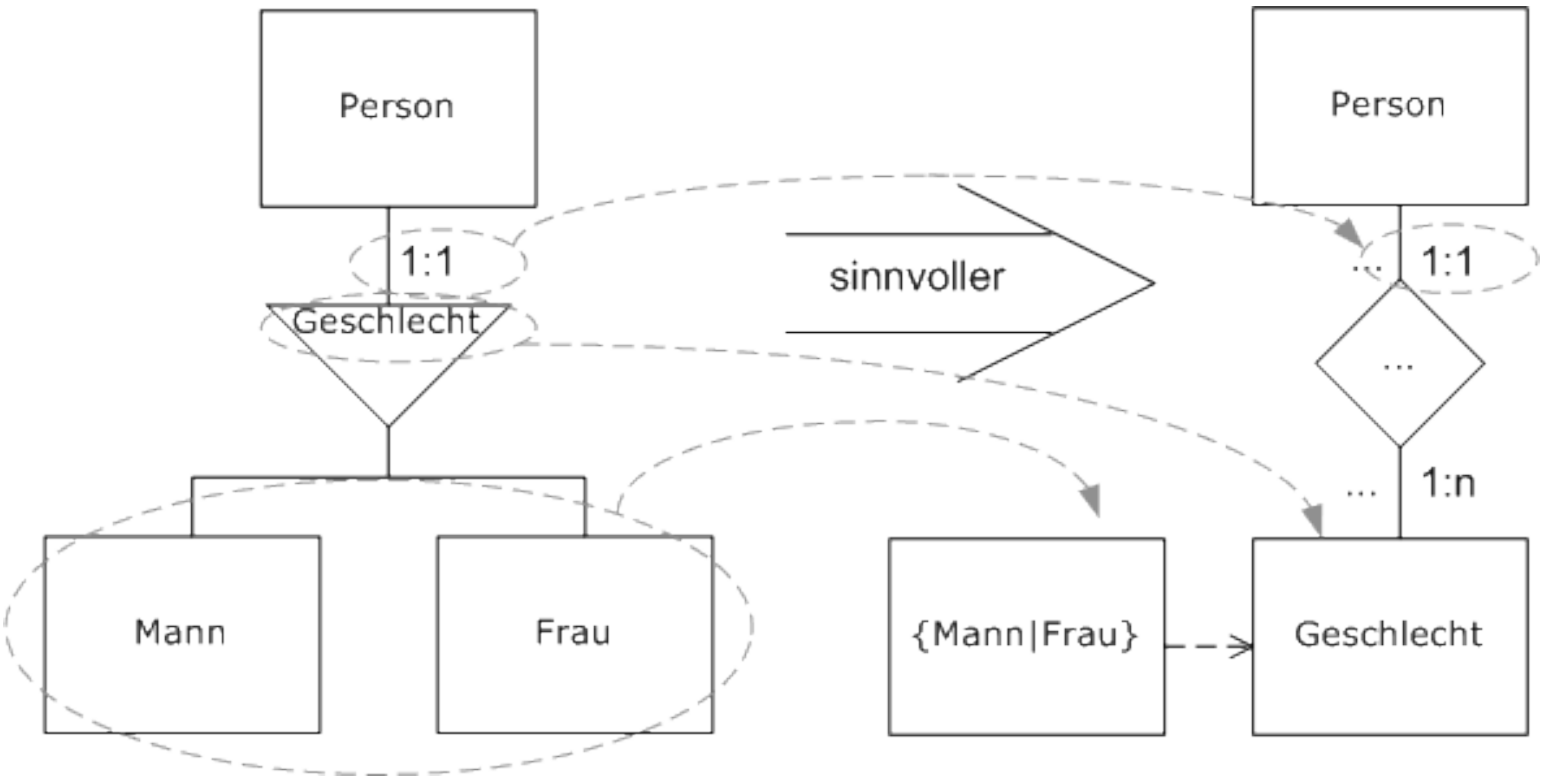

Person
0:n
arbeitet für
Arbeits-
verhältnis
1:n
beschäftigt
Firma

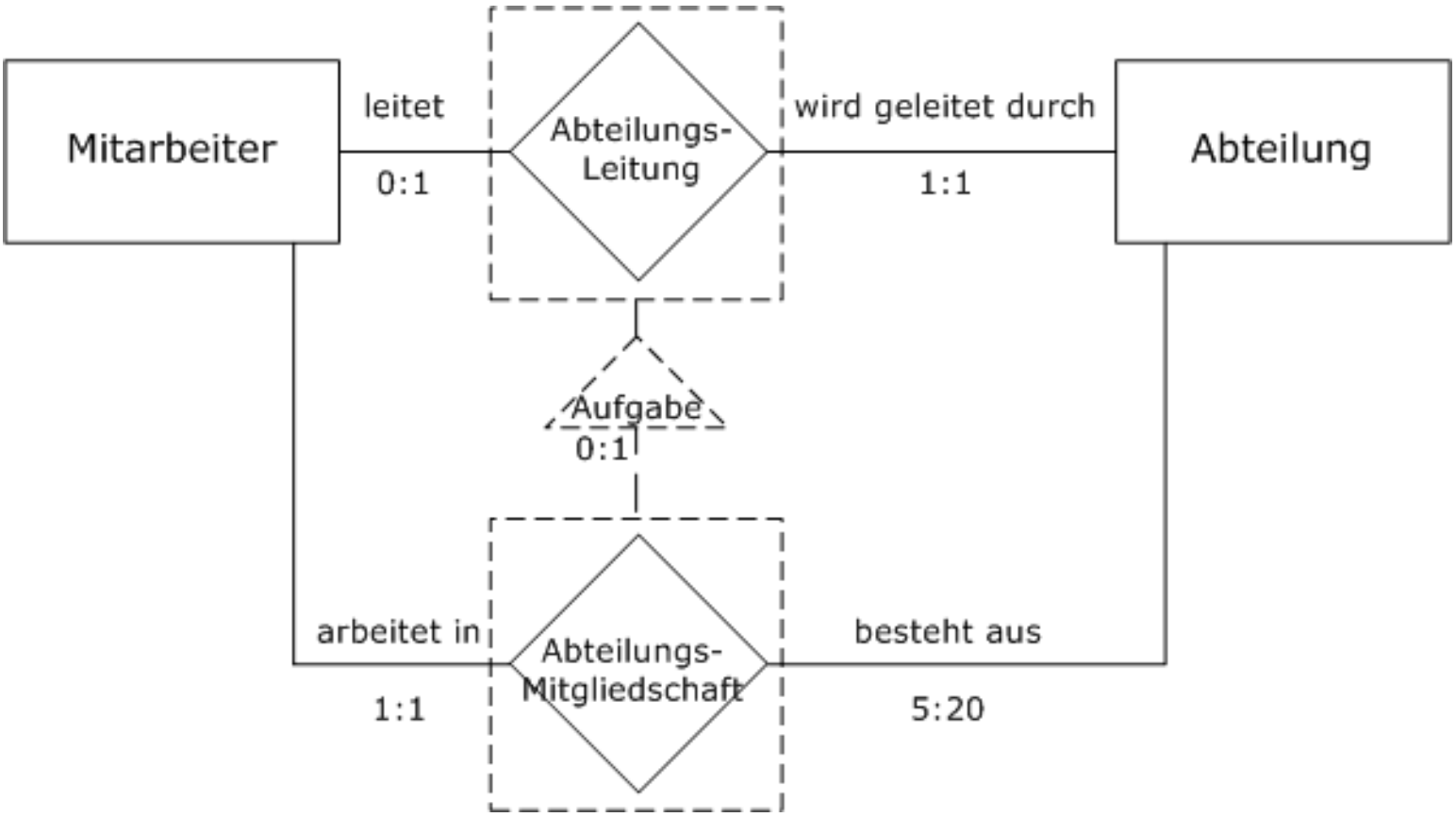

doziert
1:n
Abhal-
tung
wird gehalten
1:n
Referent
bevorzugt
0:n
Lieblings-
kurs
wird bevorzugt
0:n
Kurs
kompetent
1:n
Be-
fähigung
kann gehalten werden von
1:n

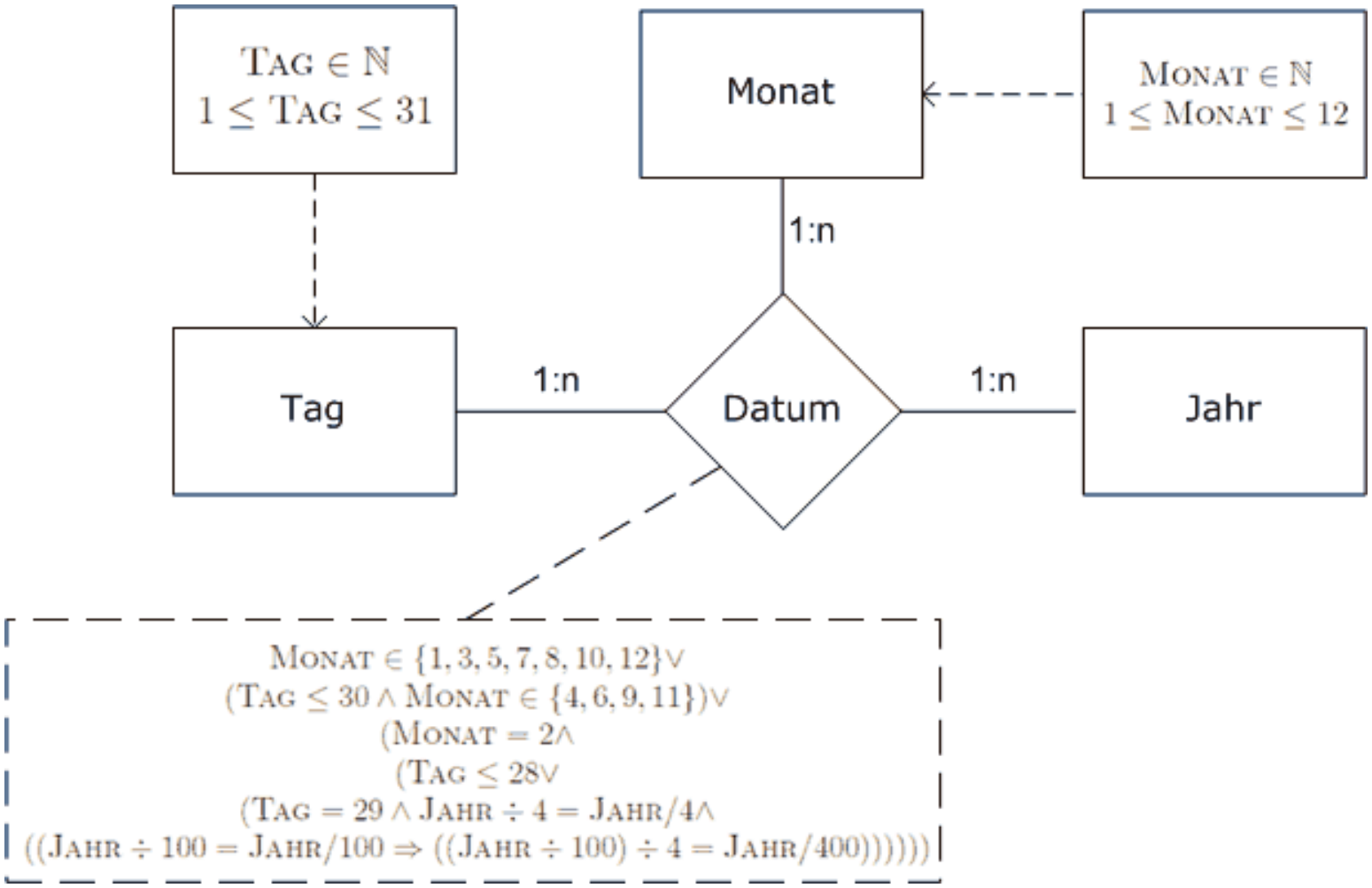

Mitarbeiter
arbeitet für
1:n
Arbeits-
verhältnis
beschäftigt
1:n
Firma
bezieht
1:1
Gehalts-
zuordnung
zugeordnet
1:n
Gehalt
Real

