ENTWICKLUNGSSCHEMA

grobes Phasenkonzept	← PROBLEMANALYSE				REALISIERUNG →
2-Ebenen-Konzept (DB)	logische Entwurfsebenen WAS? (informationsrelevant)				physische Entwurfsebene WIE? (speicher- und zugriffsrelevant
3-Ebenen-Konzept (DB)		externe Ebene	konzeptionelle Ebene		interne Ebene
Modellarten		logische Teilmodelle	logisches Gesamtmodell	Formatmodelle	physische Modelle
feines Phasenkonzept	grobes Gesamtfachkonzept	feine Teilfachkonzepte	feines Gesamtfachkonzept	Benutzeroberfläche	DV-Konzept
Funktionsmodellierung	Systemüberblick	Komplexe Berechnungsformeln	Aufgabenstruktur: Funktionsbaum	Menü mit Systemfunktionen	Feinmodularisierung
	Datenflußplan	Entscheidungstabellen	Grobmodularisierung	Datensicherung Zugriffsberechtigung Funktionstastenbelegung	Programmablaufpläne
	Prozeßmatrix				
Datenmodellierung	grobes Mengengerüst	logische Teil-Data Dictionaries evtl. 3 NF	logisches Data Dictionary in 3 NF	Abbildung der logischen Teil-Data Dictionaries auf Bildschirmmasken, Druckformate und Schnittstellen	physisches Data Dictionary Anpassung der 3 NF (Zugriffsverhalten) Dateien, Satzarten, Datensätze, Datenfelder, Datendarstellung
		Benutzerviews abgeleitete Tabellen	Basistabellen (Tupel,Attribute)		
		→ Verfeinerung Top-Down	→ iterative Integration Bottom-Up	Zeit t →	

A.4

3 Datenmodellierung und Vergleich verschiedener Datenstrukturdiagramm-Typen

Motivation :

- o Mathematische Beschreibung der in einem relationalen Datenmodell moeglichen Beziehungstypen
- o Erlaeuterung der Modellierung eines realen Systems - allgemein und am Beispiel des Subsystems "Personalverwaltung"
- o Vorstellen verschiedener DSD-Typen und Visualisierung des Datenmodells "Personalverwaltung" mit den einzelnen DSD-Typen
- o Vergleich der vorgestellten DSD-Typen und begruendete Auswahl eines DSD-Typs fuer die weitere Verwendung zur Loesung des Problems - Ableiten von allen moeglichen Joinwegen aus einem vorliegenden Relationennetz

Folgende Terminolgie wird festgelegt:

Entität:

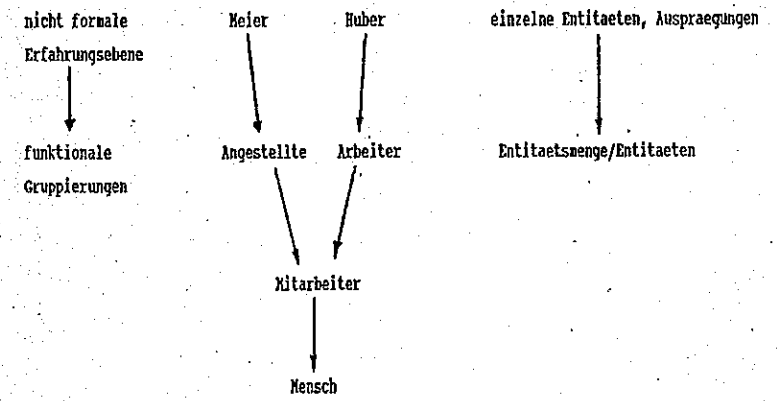
Es ist nicht das Ziel, eine wissenschaftliche Definition des Entitätsbegriffs zu geben, sondern nur eine Arbeitsdefinition.

Entiät ist eine Systembeschreibungsgröße ("Modell"). Man unterscheidet:

- Entitaeten des nicht formalen Praemodells
- Entitaeten des formalen Datenmodells

Entitäten des nicht formalen Prämodells sind gleichzusetzen mit systemimmanenten Größen des realen Systems, z.B. Personen, Institutionen bzw. immaterielle Objekte. Sie werden als intuitiv klare Entitäten bezeichnet. Diese werden durch Abstraktion auf die Ebene formaler Modelle abgebildet. Es werden Entitäten des formalen Datenmodells beschrieben, denen immanente Größen des realen Systems nicht mehr eindeutig zugeordnet werden können.

Abstraktion ---------> Oberbegriffe



Im folgenden wird der Terminus Entität im Sinne von Entitätsmenge angewendet. Unter Ausprägung werden einzelne Elemente einer Entitätsmenge/Entität bezeichnet. Unter dem Terminus Beziehung werden alle gem. der mathematischen Modellbeschreibung (siehe Abschnitt 3.1) möglichen Beziehungen zwischen Entitäten zusammengefaßt.

Hinweis:
In Abschnitt 3.3.2 wird für die Beschreibung des ERD Entität gem. [Flatscher,1990,S29] unterschiedlich definiert.

1.2

3.1 Mathematische Modellbeschreibung

Um die einzelnen Beziehungen, die in einem Datenmodell auftreten können, eindeutig zu beschreiben, müssen die zugrunde liegenden mathematischen Termini erörtert werden.

gegeben seien: D,W, f:D ----> W und $y \in W$

Die Definitionsmenge D stellt hier die Menge aller, in einer Relation möglichen Fremschlüsselausprägungen (FSA) dar. Die Wertemenge W umfaßt alle zugehörigen Primärschlüsselausprägungen (PSA). Die Interpretation von D und W läßt genau drei verschiedene "Urbild"-Typen $\bar{f}^{1}$ zu, mit
$\bar{f}^{1}(y) := \{x \in D : f(x) = y\}$

"Urbild"-Typ (1) - surjektiv

D -----f-----> W $\quad y \in W \nexists\ x \in D : f(x) = y$, d.h. $\#(\bar{f}^{1}(y))=0$

0 : 1

Es existiert keine Relation B, für die gilt: Alle PSA von A fungieren in B als FSA.

"Urbild"-Typ (2) - surjektiv und injektiv

D -----f-----> W $\quad y \in W \exists_1\ x \in D : f(x) = y$, d.h. $\#(\bar{f}^{1}(y))=1$

1 : 1

Es existiert eine Relation A und eine Relation B, für die gilt: PSA von A fungiert in B genau einmal als FSA.

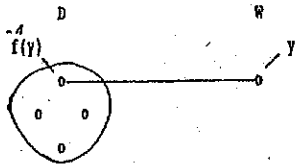


Abb. 3.1: Darstellung des "Urbild"-Typ (2)

"Urbild"-Typ (3) - surjektiv

D -----f-----> W $\quad$ zu $y \in W \exists_n\ x \in D : f(x) = y$, d.h. $\#(\bar{f}^{1}(y))=n>1$

n : I (n > 1)

Es existiert eine Relation A und eine Relation B, für die gilt: PSA von A fungiert in B mehrmals (n-mal) als FSA.

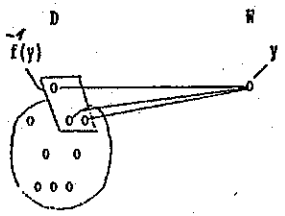


Abb. 3.2: Darstellung des "Urbild"-Typ (3)

Durch die Überlagerung der verschiedenen Abbildungsmöglichkeiten der einzelnen "Urbild"-Typen wird die Definition funktionaler Beziehungen zwischen Relationen erreicht.

Funktionale Beziehungen

(1:n*)-Beziehung

Die (1:n*)-Beziehung kommt durch die Abbildungen des "Urbild"-Typs (3) oder durch die überlagerte Abbildung von "Urbild"-Typ (2) und (3)zustande.