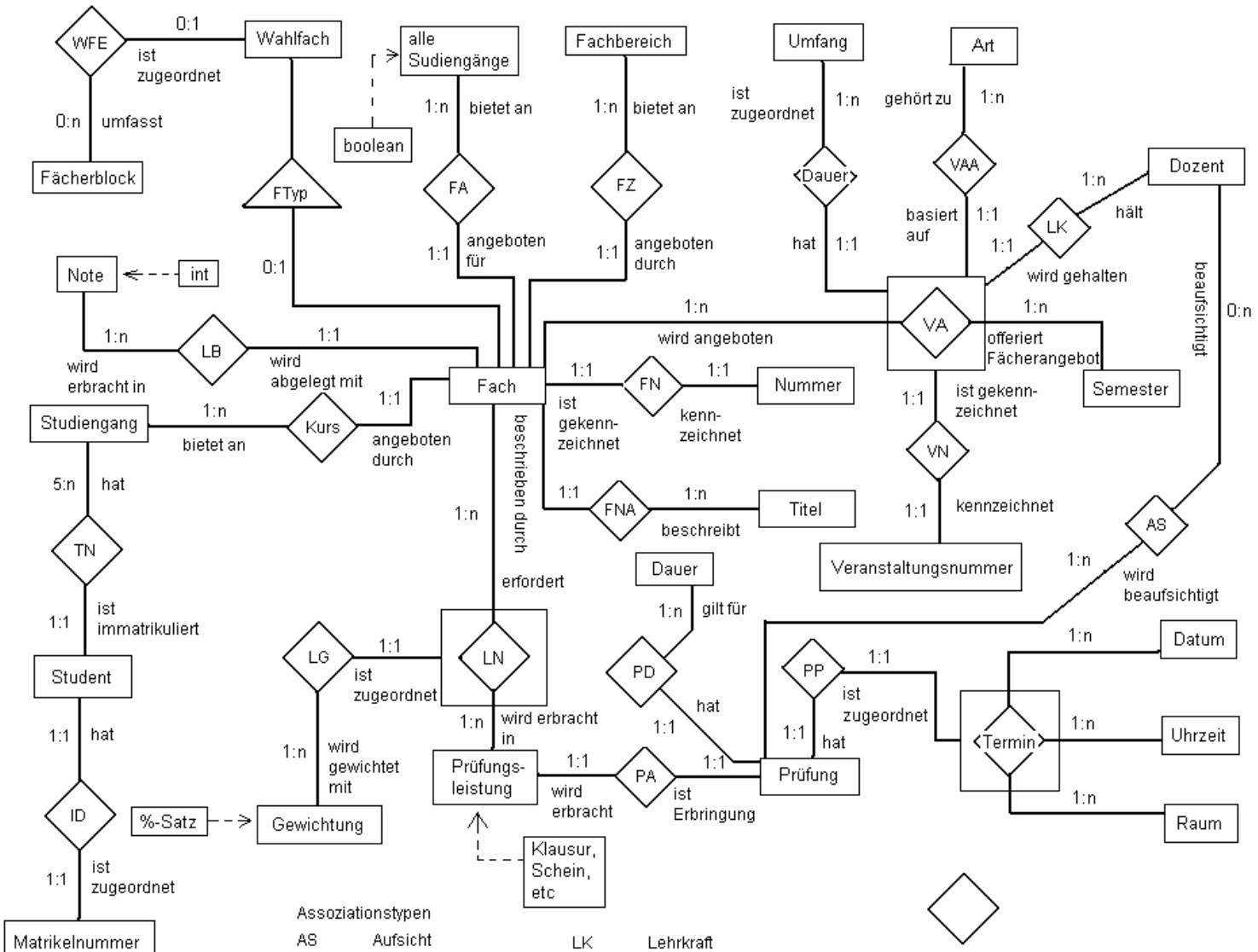

String
String
Straßenname
umfaßt
1:n
Straße
liegt in
1:n
Hausnummer
gehört zu
1:n
Adresse
Anschrift von
0:n
Anschrift
wohnt an
1:n
Person
beherbergt
1:n
Ortsname
besitzt
1:n
Ort
zugeordnet
1:n
PLZ
String
Integer


Gehalt
Money
Person
Money
Stunden-
Lohn
1:1
Vertrag
1:n
wird gezahlt an
wird gezahlt an
1:n
Einkommen
bezieht
1:1
Angestellter
Arbeiter
bezieht
1:1
Entlohnung


Person
1:1
Geschlecht
Mann
Frau
sinnvoller
Person
...
1:1
...
...
1:n
{Mann|Frau}
Geschlecht

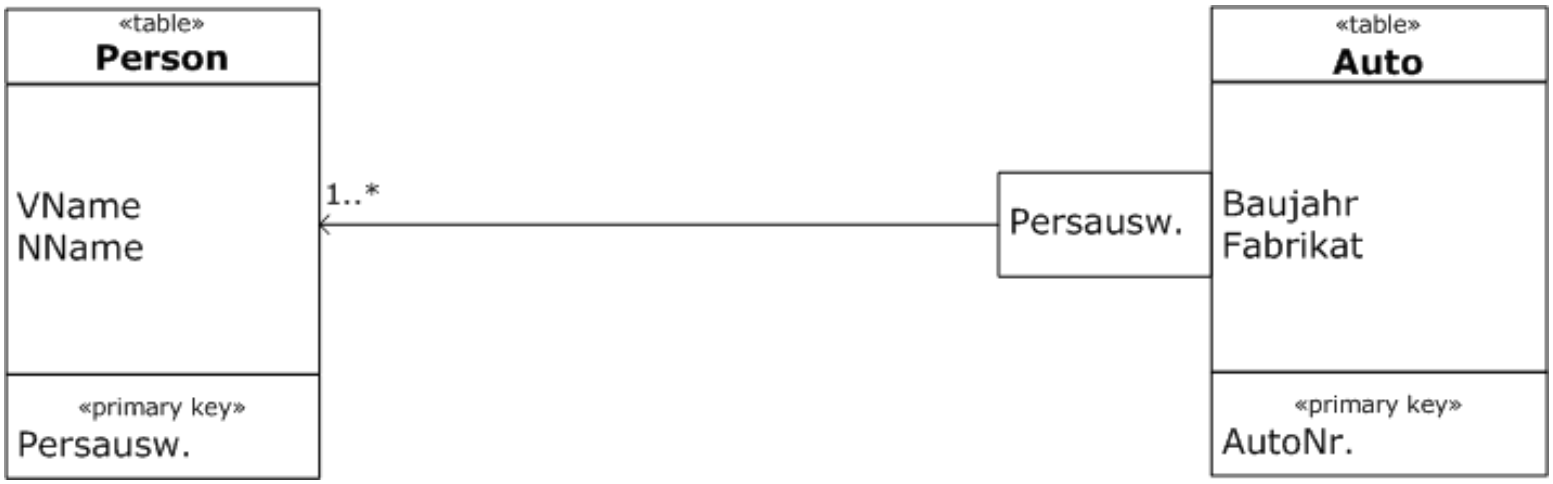

Mitarbeiter
leitet
0:1
Abteilungs-
Leitung
wird geleitet durch
1:1
Abteilung
Aufgabe
0:1
arbeitet in
1:1
Abteilungs-
Mitgliedschaft
besteht aus
5:20

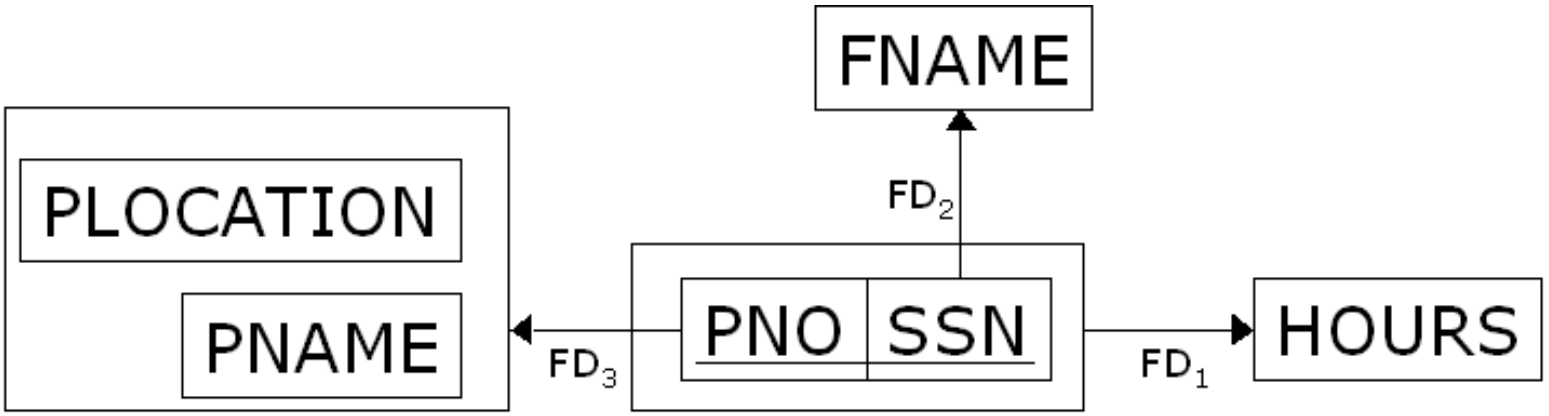

Tag ∈ ℕ
1 ≤ Tag ≤ 31
Monat
Monat ∈ ℕ
1 ≤ Monat ≤ 12
1:n
Tag
1:n
Datum
1:n
Jahr
Monat ∈ {1, 3, 5, 7, 8, 10, 12}∨
(Tag ≤ 30 ∧ Monat ∈ {4, 6, 9, 11})∨
(Monat = 2∧
(Tag ≤ 28∨
(Tag = 29 ∧ Jahr ÷ 4 = Jahr/4∧
((Jahr ÷ 100 = Jahr/100 ⇒ ((Jahr ÷ 100) ÷ 4 = Jahr/400))))))