"Urbild"-Typ (3)

```
D -----f-----> W

n     :     1  (n>1)
```

"Urbild"-Typ (2) und (3)

```
D -----f-----> W

1     :     1
n     :     1  (n>1)

==>   n     :     1  (n>=1)
```

(1:n)-Beziehung

Die (1:n)-Beziehung kommt durch die Kombination der Abbildungen der "Urbild"-Typen (1) und (3) oder durch die Kombination der "Urbild"-Typen (1), (2) und (3) zustande.

"Urbild"-Typ (1) und (3)

```
D -----f-----> W
0     :     1
n     :     1  (n>1)

==>   n     :     1
      mit (n=0 v n >1)
```

"Urbild"-Typ (1),(2) und (3)

```
D -----f-----> W
0     :     1
1     :     1
n     :     1  (n>1)

==>   n     :     1  (n>=0)
```


(1:c)-Beziehung

(1:c)-Beziehungen resultieren aus der Überlagerung der Abbildungen von "Urbild"-Typ (1) und (2)

```
D -----f-----> W

0     :     1
1     :     1

==>   c     :     1  (c=0 v c=1)
```

Relationale Beziehungen

(n:m)-Beziehung

Die (n:m)-Beziehung besteht aus der Verknüpfung von zwei (1:n)- bzw. (1:n*)-Beziehungen oder auch umgekehrt, d.h. eine (n:m)-Beziehung läßt sich in zwei (1:n*)-Beziehungen auflösen:

```
D -----f-----> W

n     :     1
1     :     m
```

Im Mengendiagramm wird folgende Situation dargestellt :

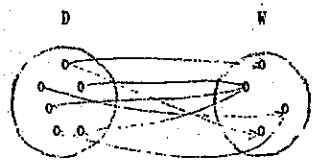


Abb. 3.3: Mengendarstellung einer (n:m)-Beziehung

An einer (n:m)-Beziehung beteiligte Relationen müssen, um sie mit funktionalen Abbildungenvorschriften darstellen zu können, in zwei getrennte (1:n)- bzw. (1:n*)-Beziehungen aufgespalten werden. Die Relationen, die aus der Aufspaltung einer (n:m)-Beziehung resultieren (d.h. Relationen, die einen zusammengesetzten Primärschlüssel, bestehend aus den Primärschlüsseln der Ausgangsrelationen, besitzen), werden als Ergebnisrelation einer (n:m)-Beziehung bezeichnet.

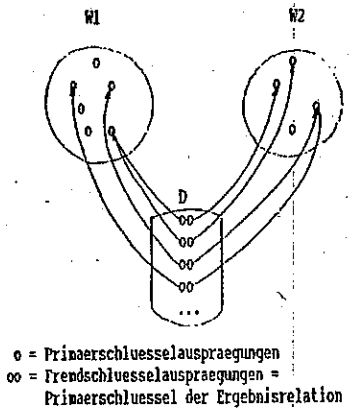


Abb. 3.4: Darstellung einer aufgespaltenen (n:m)-Beziehung

Durch die mathematische Definition der einzelnen Beziehungen können die möglichen Beziehungen in Datenmodellen eindeutig beschrieben werden.

3.2 Datenmodellierung

Die Datenmodellierung stellt einen Prozeß dar, der reale Systeme auf Datenmodelle abbildet, die in DSDen visualisiert werden können (siehe Abschnitt 3.3).