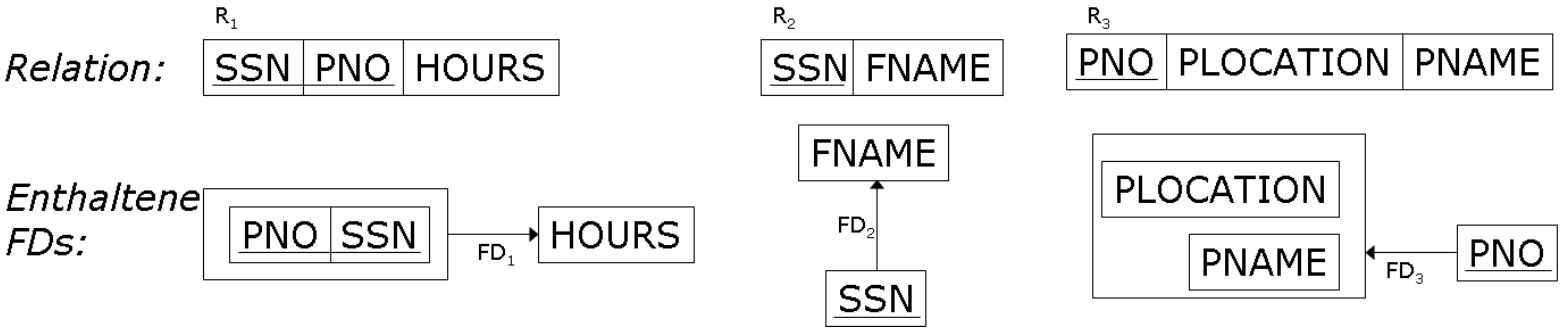


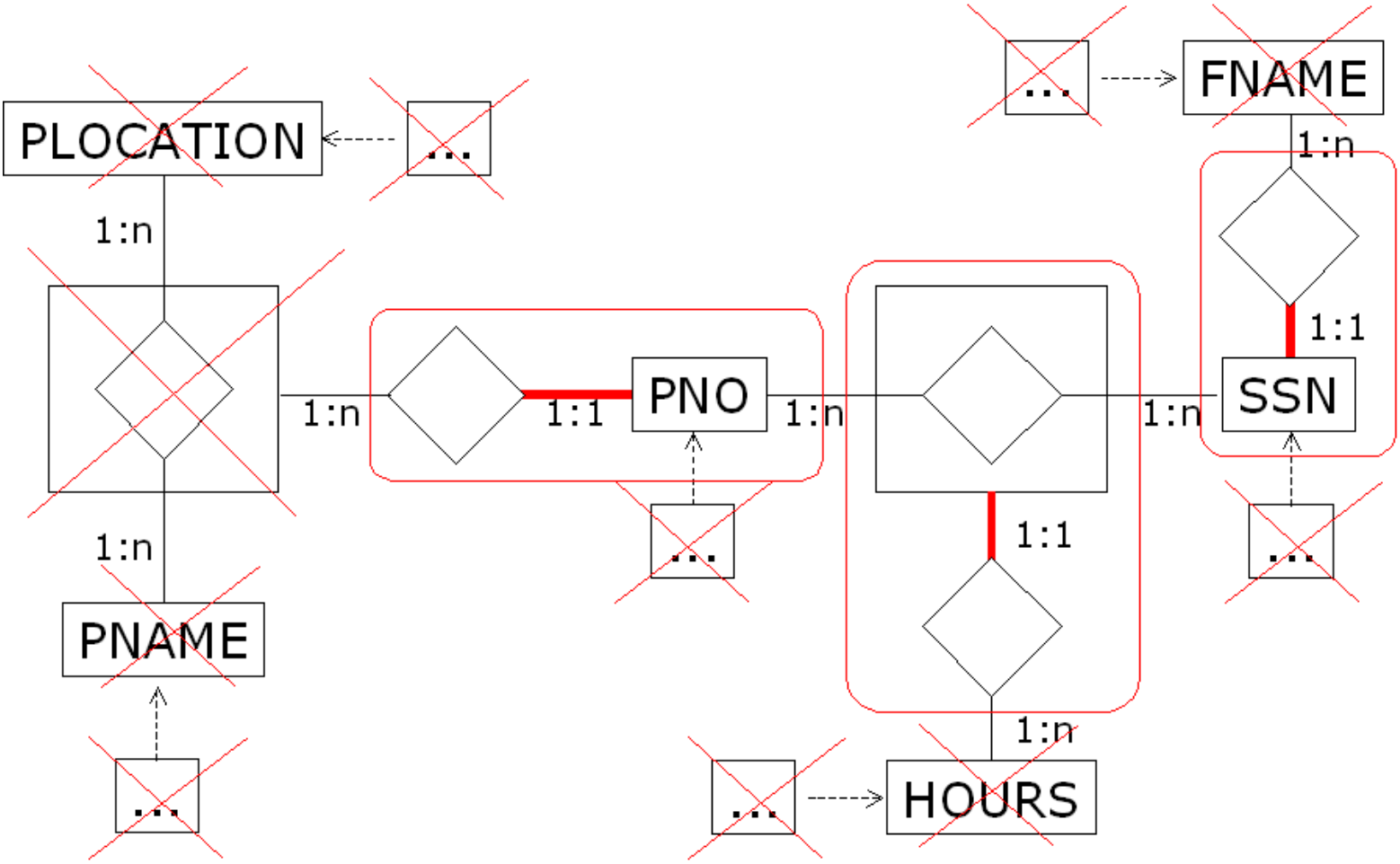
Assoziationstypen

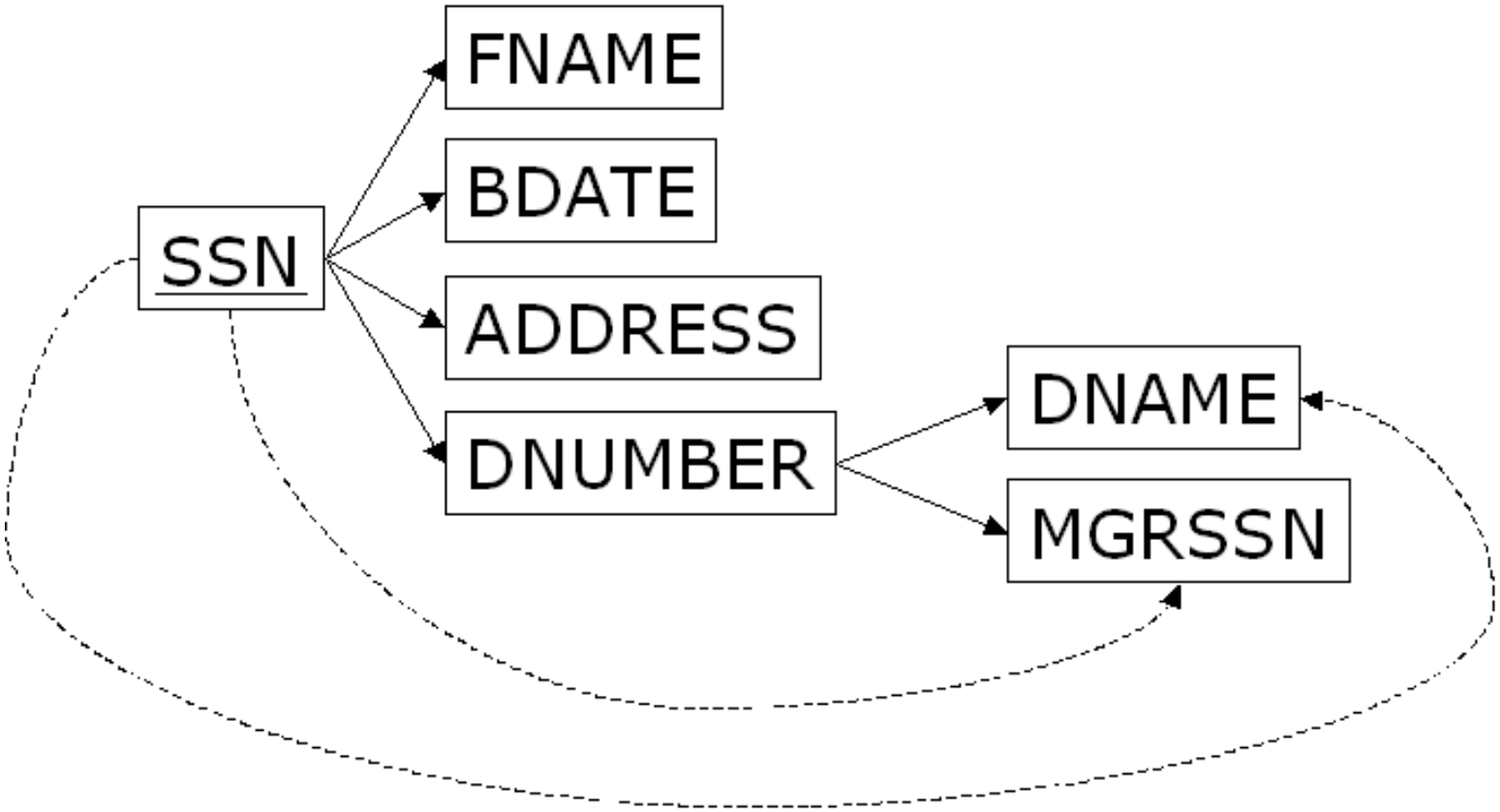
AS	Aufsicht	LK	Lehrkraft
FA	Fächerangebot	LN	Leistungsnachweis
FN	Fachnummer	PA	Prüfungsart
FNA	Fachname	PD	Prüfungsdauer
Ftyp	Fachtyp	PP	Prüfungsplanung
FZ	Fächerzuordnung	TN	Teilnahme
ID	Identifikation	VA	Veranstaltung
LB	Leistungsbewertung	VAA	Veranstaltungsart
LG	Leistungsgewichtung	VN	Veranstaltungsnummer
		WFE	Wahlfacheinteilung

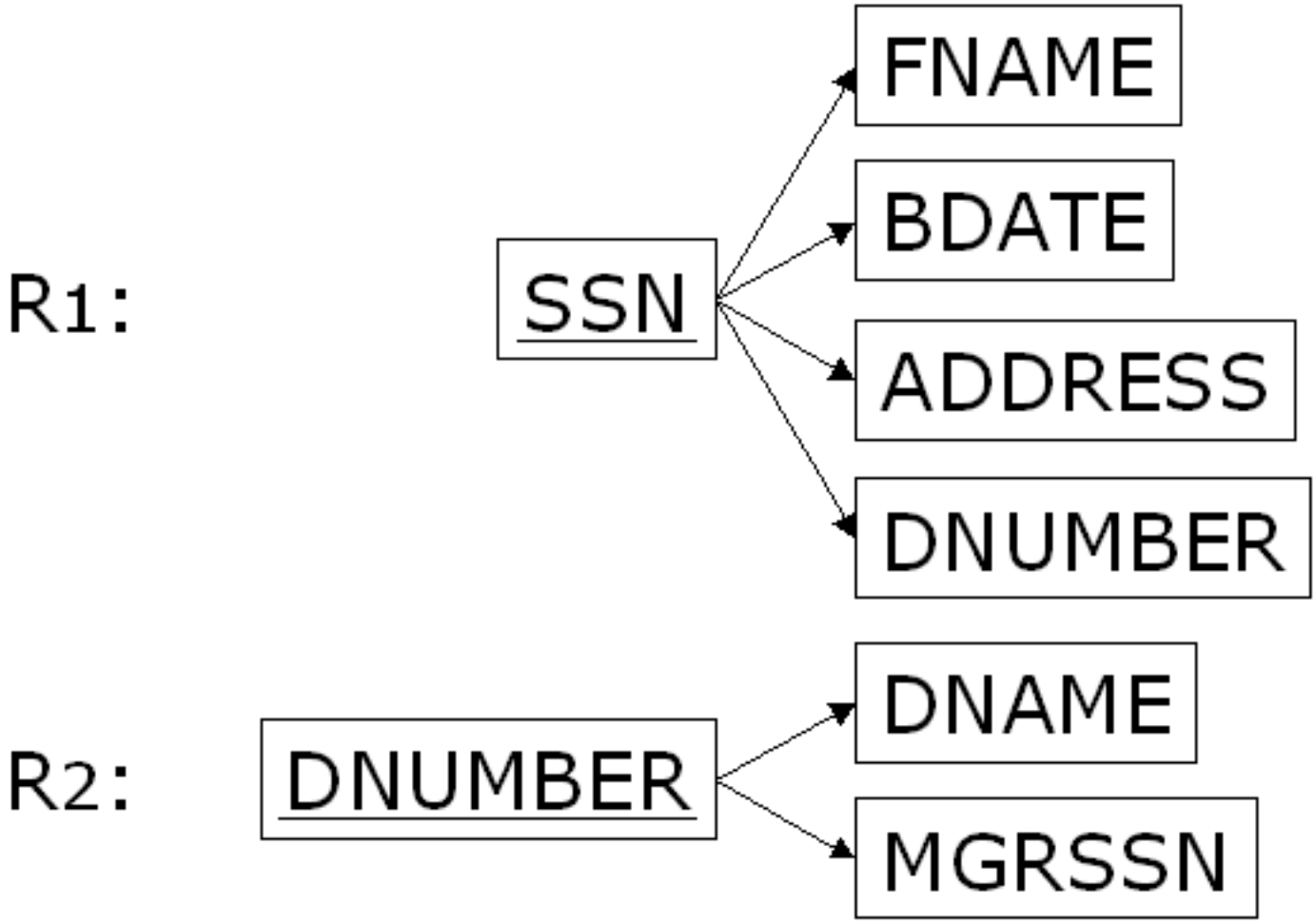

«table»
Person
VName
NName
«primary key»
Persausw.
1..*
Persausw.
«table»
Auto
Baujahr
Fabrikat
«primary key»
AutoNr.

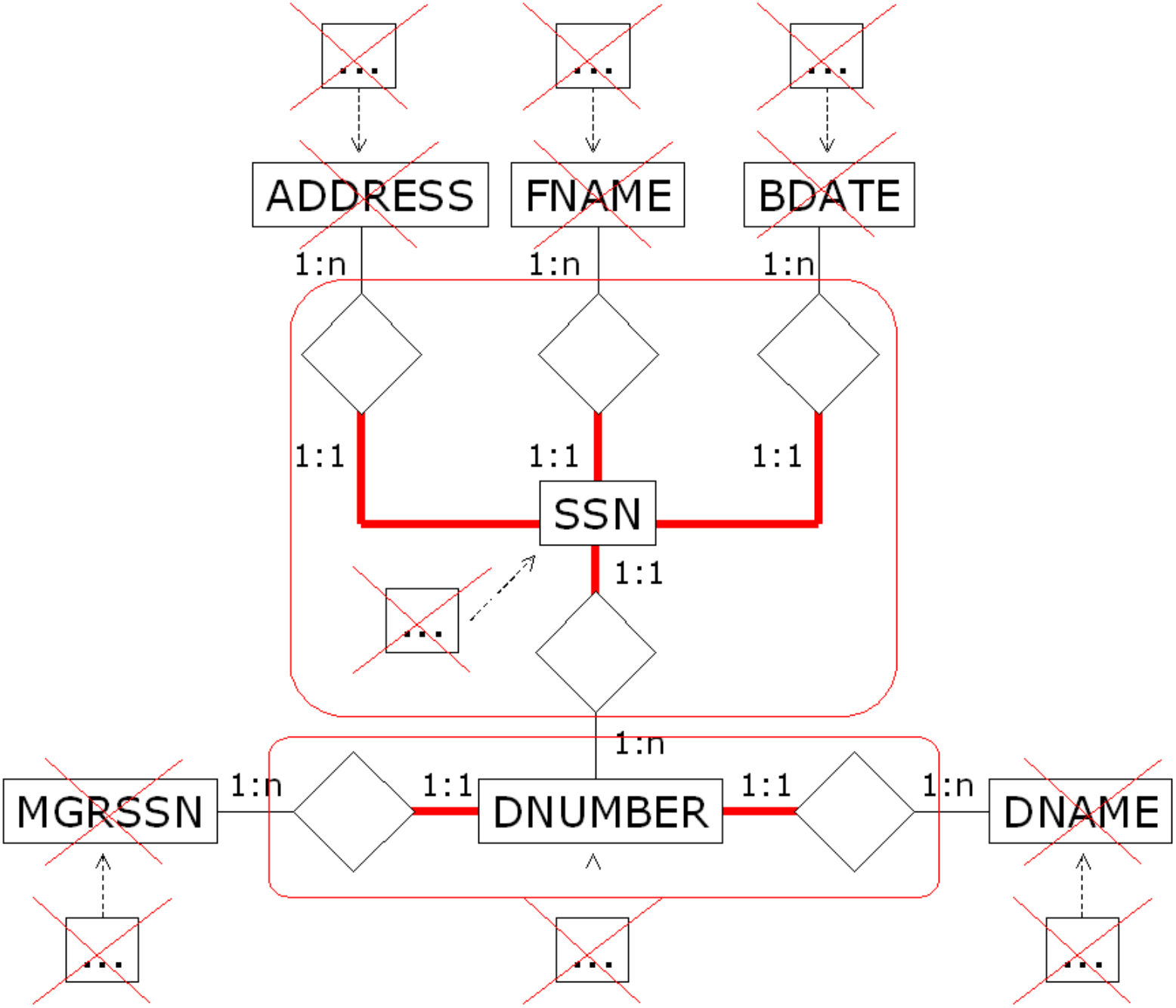

FNAME
PLOCATION
PNAME
FD2
PNO
SSN
HOURS
FD3
FD1

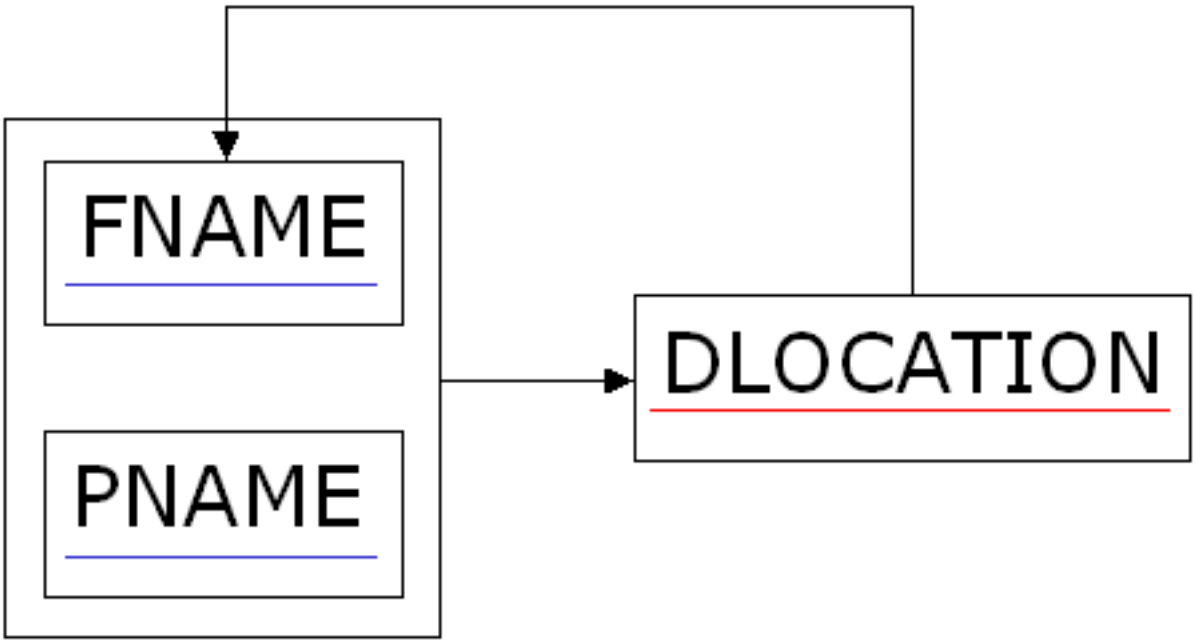

R1
Relation:
SSN
PNO
HOURS
R2
SSN
FNAME
R3
PNO
PLOCATION
PNAME
Enthaltene FDs:
PNO
SSN
FD1
HOURS
FNAME
FD2
SSN
PLOCATION
PNAME
FD3
PNO

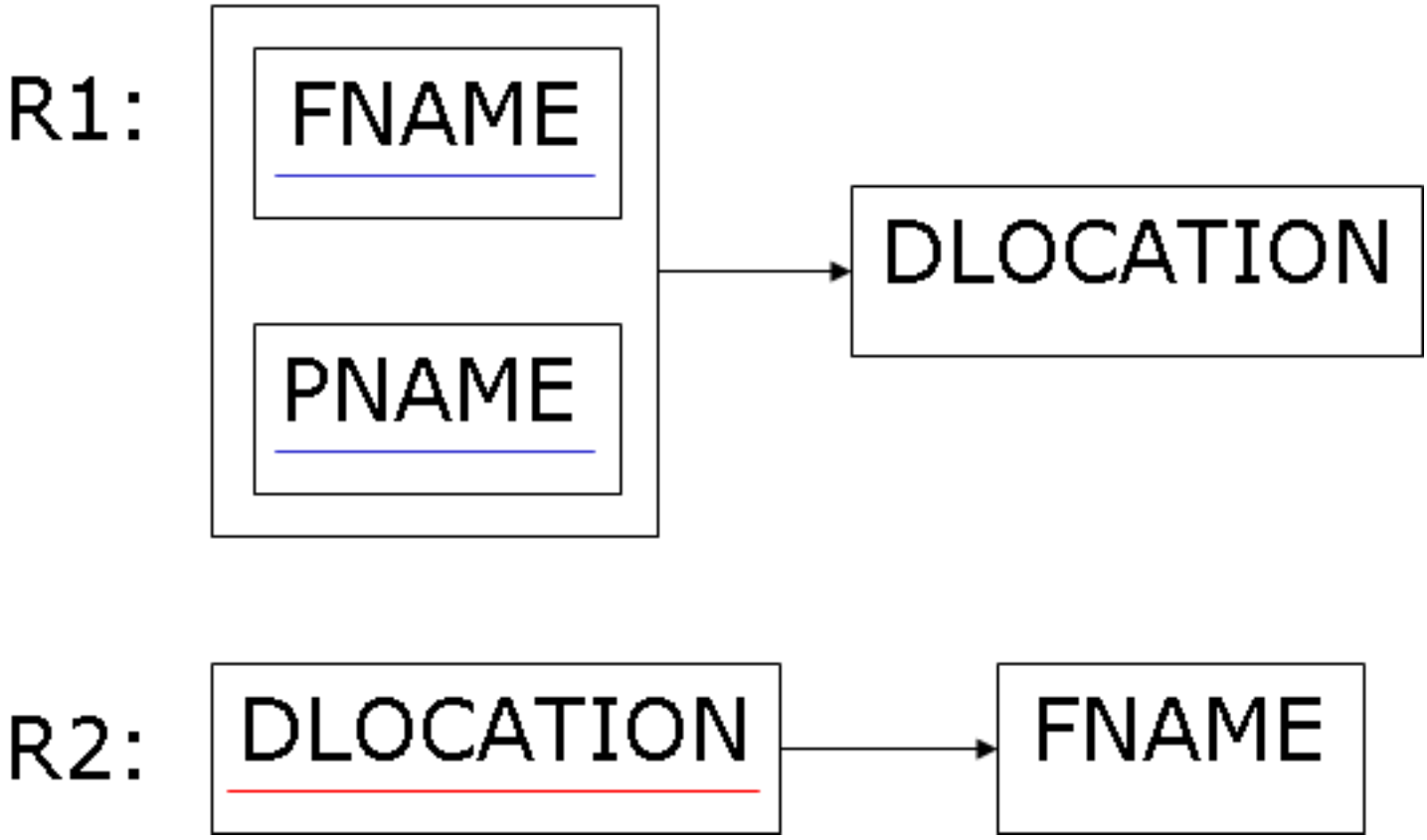

PLOCATION
...
1:n
1:n
PNAME
...
1:n
PNO
1:1
...
1:n
1:n
1:1
...
HOURS
...
FNAME
1:n
1:1
SSN
...