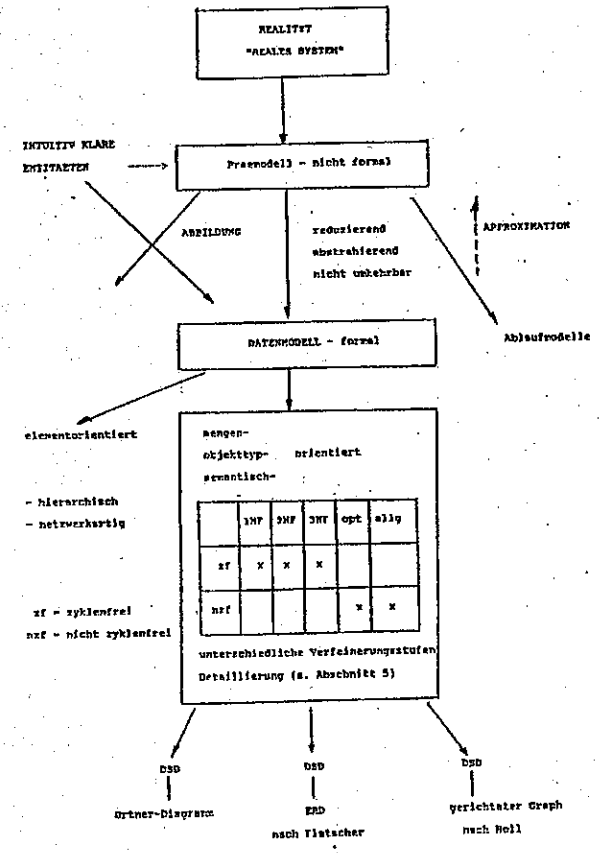


Abb. 3.5 Verfahrensweise bei der Datenmodellierung

Die Abbildung eines realen Systems auf ein Datenmodell stellt einen reduzierenden und abstrahierenden Prozeß dar, d.h. das reale System wird in ein Datenmodell gezwängt, bei dem unweigerlich Informationsverluste auftreten. Aus diesem Grund ist eine Umkehrung: Datenmodell —> Realität nicht möglich; man spricht in diesem Fall von einer Approximation des realen Systems durch das Datenmodell.

Es muß eine möglichst informationsverlustfreie Abbildung des realen Systems auf das Datenmodell angestrebt werden, um eine maximale Annäherung des Datenmodells an das reale System zu erreichen (d.h. Minimierung der Informationsverluste).

Die Datenmodelltypen werden eingeteilt in

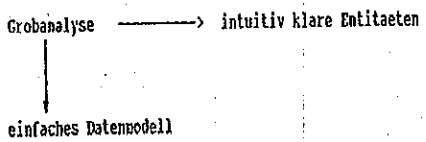
- elementorientierte Modelltypen (hierarchisch, netzwerkartig)
- mengenorientierte Modelltypen (relational)

Es wird von einem mengenorientierten Modelltyp ausgegangen. Er bildet die Grundlage für die Relationenmodellierung (siehe Abschnitt 4).

Die Datenmodellierung wird in folgende Einzelschritte gegliedert:

- Grobanalyse
- Feinanalyse

1. Grobanalyse



Wird ein reales System auf ein Datenmodell abgebildet, so wird in einem sehr frühen Stadium mit Hilfe der Sinneswahrnehmung und der Sprache eine Prämodellierung durchgeführt, die intuitiv klare Entitäten erkennt. Im Beispiel "Personalverwaltung" werden "Abteilung", "Gruppe", "Betriebsangehörige", "Sozialversicherungsträger" etc. als intuitiv klare Entitäten bezeichnet.

Zur genauen Beschreibung der intuitiv klaren Entitäten und ihren gegenseitigen Beziehungen untereinander, können folgende Teilschritte durchgeführt werden :

- Aussagen- bzw. Stoffsammlung
- Klaerung der Fachbegriffe (Beziehungen von Entitaetsmengen), soweit sie nicht durch das allgemeine Verstaendnis zweifelsfrei definiert sind
- Analyse der in der Aussagensammlung beschriebenen Beziehungen zwischen den Entitaeten, um diese Beziehungen mathematisch eindeutig gem. Abschnitt 3.1 zu analysieren

Die Ergebnisse der Durchführung dieser Teilschritte für die Entwicklung der Beispieldatenbank "Personalverwaltung" wurde in vollem Umfang in Anhang A1 - A3 aufgenommen.

Die Informationen über die intuitiv klaren Entitäten und ihre gegenseitigen Beziehungen finden in ihren Niederschlag in einem einfachen Datenmodell. Die einzelnen Beziehungskomplexe (Anhang A3) werden mosaikartig zu einem vollständigen Grobdatenmodell zusammgengefügt.