FNAME
BDATE
SSN
ADDRESS
DNAME
DNUMBER
MGRSSN

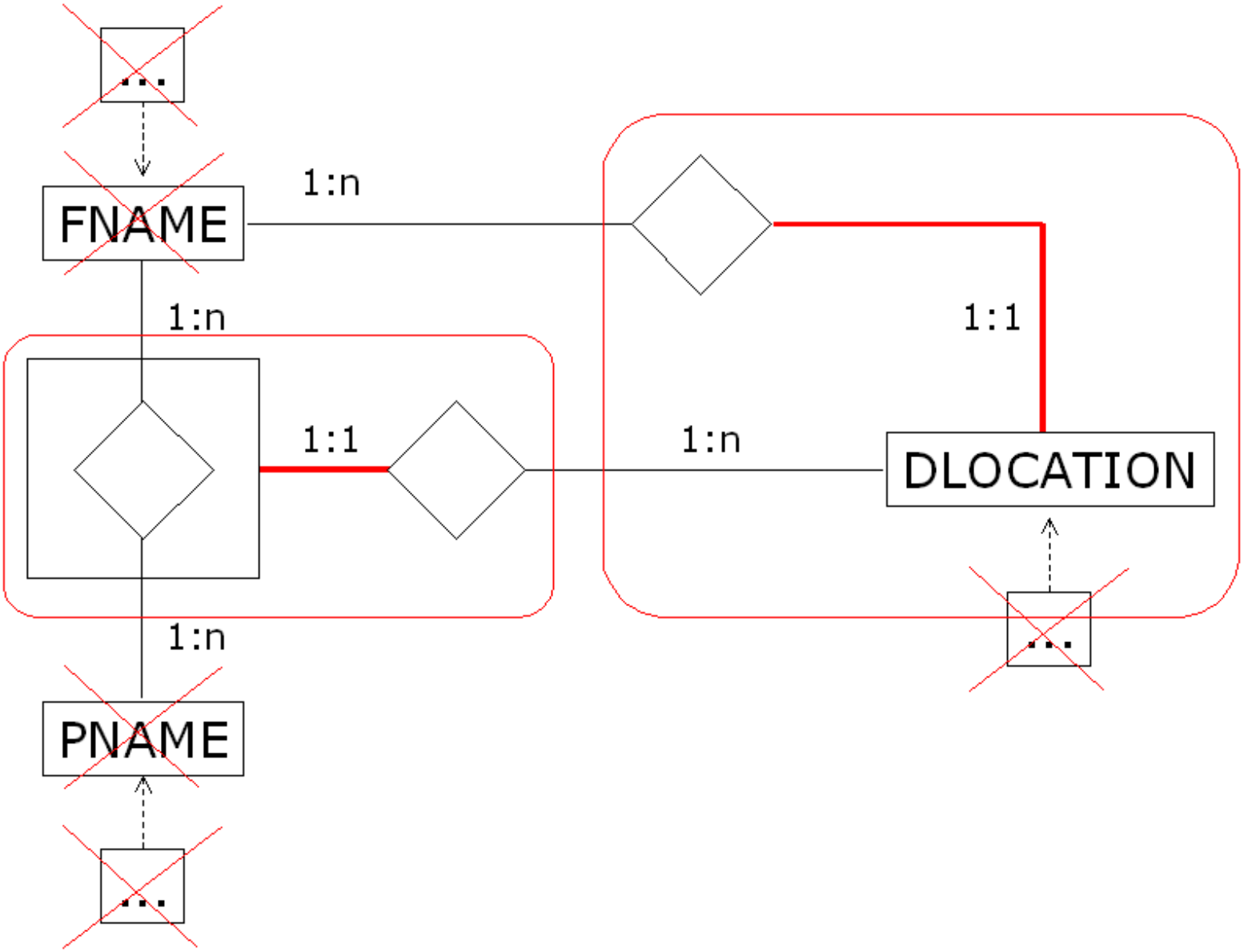

R1:
SSN
FNAME
BDATE
ADDRESS
DNUMBER
R2:
DNUMBER
DNAME
MGRSSN

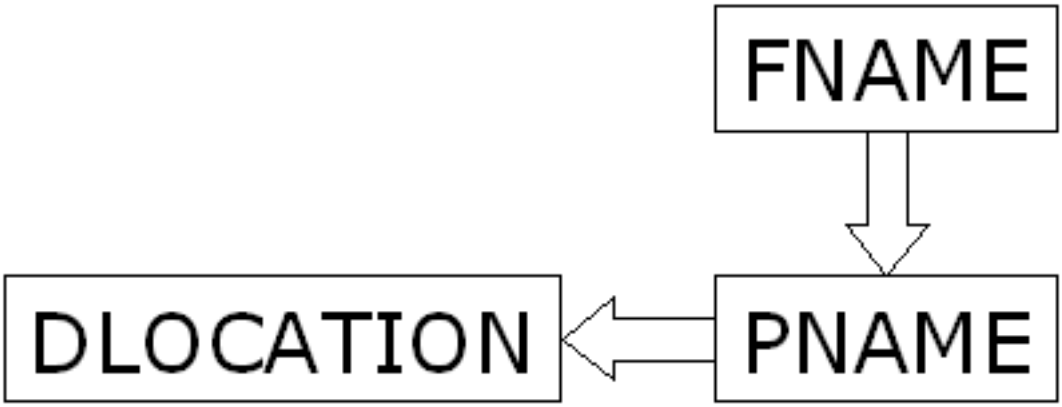

ADDRESS
FNAME
BDATE
1:n
1:n
1:n
1:1
1:1
1:1
SSN
1:1
1:n
MGRSSN
1:n
1:1
DNUMBER
1:1
1:n
DNAME

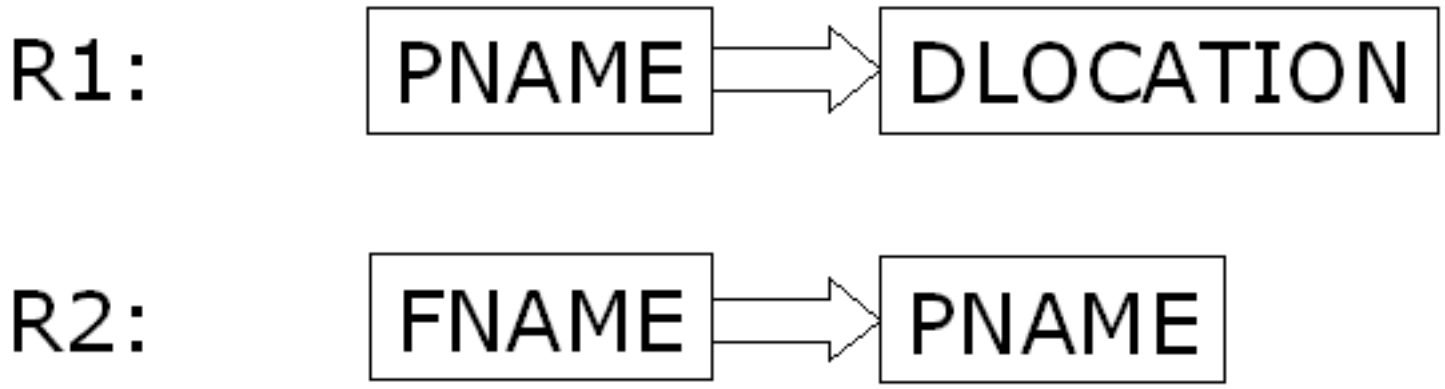

FNAME
PNAME
DLOCATION

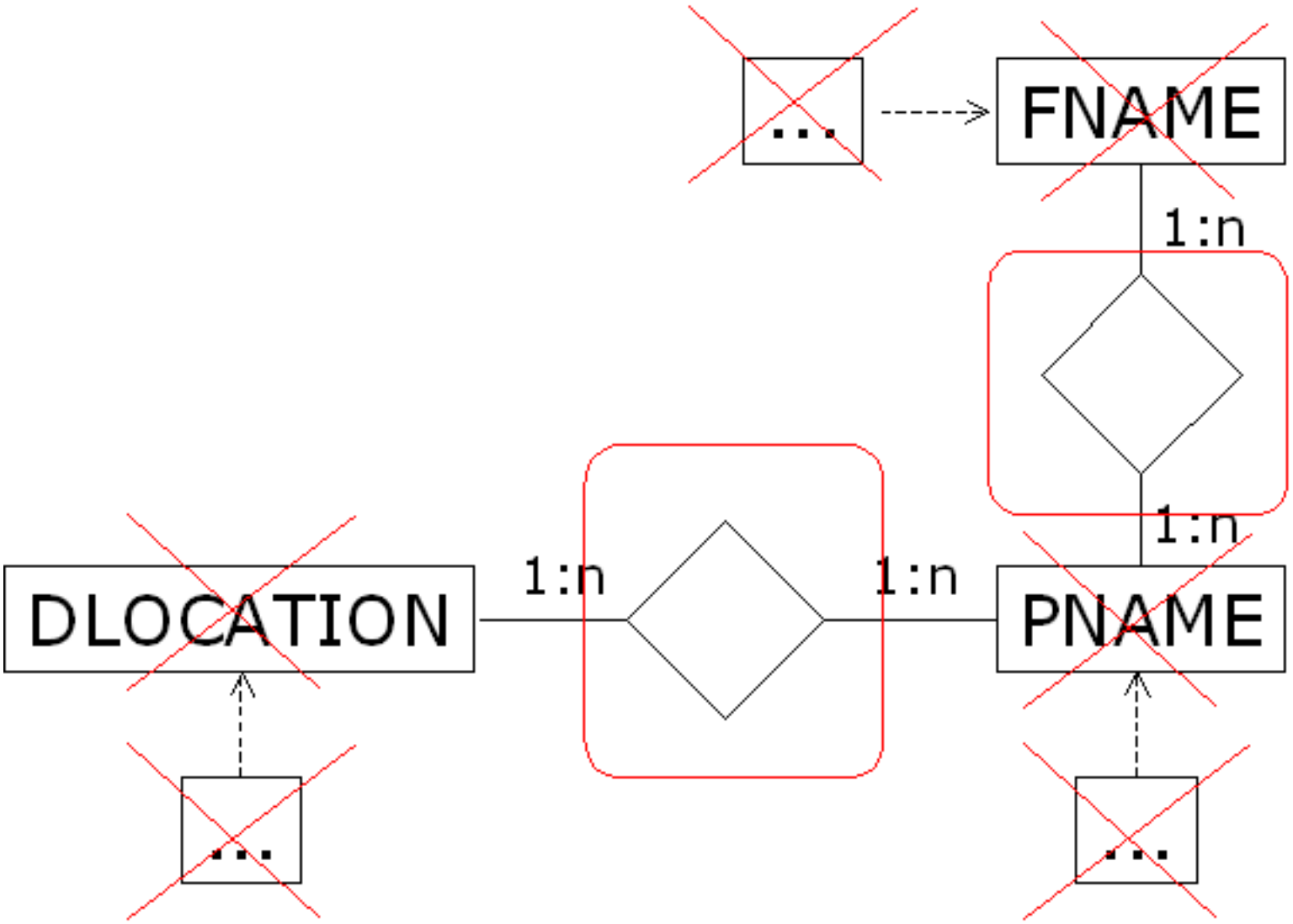

R1:
FNAME
PNAME
DLOCATION

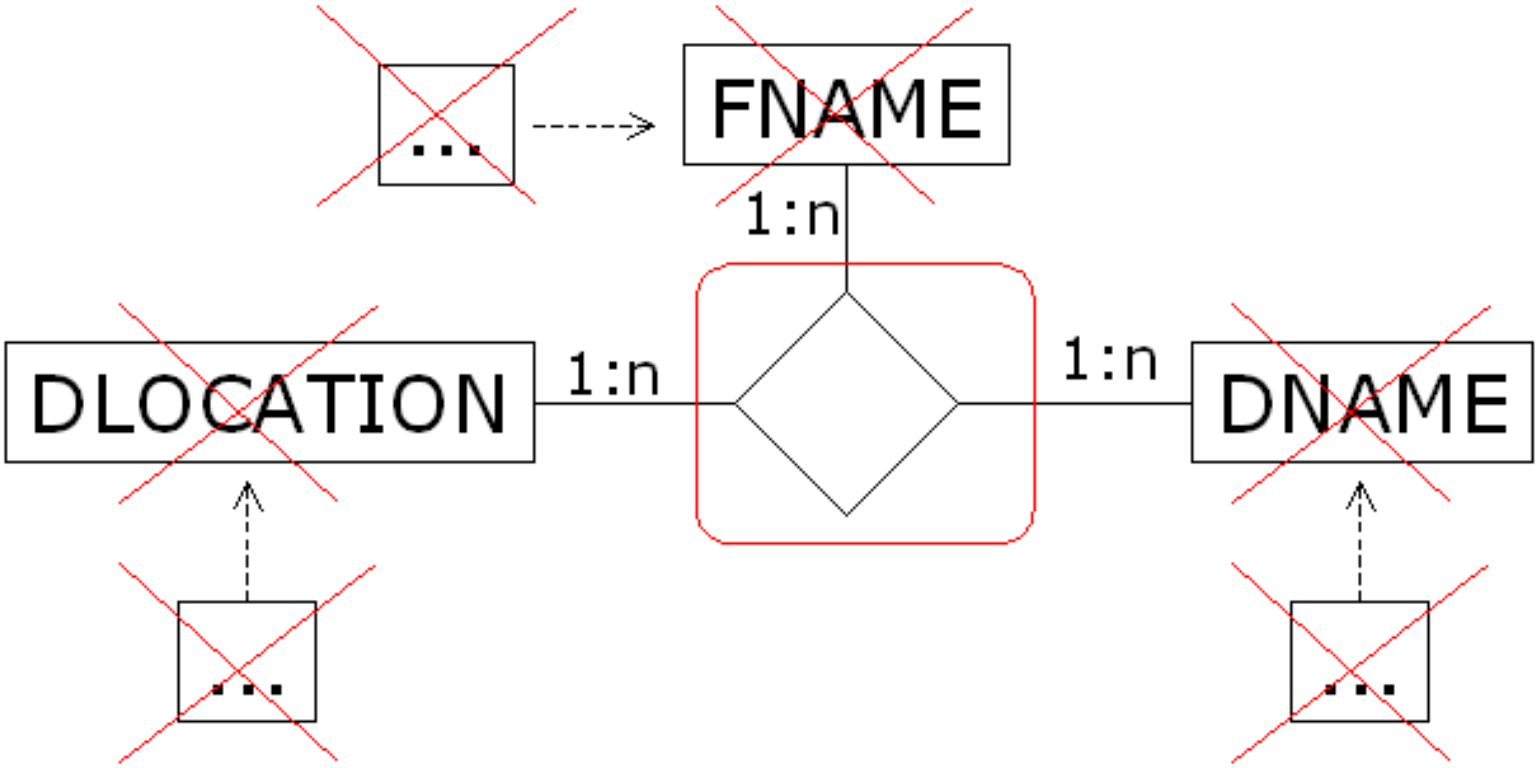
R2:
DLOCATION
FNAME


FNAME
1:n
1:n
1:1
1:n
1:1
1:n
DLOCATION
PNAME


FNAME
DLOCATION
PNAME


R1:
PNAME
DLOCATION
R2:
FNAME
PNAME


...
FNAME
1:n
1:n
DLOCATION
1:n
1:n
PNAME
...
...


...
FNAME
1:n
DLOCATION
1:n
1:n
DNAME
...
...

November 1995

A Relational Model of Data for Large Shared Data Banks

E. F. Codd

Reprinted from *Communications of the ACM*, Vol. 13, No. 6, June 1970, pp. 377-387.

MAC / 95-Nov-1

Quarter 2, 2004 Vol. 9, Issue 2

Six Security Secrets Attackers Don't Want You to Know

By Aaron C. Newman

How secure is your data? Looking at your information management resources through a would-be intruder's eyes can help you find (and fix) vulnerabilities.

When E. F. Codd developed his relational data model in 1970, the business world was a different place. Almost 35 years after his seminal work appeared, RDBMSs that sprung from Codd's ideas are the standard for storing corporate information. And, with government and industry regulations dictating what kinds of information companies have to store, manage, and audit (and for how long), protecting this information is more important than ever.

Unfortunately, it's also more challenging. Even in 1985, when Dr. Codd published 12 guidelines for RDBMSs, there was little concern for data security. In those days, gaining access to a database was so difficult that advanced security features on the database were irrelevant.

Today, RDBMSs carry the lifeblood of every organization. Note the use of the plural: Organizations now have many databases that are decentralized in terms of use and security controls. E-business demands that data access be extended to customers, partners, suppliers, and other parties who were rarely considered in the early data management days. With all this availability — not to mention pressure from an array of government and industry regulations (see the sidebar, "Security and Compliance") — the need to control exactly who can access or modify data is becoming paramount.

DB2 Universal Database (UDB) is one of the most commonly used RDBMSs on the market; companies use it to store information for both internal and external (Internet access) purposes. E-business has driven the popularity of DB2 as a back end for Web applications. But the Internet is also the source of many (though not all) attempts to breach data security. Understanding the nature of potential threats so that you can protect against them is key to surviving in the Internet age.

For that reason, I'm going to show you how an attacker looks at your database. If you're going to protect your information assets, you must understand the mindset of an attacker.

First, I want to clear up a common misconception. People often associate some level of security with an installed software product. Some products have reputations as inherently insecure, others have reputations as inherently secure. DB2 has a reputation for providing solid security, but it would be a mistake to conclude that information in DB2 is automatically secure.