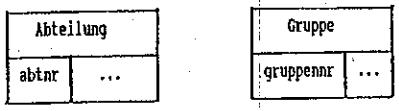
Vergabe der Primärschlüssel- bzw. Fremdschlüssel

Entitätsmengen eines Datenmodells müssen eindeutige Primärschlüssel (bestehend aus einem Datenfeld oder(- kombinationen) zugeordnet werden bzw. die Beziehungen der Entitätsmengen untereinander durch die Vergabe von Fremdschlüsseln beschrieben werden :

A.6

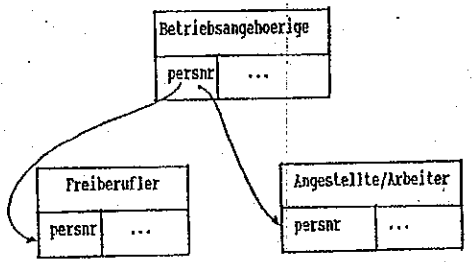
(a) Einfachschlüssel

Einfachschlüssel erhalten alle Entitätsmengen, die nicht an einer (1:c)-Beziehung beteiligt sind und nicht Ergebnis-Entitätsmenge einer (n:m)-Beziehung sind (siehe Abschnitt 3.1), z.B. :

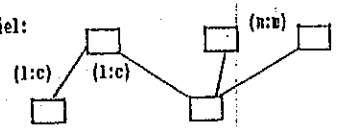


(b) Gemeinsamer Schlüssel

Gemeinsame Schlüssel erhalten alle an einer (1:c)-Beziehung beteiligten Entitätsmengen, die nicht Ergebnis-Entitätsmenge einer (n:m)- Beziehung sind. Der Gemeinsame Schlüssel fungiert in den verfeinerten Entitätsmengen ("Freiberufler","Angestellte/Arbeiter") sowohl als Primärschlüssel als auch als Fremdschlüssel, z.B.:



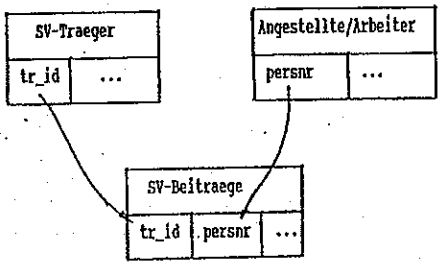
Negativbeispiel:



Die Entitaeaetsmenge, die an einer (1:c)-Beziehung beteiligt ist, aber Ergebnis-Entitaetsmenge einer (n:m)-Beziehung darstellt, erhalten keinen gemeinsamen Schluessel (siehe c)).

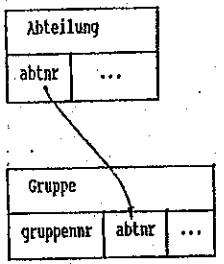
(c) Komplexer Schlüssel

Entitätsmengen, die aus einer (n:m)-Beziehung resultieren, erhalten einen komplexen Schlüssel, bestehend aus den identifizierenden Schlüsseln der beteiligten Ausgangsentitätsmengen. Die einzelnen Attribute des zusammengesetzten Primärschlüssels der Ergebnis-Entitätsmenge der (n:m)-Beziehung fungieren zugleich als Fremdschlüssel. Ein Beispiel dafür aus unserer Personalverwaltung ist die (n:m)-Beziehung zwischen "SV-Träger" und "Angestellte/Arbeiter".



(d) Fremdschlüssel

Entitätsmengen, die aus einer (1:n)-/(1:n*)-Beziehung resultieren, erhalten einen Fremdschlüssel durch die Übernahme des eindeutig identifizierenden Schlüssels der Ausgangsrelation, z.B.:



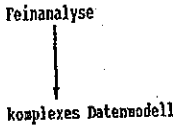
47

Für das Beispiel "Personalverwaltung" ergibt sich folgende Schlüsselverteilung : (Die Schlüsseltabelle becksichtigt bereits die in Abschnitt 4 festgelegten Namenskonventionen, d.h. jedem Attribut muß ein Tabellennamenkürzel vorangestellt werden, das die Attribute eindeutig Tabellen zuordnen.)