Security depends more on the administration and setup of the system than on the underlying software. Of course, if the underlying software lacks basic features, you won't be able to fix that. Attackers know, however, that the security features in even the most secure software are seldom put in place, leaving the software vulnerable to a wide range of threats.

Security is a process, not a product. You don't just install DB2 and forget about it. DB2 security is an ongoing process that involves setting up the database securely, setting up procedures to ensure that DB2 stays secure, and continually reviewing procedures to make sure no holes have opened up that could expose critical data assets.

Here are six common security vulnerabilities that most attackers are all too familiar with — and that every DB2 DBA should know by heart. Although these examples are drawn from DB2 for Linux, Unix, and Windows, the concepts hold true for mainframe environments as well.

1. DEFAULT DB2 USERNAME AND PASSWORDS

An attacker's favorite way to break into any database is to take advantage of the simplest holes; default usernames and passwords are among the easiest to exploit. When software is installed, accounts are typically created and assigned default passwords so that the person installing the software can immediately start using the program. A simple, but common, mistake is to not immediately reset the default password. Sometimes administrators intend to do so later (and never do). Or, and this is just as likely, the administrator isn't aware that there are multiple default usernames and passwords to reset. If you're setting up a single database, you might not make this mistake. However, if you're installing databases day in and day out, it's much easier to forget occasionally.

This oversight plays right into the attackers' hands. Many attackers are equipped with lists of default usernames and passwords collected from various sources, including Web sites dedicated to aggregating known default password lists.

You can change passwords by the native mechanisms for Unix or Windows. However, DB2 also provides a method to change a password through a DB2 client. To change the password, use the following command:

```
CONNECT TO [database] USER [userid]
USING [password] NEW [new_password]
CONFIRM [new_password]
```

You can also use the Password Change dialog of the DB2 Client Configuration Assistant.

Third-party applications (such as SAP or PeopleSoft) are another possible threat. They may also include unchanged (and, therefore, vulnerable) default usernames and passwords. In order to protect against these defaults, eliminate any default passwords in third-party applications that are accessing your data.

2. CONFIGURING AUTHENTICATION

Attackers always check the authentication mechanism configured for DB2, so pay attention to the authentication type you've configured. The authentication type determines the mechanism used to identify users and defines a combination of where and how authentication occurs.

Which authentication type you can choose is limited based on the environment and the operating system. Authentication type is configured at both the client and the server. For the server, authentication is defined in the database manager configuration file associated with an instance. All databases under the instance share that configuration file, so an authentication method configured for a particular instance will apply to all databases within the instance and all users within the database.

DB2 currently supports the following authentication types:

- SERVER
- SERVER_ENCRYPT
- CLIENT
- DCE
- DCE_SERVER_ENCRYPT
- DCS
- DCS_ENCRYPT
- KERBEROS (introduced in 7.x for Windows 2000 only)
- KRB_SERVER_ENCRYPT (introduced in 7.x for Windows 2000 only).

If the authentication type is set to `CLIENT`, two other parameters in the database manager configuration file are used: `TRUST_ALLCLNTS` and `TRUST_CLNTAUTH`.

Make sure you select a secure mechanism for authentication. Don't rely on client authentication because any computer can hook up to the network - you can't assume

that these clients are secure. Even configuring DB2 to trust only specific clients isn't foolproof; clients can be "spoofed."

Client credentials should always be encrypted before being sent to the server. This step is important to prevent someone from "sniffing" credentials as they go over the network.

Select one of the secure authentication modes (such as `SERVER_ENCRYPT`, `DCE_SERVER_ENCRYPT`, or `KRB_SERVER_ENCRYPT`). With any of the other authentication types, passwords are sent in clear text over the network.

You can update the authentication method for the instance using the Command Center or the Command Line Processor. Use the following commands to update the configuration:

```
UPDATE DBM CFG USING AUTHENTICATION [new method]
```

For the changes to take effect, you must stop and then restart the instance.

You can modify the `AUTHENTICATION` configuration parameter using the Control Center by following these steps:

1. Open up the DB2 Control Center.

2. Find and select the instance whose configuration you want to modify.

3. Choose the Selected/Configure ... menu item.

4. Choose the Administration tab.

5. Select the Authentication value.

6. Under the Value section, set the new value.

7. Choose OK.

8. Restart the server.

You must be a member of the SYSADM group to update the database manager configuration file. For more information on authentication, read "The Database Security Blanket," by Paul Zikopoulos and "Selecting an Authentication Method for Your Server" in the Administration Guide (see Resources).

Resources

"The Database Security Blanket," Quarter 1, 2001

"Selecting an Authentication Method for Your Server," DB2 Administration Guide

DB2 Technical Support (FixPaks)

Application Security Inc.

Security Focus

3. DATABASE PRIVILEGES

Although most people think of attackers as unknown, remote villains, the truth is that an attack is much more likely to come from the person down the hall. Attacks can come from system users trying to see data they aren't authorized to see or from someone who exploited a more privileged account. In either case, the attacker will first try to gain elevated privileges.

A popular strategy is to collect a list of accounts that can be probed for weak passwords. An attacker gathers a complete list of system users and immediately starts looking for the following weaknesses:

1. Blank passwords

2. Easy to guess passwords (such as the username)

3. Passwords that are words found in a dictionary

4. Default passwords that haven't been changed.

Cracking a password that falls into one of these categories is relatively simple. It typically takes nothing more than a PERL script that takes 10 minutes to write. On any system, there will be some users who fail to use strong passwords.

DB2 doesn't maintain password credentials within the database, relying instead on external sources to manage the user list. In a typical configuration, DB2 accounts are operating system accounts; authentication is performed under the operating system. DB2 doesn't have one specific table in which all accounts are listed. It has several, including:

- SYSIBM.DBAUTH
- SYSIBM.TABAUTH
- SYSIBM.INDEXAUTH
- SYSIBM.COLAUTH
- SYSIBM.SCHEMAAUTH
- SYSIBM.PASSTHRUAUTH.

Make sure you remove all permissions granted to `PUBLIC` on these tables. Carefully review the `SYSADM` group to ensure that it includes only authorized users, that the users are using strong passwords, and that proper auditing is enabled.

4. FIXPAKS

Security holes are an inevitable side effect of complex software. All but the simplest software contains security holes. Attackers prey on databases whose administrators haven't kept up with the appropriate patches.

IBM regularly releases FixPaks that provide various enhancements, including security fixes. Staying up to date on the latest FixPak minimizes vulnerability. DB2 FixPaks are cumulative; each new one includes all previous fixes. FixPaks can be applied to an original installation or to one in which previous FixPaks have been applied.

The most current FixPaks (at press time) are:

- V.6.1: Fix Pak 11
- V.7.1: Upgrade to V7.2
- V.7.2: Fix Pak 11
- V.8.1: Fix Pak 5.

You can download FixPaks and check for new ones at on the DB2 Technical Support site (see Resources).

5. BUFFER OVERFLOWS

Buffer overflows, which allow an attacker to hijack or crash a program, illustrate why staying up to date with FixPaks is so important.

When buffer overflows crash the program, it's called a denial of service attack. These attacks cause the database server to fail or to stop responding to requests.

Denial of service attacks, which can result from sending malformed packets or information the database doesn't expect, are sometimes a form of vandalism by unsophisticated attackers. However, they can also be used in sophisticated spoofing attempts.

Of more concern is the kind of attack that results in an unauthorized user causing the application to perform an action it wasn't intended to perform. Sending a malicious packet can cause a buffer to overflow and allow the attacker to hijack program execution. This kind of attack occurs with buffer overflows and format string vulnerabilities (exploits that alter the flow of a program by passing malicious values to format specifiers in functions such as printf).

Vulnerabilities within an application are usually the result of a programming error. Developers are often unaware of the security implications of using functions such as `gets()` or `printf()`. These functions don't perform any buffer checking or verify that the user isn't overwriting invalid memory. A lot of legacy code was written before these security issues became well known; as a result, these vulnerabilities are sprinkled throughout many applications.

The most dangerous vulnerabilities allow unauthenticated users to execute arbitrary commands. An exploitable buffer overflow bypasses passwords and authentication features (no matter how strong they are).

How does a buffer overflow occur? A computer program allocates areas of memory that are used to stored data. When multiple areas of memory are allocated, they're typically contiguous. When a program sets up 1,000 bytes of memory and then tries to write 1,001 bytes of data into that memory, the data runs over and writes into memory that's allocated for other purposes.

Attackers look for places in which a program can be forced to overwrite memory. If the memory overwrite occurs, the program's execution can actually be changed to run arbitrary commands.

To understand a stack-based buffer overflow, you have to understand how memory on the stack is allocated and how functions are executed.

The stack contains two components: an area of memory that's extremely quick to allocate and a pointer that keeps track of the top of the stack. If a program wants to save a piece of information, it can be pushed onto the stack (in other words, the program writes the data to the area of memory currently pointed to by the stack pointer and then increments the stack pointer so that it points to new memory). When the program wants to release an area of memory, the pointer is decreased to point back to the beginning of the free memory so this memory can be used for other purposes.

When machine code calls out to a function, the code must remember where the call originated so that it knows where to continue executing when the function completes. This "remembering" is accomplished by pushing the return address onto the stack.

Figure 1 shows a function pushing its return address onto the stack and allocating four bytes of memory as a variable of char username[4]. Those four bytes are then filled up with sys and a string terminator.

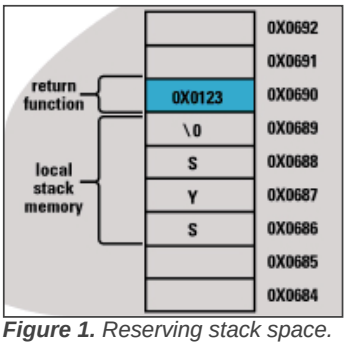


Figure 1. *Reserving stack space.*

So what happens if the program was tricked into writing five bytes of code into the four bytes allocated? The address of the return function will be overwritten with data indicated by the attacker.

In Figure 2, the execution of the program returns to the address 0X0689. The attacker can place machine code rather than an X into 0X689 and the target machine will execute this code.

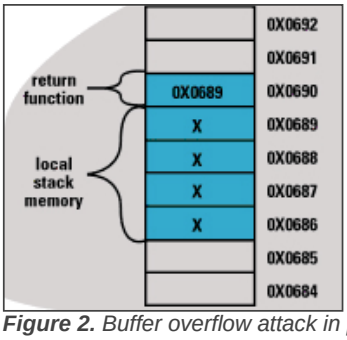


Figure 2. *Buffer overflow attack in progress.*

Buffer overflows happen most commonly with functions such as strcpy that copy one buffer into another. Imagine a DB2 database that's accepting packets from the network. It expects that a certain packet type will be 1,000 bytes or less, so it creates a buffer that's 1,000 bytes long and copies the data received from the network into the buffer. An attacker can purposely send this database a 2,000-byte buffer to cause a buffer overflow.

The best way to avoid buffer overflows, denial of service attacks, and other problems is to apply the latest FixPak for your version of DB2.

6. SQL INJECTION

Another vector of attack, called SQL injection, comes via Web applications. SQL injection is a method of changing the SQL statements designed for your applications into malicious statements that force your server to do the attacker's bidding. For instance, an attacker may change this SQL statement:

```
Select * from my_table where column_x = '1'
```

into this one:

```
Select * from my_table where column_x = '1' UNION select 1 from
SYSIBM.TABAUTH where 'q'='q'
```

A single query has been modified to change the results entirely.

How does such a transformation take place? Listing 1 shows the source code from a Java Server Page (JSP) that our attacker could use to enter our system. (Notice that the ability to view the source code makes this kind of attack much easier to pull off. Without the source code, an attacker must rely on trial and error to determine if an application is vulnerable — and more trial and error to determine how to exploit the vulnerability.)

```
String sql = new String("SELECT * FROM WebUsers WHERE Username='" +
request.getParameter("username") + "' AND Password='" +
request.getParameter("password") + "'"

stmt = Conn.prepareStatement(sql)
rs = stmt.executeQuery()
```

Listing 1. *Java Server Page authentication code.*

This application is vulnerable because it trusts input from the user. In this case, the user can specify a value that the application uses to create a SQL statement. Specifically, the application concatenates input into a SQL statement without parsing out single quotes.

As you may be able to tell from looking at the code, the page includes an authentication mechanism of the kind typically used to log in to a Web site: The user enters a password and username. Let's say that one valid username for our system is `Bob`, and Bob's password is `Hardtoguess`. When we enter this data, the application generates a SQL statement that selects from the tables where the username and password match the input:

```
SELECT * FROM WebUsers WHERE Username='Bob' AND
Password='Hardtoguess'
```

If a match is found, the user is authenticated. If the recordset is empty, then an incorrect username or password was entered and the login is denied.

This system seems pretty secure, right? Well, imagine that an attacker enters the following in the password field:

```
Aa' OR 'A'='A
```

The attacker has entered a letter, used a single quote to end the string literal, and inserted another Boolean expression. The `WHERE` clause sent to our database by the Java Server Page now looks like this:

```
SELECT * FROM WebUsers WHERE Username='Bob' AND Password='Aa'
OR 'A'='A'
```

The Boolean expression after the OR operator is true in all cases, which means that this query will return all of the rows in the table. Our attacker is now authenticated to the application. And here's the real kicker: The recordset contains the entire user set — and the first user listed is typically the system administrator. There's a good chance the attacker will gain full access to the application.

This is just one example of SQL injection. Attackers can also use this technique to pull data back from tables that aren't directly involved in the original query. One method is to find JSP code on a site that retrieves a dynamic list of items. The trick is to make a single query into two queries and `UNION` them to return additional data in the recordset. In order for this approach to work, the attacker must match up the number of columns and column types. However, if the server provides error messages (for example, `Number of columns does not match` or `Second column in UNION statement does not match the type of the first statement`), the attacker can use trial and error to deduce the number and types of columns.

Listing 2 shows JSP code that would be vulnerable to such an attack.

```
String sql = new String("SELECT * FROM PRODUCT WHERE
ProductName='" + request.getParameter("product_name") + "'"

stmt = Conn.prepareStatement(sql)
rs = stmt.executeQuery()
```

Listing 2. *Java Server Page to retrieve list of products.*

Once again, the JSP doesn't use parameterized queries. Instead, it concatenates a string to build a SQL statement. A valid input for `product_name` might be `DVD Player`, resulting in:

```
SELECT * FROM PRODUCT WHERE ProductName='DVD Player'
```

This statement would select from the `PRODUCT` table. Now, imagine that our attacker sets `product_name` to:

```
test' UNION select TABLE_NAME, COLUMN_NAME from SQLColumns
where 'a'' = 'a
```

The SQL statement sent is now:

```
SELECT * FROM PRODUCT WHERE ProductName='test' UNION select
TABLE_NAME, COLUMN_NAME from SQLColumns where 'a'='a'
```

I'm using the table `SQLColumns` only to demonstrate the example. The attacker can retrieve the results of any table to which the application has access.

How can you prevent SQL injection attacks? Parsing out single quotes from input is a good first step, but true security will require a more systematic approach. You'll need to need to review and update any Web pages that access the database. Establish guidelines for Web programmers that emphasize the use of parameterized queries and prohibit concatenating strings to build SQL statements. In general, you want to tell your developers how to build safe code, not how to patch code that was put into production with holes in it.

Listing 3 shows the application example written more securely using parameterized queries.

```
String sql = new String("SELECT * FROM PRODUCT WHERE ProductName=
:USER_PASSWORD"

Conn.parameters.add(":USER_PASSWORD", request.getParameter("product_name"), 1)
stmt = Conn.prepareStatement(sql)
rs = stmt.executeQuery()
```

Listing 3. *Java Server Page code using bind parameters.*

Due Diligence

Now that you know how an attacker will attempt to exploit your database, you're probably wondering how best to fend attacks off. One strategy is to remain diligent in setting up and maintaining databases, including these critical steps:

- Stay up to date with all FixPaks.
- Perform automated audits of configuration and permissions on a regular basis.
- Check and verify that users are using passwords that can't be guessed.

Practice defense in depth. Realize that computer systems, including DB2, have weaknesses; a skilled attacker can likely find a way around a single layer of defense. Implement multiple layers of defense to reduce the likelihood of a successful attack and the potential damage caused by an attack.

Make sure you have strong authentication in place. Supplement authentication with network encryption (and consider data encryption, too). Enable database auditing and set up procedures to periodically review the log files. Implement an intrusion detection system so that you know when you're under attack. Finally, perform audits and penetration tests (simulated attacks) on your applications regularly.

Securing DB2 is not a trivial task: Performing all these tasks could be a full-time job. Finding the right tools and partners in defending your database is critical.

Security and Compliance

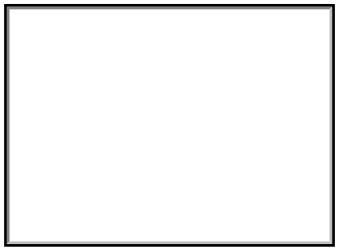
Regulations around the globe are making organizations accountable for information management practices. Here's a short list of the statutes that might be making your organizations sweat.

- The U.S. Sarbanes-Oxley Act requires executives and auditors to attest to the effectiveness of internal controls over financial reporting.
- The U.S. Health Information Portability and Accountability Act (HIPAA) guidelines include legal standards and requirements for protecting nonpublic personal data in databases used by healthcare organizations.
- The Basel II Capital Accords, developed by a committee of 10 countries including the United States, establishes how internationally active banks report on cash and credit risks to protect against losses resulting from internal causes (including employees, processes, and systems) or from external events. Full compliance is required by the end of 2006.
- The U.S. Gramm-Leach-Bliley Act requires financial institutions and their partners to protect nonpublic personal information through access and security controls. Security measures should include management controls that provide segregation of duties and restrictions on access to data. Database auditing is essential for compliance with this law.
- The VISA U.S.A. Cardholder Information Security Program requires that personally identifiable information related to credit card transactions be encrypted in the database.

Aaron C. Newman is CTO and founder of Application Security Inc. (www.appsecinc.com). You can reach him at anewman@appsecinc.com.

Comments? Questions?

Give us your feedback or ask a question of the author.



Please enter your e-mail address below:

Submit

IEEE Std 754-1985 IEEE Standard for Binary Floating-Point Arithmetic - Description

Content

- 7.3 Overflow
- 7.4 Underflow
- 7.5 Inexact

- **8. Traps**
  - 8.1 Trap Handler
  - 8.2 Precedence
- **Annex A Recommended Functions and Predicates**

---

links: [Ordering Information] - [Catalog] - [Standard Status]

(m.v.rodriguez@ieee.org)
URL:
(Modified:Tue Nov 4 03:48:17 2003)

Kartesisches Produkt

aus Wikipedia, der freien Enzyklopädie

Definition

In der Mathematik bezeichnet man als **kartesisches Produkt** (geschrieben als $A \times B$) zweier Mengen A und B die Menge aller geordneten Paare (a,b), wobei a aus A und b aus B ist.

$$A \times B := \{(a,b) | a \in A, b \in B\}$$

Eine Verallgemeinerung ist das kartesische Produkt von n Mengen $A_1, ..., A_n$, es besteht aus allen n-Tupeln $(a_1, ..., a_n)$ mit a_i aus A_i, man schreibt es als $A_1 \times ... \times A_n$, oder als

$$\prod_{i=1}^{n} A_i = A_1 \times ... \times A_n := \{(a_1, ..., a_n) | a_i \in A_i \quad i = 1, ..., n\}$$

Das n-fache kartesische Produkt, bei dem alle A_i gleich A sind, schreibt man auch als A^n.

$$A^n := \prod_{i=1}^{n} A$$

Beispiele

Sei $A=\{a, b, c\}$ und $B=\{x,y\}$. Dann ist: $A \times B = \{(a,x), (a,y), (b,x), (b,y), (c,x), (c,y)\}$.

Sei $A=\{0,1\}$, dann ist $A^3 = A \times A \times A = \{(0,0,0), (0,0,1), (0,1,0), (0,1,1), (1,0,0), (1,0,1), (1,1,0), (1,1,1)\}$.

Der dreidimensionale Vektorraum $\mathbf{R}^3$ besteht aus dem dreifachen kartesischen Produkt von $\mathbf{R}$.

Etymologie

Kartesisch oder **kartesianisch** kommt von *R. Cartesius* (Rene Descartes) und bedeutet allgemein *von Cartesius eingeführt* oder speziell im Fall des Produkts bzw. der Koordinaten rechtwinklig.

Published on **ONJava.com** (http://www.onjava.com/)
http://www.onjava.com/pub/a/onjava/2004/01/07/SQLJoins.html
See this if you're having trouble printing code examples

The Effective Use of Joins in Select Statements

by Satya Komatineni
01/07/2004

There is a school of thought that one programming language could be used for all purposes in developing a solution. For example, if I am a Java programmer, I must be able to do not only my business logic and presentation in Java, but also my persistence in Java, as well. A good goal, certainly, to have. But practicality has a wicked sense of humor and shows up at the table as SQL.

With sleeves drawn up, we, the Java programmers, start to tackle the first citizen of the SQL, called `select`. Very quickly, we meet these quirky things called `join`s. Our original suspicion that we should be sticking to Java is quickly reinforced as the syntax and oddities of `join`s put us on the defense and suddenly seem to show Java in a shining light. But the task at hand is unwavering and your boss is quite certain that it can't be all that hard to write a few SQL statements.

The story of one such journey from a simple `select` to a practical, everyday `select` is presented here in a set of 11 principles. I am hoping this article will be a good companion to non-relational programming languages such as Java, C#, and Perl. The provided examples use SQL Server syntax, but should be applicable to most databases.

The Summary of the 11 Principles

- Principle 1. You can join tables.
- Principle 2. You can join tables with different weights using left outer joins.
- Principle 3. You can outer join a composite table which itself is an equi-join.
- Principle 4. An outer join of two equi-joins is two outer joins combined.
- Principle 5. Once you outer join a table, any joins that include the fields of that table needs to be outer joined as well.
- Principle 6. It is usually not necessary to have a sub-`select` in the `from` clause.
- Principle 7. You use a `select` to retrieve primary "entities" and not a general conglomeration of columns.
- Principle 8. You use joins to add additional properties to a primary entity.
- Principle 9. You can use functions to add additional properties to a primary entity.

- Principle 10. You can conditionally add properties to a primary entity.
- Principle 11. You can add properties to a primary entity using a sub-`select`.

Principles 1 through 6 deal with how to use the `join` construct to solve practical issues in retrieving data from relational databases. A `join` is a general-purpose mechanism (almost like a mathematical tool) for data retrieval. It does not prescribe specific patterns of usage. These principles outlined here put some boundaries on the `join` and show how to use the `join` in a restrictive patternized sense. This is similar to saying that a letter called "Q" exists in the English language, but is almost always followed by a "u" in usage. Similarly, it is important to recognize these `join` patterns aside from their syntax. These six principles show you when to use outer `join`s and the implication of using outer `join`s on the `where` clause. In the process, you will get to know some surprises and patterns.

The second set of principles, 8 through 10, deal with the general question of patternizing a `select` statement. From an OO perspective, the `select` statement is seen as a vehicle for retrieving objects that are called "primary entities."

Together, these principles will form a good foundation for retrieving data from relational databases.

Principle 1. You can join tables.

Database tables are quite simple in their intent. They are no different from their paper cousins, where we use them a lot. In fact, we learn about tabulated data in the fourth and fifth grades. When they are viewed as such, databases are simple to understand. This is an important insight in learning. When you see things for what they really are (as the saying goes, "keeping it real"), the element of fear is taken out of the problem and we begin to learn.

I digress, but I want to tell you an interesting insight about this idea of knowing things for what they are. During the last passage rites of a person in my part of the world, the dear ones of the departed carry with them to the site a pair of mud pots in a wooden sling to carry offerings. Once the rites are complete, these things get abandoned at the site at times. There is a large sense of fear attached to these two objects any context, auspicious or not, as death is equated to the pair. Discounting this, a famous balladeer pronounces "Listen, the sling a piece of wood, and the pots of mud, if you were to perceive the truth."

When information is kept in more than one table, and when those two tables express an idea about a common item, it is a common practice to join the two tables to retrieve information about that item. The join is necessary because we need information about that item, but the information is kept in two tables.

Following that idea, consider six tables: t1,t2,t3,t4,t5, and t6. Let use see how we are told to retrieve information from all of these tables.

```
Select *
>From   t1
        ,t2
        ,t3
        ,t4
        ,t5
        ,t6
Where (any clause involving t1 through t6)
        ex: t1.id=10 and t2.id=10 and t3.id=10 and t4.id=10 and t5.
id=10 and t6.id=10
```

This is called an *equi-join*. There is equal weight for every table. If the joined tables don't have a matching row, those rows from both tables will be eliminated. When the rows match, the rows from both tables are combined and returned. This is the simplest of joins. There is no complex ANSI syntax, where you indicate if this is an outer join, left outer join, right outer join, etc.

The above statement is saying that there is an item called 10 in all of the tables. It retrieves all of the information pertaining to the item 10 that exists in all of the tables. The behavior of this default join is somewhat unintuitive. If t6 doesn't have an item called 10, then the join won't give out any information at all from the other tables. You have to resort to something called an *outer join*, where you tell the join that, after all, t6 is not that important.

Principle 2. You can join tables with different weights using left outer joins.

```
The six tables: t1,t2,t3,t4,t5,t6
Assume:
T1
    t2
    t3
T4
T5
T6
```

Consider the above relationship, where t2 and t3 are independent of each other but both are dependent on t1 and will give t1 more weight. Meaning that if there is a row in t1, give it out, irrespective of t2 and t3. This is represented in SQL Server as:

```
Select *
>From   t1
        left outer join t2 on t2.f1=t1.f1 and t2.f2 = t1.f2
        left outer join t3 on t3.f3=t1.f3
        ,t4
        ,t5
        ,t6
Where (any clause involving t1 through t6)
```

This code is demonstrative of a couple of things. You can get the basics of the syntax for left outer joins. In the example, `t1` is left outer joined with `t2` and `t3`. You can also see how you can combine some left outer joins and some regular joins in the same `select` statement. In the example, `t1` (having already outer joined) is equi-joined with `t4`, `t5`, and `t6`.

Principle 3. You can outer join a composite table which itself is an equi-join.

The previous example treated `t2` and `t3` at par and individually outer joined to `t1`. What if you want to narrow the rows in `t2` first based on `t3`, and then outer join the result to `t1`? The following snippet illustrates this relationship.

```
The six tables: t1,t2,t3,t4,t5,t6
Assume:
T1
    t2
        t3
T4
T5
T6
```

That above requirement necessitates that `t2` and `t3` are equi-joined first. This would have eliminated any rows in `t2` based on `t3`. The result is then outer joined to `t1`. This means we are treating `t2` as an optional data source for additional needs and not a mandatory data source. Subsequently `t1` is equi-joined to `t4`, `t5`, and `t6`. This can be done in SQL Server as follows:

```
Select *
>From   t1
        left outer join
                (select *
                from t2,t3
                where (any where clause involving t1, t2, t3)
```

```
                ) t2_t3_composite_table on t1_t2_composite_table.f1
= t1.f1
        ,t4
        ,t5
        ,t6
Where (any clause involving t1, t2_te_composite_table, t4, t5, t6)
```

Additionally, this code demonstrates the following:

1. Left outer join syntax.
2. Using an inner select as the target of a left outer join.

Principle 4. An outer join of two equi joins is two outer joins combined.

Let us think about the above example for a second and see what necessitated the inner `select`. If `t1` and `t2` are to be outer joined, we can say that rows from `t1` will be displayed irrespective of `t2`. If we are to equi-join `t3` (as `t3` is known to depend on `t2`), then `t1` will not display any rows, even though we mentioned that `t2` is not important. This is because in the next line we said that `t3` is important. So the crux is: how can `t3` be important when `t3` depends on `t2` and `t2` is not important? This situation led us to the inner `select` where we hashed out `t2` and `t3` dependencies first, and then joined the results to `t1`.

The issue with the above is that the syntax is a bit clumsy, as you now have to invent an intermediate table and an intermediate `select`, not to mention the side effects on the optimizers (which may be positive or negative). But after some thought, we can avoid the inner `select` by telling `t1` that both `t2` and `t3` are unimportant. This is because if `t2` is unimportant, then `t3` is automatically unimportant, as it is dependent on `t2`.

This is similar to an algebraic operation such as `-(a+b)` equals `-a and -b`. So we can rewrite the above SQL as follows:

```
Select *
>From   t1
        left outer join t2 on t2.f1=t1.f1 and t2.f2 = t1.f2
        left outer join t3 on t3.f3=t1.f3
        ,t4
        ,t5
        ,t6
Where (any clause involving t1 through t6)
```

Notice that this is similar to the solution of Principle 2. This code also demonstrates the following:

1. Shows the syntax for two left outer joins.
2. Shows how an inner `select` could be avoided with two outer joins.

Principle 5. Once you outer join a table, any joins that include the fields of that table needs to be outer joined, as well.

The above principle of "propagating unimportance" will lead us to this principle as well. This principle guards against some obvious errors in `select` statements that involve outer joins. Once you outer join a table, then the field comparisons on these fields with other tables need to be outer joined as well.

Otherwise, your intention of outer joining (or less emphasizing) the table will be lost. Because when that table returns a null, the subsequent equi-joins on that field will fail and will result in removing the rows from your primary table.

Principle 6. It is usually not necessary to have a sub-`select` in the `from` clause.

Following Principle 4, these inner select joins can be accomplished by listing those tables directly outside. As mentioned, as `-(a+b)` becomes `-a-b`, you can flatten the join structure. This is based on a quick, intuitive conclusion. I might be wrong, but my suspicion is that I may not be that far off. What I am not so sure about is what the implications to optimization are; i.e., whether an inner `select` or an outer `select` is better, from a performance perspective.

Principle 7. You use a `select` to retrieve entities and not a general conglomeration of columns.

For example, you can use a `select` to retrieve orders. You can also use a select to retrieve parts. But you rarely go to retrieve orders, parts, and invoices at the same time. Even when you do, there is always a primary entity you are focusing on. When you retrieve orders, you may retrieve parts that belong to each order, but still, the focus is an order. Similarly, when you retrieve parts, you have an order ID associated with each part, but the focus is still a part in this case. Such an entity of focus in a `select` statement is called the *primary* table.

Principle 8. You use joins to add additional properties to a primary entity.

You can do joins with other tables to provide additional columns. Typically, these other tables will have a one-to-one relationship with the primary table. When they have a one-to-many relationship with the primary entity, you may decide to use "derived one-to-one" relationships. For example, an order has many parts; that is a one-to-many relationship. When you are retrieving a list of orders, it doesn't make sense to join with a parts table. But when you do, you may want to know some aggregate properties of parts belonging to an order. For example, the following are "derived one-to-one" relationships:

```
1. The total number of parts in a given order
2. The total cost of parts in a given order
3. The 1 and 2 are examples of aggregate functions
4. The first part in an order
5. The last part in an order
```

Principle 9. You can use functions to add additional properties to a primary entity.

In the following example, a function is used to add a derived additional property to the primary entity.

```
Select
        col1 as col1,
        my_function(col1,col2,col3) as my_derived_column
From
        table1
```

Inside of the function, you can use complex logic (including many `select`s, `join`s, etc.) to arrive at a value that gets returned. Functions are quite useful when the total number of rows returned are in the tens, as opposed to the hundreds. This is because a function gets called for every row returned; when your result set is large, you are better of doing joins where you can. When your result set is small, you may be able to minimize joins using functions.

Principle 10. You can conditionally add properties to a primary entity.

```
Select
        pt.column1,
        case when pt.conditional_column is null
                then st1.column1
                else st2.column1
        end as column2,
        pt.column2
From
        primary_table pt,
        secondary_table1 st1,
        secondary_table2 st2
Where 1=1
        and (additional where clause)
```

The example demonstrates the following:

1. The syntax for the `case` statement.
2. How to derive a column from two tables, based on a condition.

This pattern could be useful when you manage information in two tables, based on a certain attribute. This may not be a good data model example, but you will see databases where you may have to attempt this.

Principle 11. You can add properties to a primary entity using a sub-select.

```
Select
        pt.column1,
        (select col1 from secondary_table where col2=pt.column2) as
derived_column2
        pt.column3
From
        primary_table pt
Where 1=1
        and (additional where clause)
```

Following the functional approach for columns (where applicable), you can also use a simple sub-select to retrieve a column. The drawback is that you can only return one column for this inner select. Technically, there is no reason why you can't imagine such a thing. For example, one can do this in an app server quite easily. Where possible, such an aggregation will save multiple inner select calls for each outer row.

Instead, you can also join the additional tables to get more columns; this option is always there. But the whole intent of using functions or sub-selects is to minimize the pressures on joins when the rows retrieved are small.

Summary

Programming and effectively using relational databases is an important ingredient in successfully implementing web-based systems. The malleable and transparent nature of databases provide a powerful paradigm for rapid development of front-end systems. The key to harnessing this power depends on effectively using `select` statements to mine relational databases. A `join` construct is an important element in this arsenal. Recognizing well known patterns involving outer joins will form a ground-level vocabulary for your further explorations of your data. This article examined syntax, surprises, and rules of thumb with respect to the usage of joins. The article also explored the primary motive behind a `select`, which, in my mind, is the retrieval of malleable and extensible objects. Knowing these elements should improve productivity as now, one can understand the systems from their front ends to their database back ends.

Further References

- SQLServer Code Samples Knowledge Folder (*kf: /akc/satya/cs-sqlserver*). This knowledge folder documents some helpful information on SQL Server and databases.
- A list of all of the author's knowledge folders (*kf: /akc/satya/public folders*).

Satya Komatineni is the CTO at Indent, Inc. and the author of Aspire, an open source Web development RAD tool for J2EE/XML.

Return to ONJava.com.

Home

Dialog

Fragen, Anregungen, Wünsche? Die Feedback-Seite!

Hyperlinks

- ○ Ungültiger Hyperlink ...
- ○ Überflüssiger Hyperlink ...
- ○ fehlender hyperlink ...

... von Seite:

http://www.jeckle.de/

... nach Seite:

http://

Bemerkungen:

Seiteninhalt

- ○ Nicht mehr aktueller Inhalt ...
- ○ Inkorrekter Inhalt ...
- ○ Irreführender Inhalt ...

... auf Seite:

http://www.jeckle.de/

Bemerkung:

Anregungen, Wünsche, Fragen

Bitte tragen Sie weitere Anregungen, Wünsche und Fragen hier ein:

Und nun...

Ihr Name:

Ihre Email-Adresse:

ab die Post! Eingaben löschen

Service provided by Mario Jeckle

Generated: 2004-06-08T12:49:04+01:00

Feedback SiteMap

This page's original location: http://www.jeckle.de/feedback.html

RDF description for this page

Unified Modeling Language (UML)

- UML Resource Center
- UML FAQs @ jeckle.de
- UML Links
  - Generelles
  - Tutorials
  - Frequently Asked Questions (FAQ) und Mailinglisten
  - UML @ OMG
  - Austauschformate
  - Werkzeugbezogenes ...
  - Objektorientierung allgemein
  - UML Autoren
  - UML auf gut Deutsch(Vorschlag einer Gruppe deutschsprachiger Autoren zur einheitlichen deutschen Übersetzung der UML-Begriffe)
- UML Bücher
- UML Official Specification Documents / Offizielle UML Spezifikation
  - General Information
  - UML Profiles und sonstige UML-bezogene Aktivitäten
  - UML v2.0
  - UML v1.5
  - UML v1.4
  - UML v1.3
  - UML v1.2
  - UML v1.1
  - Unterschiede zwichen den UML-Versionen
- UML Online Publications / Netz Resourcen zur UML
- UML Tools
- UML2SVG-Transformationsdienst

eXtensible Markup Language (XML)

▲

XML Metadata Interchange (XMI)

▲

Vorträge und Publikationen

Vorträge und Publikationen

Abstracts und Kurzfassungen

▲

Vorlesungen

Studien- und Abschlußarbeiten

Web Services

- Web Services -- eine technisch orientierte Einführung
- Sicherheitsaspekte XML-basierter Web-Services
- Web Service Workshop *WS-RSD'02*
- Web Services @ Berliner XML-Tage

▲

GOOAL.net

Deutsche Mailingliste rund um die Objekttechnologie

direkt zu GOOAL.net ...

▲

Internet Search Engines / Suchmaschinen

- Deutsche Suchmaschinen
- Meta-Suchmaschinen
- Internationale Suchmaschinen
- E-Mail Finder
- Dictionaries und Übersetzungshilfen
- File Finder
- Books & Journals
- Page Registrierung

▲

Mersennesche Primzahlen

- Mersennesche Primzahlen
- SETI @ Home
- Rätsel der Proteinfaltung

▲

Web Services Workshop WS-RSD'02

- Workshop Homepage
- Call for Papers
- Workshop Program, Schedule, and Presentation Slides Workshop Program, Schedule, and Presentation Slides

▲

Free Stuff --- Software & Downloads

- XInclude Task for Ant
- Java version of Linpack benchmark
- Timer Task for Ant
- SOAPing New
- Java Assembler Playground New

▲

RSS-Newsfeeds

- jeckle.de Newsfeed (HTML-Version)
- Linux Kernel News
- Call for Papers Corner (HTML-Version)
- DB-Anwendungen Newsfeed (HTML-Version)
- e-Business Engineering Newsfeed (HTML-Version)
- Datenbanken Newsfeed (HTML-Version)
- Mehr Informationen zu RSS

▲

Feedback

direkt zu Seite ...

Rotkreuz Mitgliederverwaltung

direkt zu Seite ...

Dialog

direkt zu Seite ...

Über diese Seiten ...

direkt zu Seite ...

Suchen ...

direkt zu Seite ...

Mario Jeckle ...

direkt zu Seite ...

PGP public key

Service provided by Mario Jeckle

Generated: 2004-06-08T12:51:56+01:00

Feedback SiteMap

This page's original location: http://www.jeckle.de/sitemap.html

RDF description for this page