Entitaet	Einfach-schluessel	Geneinsaaer Schluessel	Komplexer Schluessel	Trend-schluessel	Art-schluessel
Abteilung	abt_abtnr				
Adresse	ad_adressnr			ad_persnr	
Ang_Arb		aa_persnr		aa_persnr aa_fischl	
Arbeitsplatz	ar_platznr			gr_gruppennr	
BA		ba_persnr		ba_platznr ba_blz	ba_art
Bank	bk_blz				
Fehlzeiten	fehl_fehlnr			fehl_persnr	
Finanzart	fi_fischl			fi_blz	
Freiberufler		fb_persnr		fb_persnr	
Gruppe	gr_gruppennr			abt_abtnr	
SV_Beitraege			(sv_persnr, sv_tr_id)	sv_persnr sv_tr_id	
SV_Traeger	svt_tr_id			svt_blz	
Ueberstunden	ue_uenr			ue_persnr	
Vers_vertrag	vv_vvnr			vv_persnr vv_veschl	
Versicherung	ve_veschl			ve_blz	

Abb. 3.6 Schluesseltabelle fuer das Beispiel "Personalverwaltung"

2. Feinanalyse



Mit der Durchführung einer Feinanalyse bzw. der Normalisierungsvorschriften (siehe Abschnitt 4) kann ein einfaches Datenmodell verfeinert werden. Ein komplexes Datenmodell wird durch diesen Vorgang beschrieben.

Datenmodelle - jeder Abstraktionsebene können in einem DSD optisch dargestellt werden. DSDe sind Hilfskonstruktionen, die über syntaktische Elemente verfügen, um Datenmodelle, d.h. Entitätsmengen und ihre gegenseitigen Beziehungen untereinander, zu porträtieren.

Auf dem "Markt" existieren viele DSD-Typen, es gibt keinen einheitlichen Standard, Datenmodelle graphisch in einem DSD darzustellen. In Abschnitt 3.2 findet ein Vergleich verschiedener DSD-Typen statt. Es soll ein DSD-Typ ausgewählt werden, der sich für die Problematik - Ableiten von allen möglichen Joinwegen aus einem vorliegenden Relationennetz am besten eignet.

3.3 Vergleich verschiedener Datenstrukturdiagramm-Typen

DSDe werden in der Literatur häufig als Datenstrukturmodelle und sogar als Datenmodelle angepriesen und suggerieren daher einen weit höheren theorethischen Wert als er ihnen tatsächlich zukommt. DSD sind nur Mittel zur graphischen Darstellung von Datenmodellen.

3.3.1 Das Datenstrukturdiagramm nach Ortner

Dieses DSD erfordert ausführliche und detaillierte Erklärungen, da es sich hierbei um eine weniger verbreitete Darstellungsart handelt. Folgende Begriffe werden synonym verwendet :

eigene Terminolgie	Ortner
Ausspraegung	Objekt
Entitaetsmenge/Entitaet	Objekttyp
Beziehung	Beziehung

Ganz allgemein handelt es sich um eine Darstellung eines semantischen Datenmodells. Anhand von Aussagen über das System werden (Objekttypen) dadurch eindeutig bestimmt, daß man ihre Beziehungen zu anderen Objekttypen erfaßt. Die gewonnenen Objekttypen und ihre Beziehungen werden mit folgender Symbolik dargestellt [Ortner,1989,S33]:

Objekttyp

---------- Beziehungen werden durch Kanten zwischen Objekttypen dargestellt

Diese Kanten tragen Beziehungsinformationen :

Pfeile geben die "Beziehungsverhaeltnis" (Ortner) an :

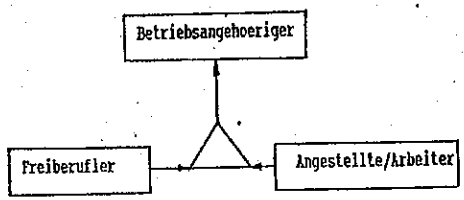
-----> fuer ein
---->> fuer n bzw m (n,m >= 1)

senkrechte Striche informieren ueber die "Beziehungshaeufigkeit" (Ortner):

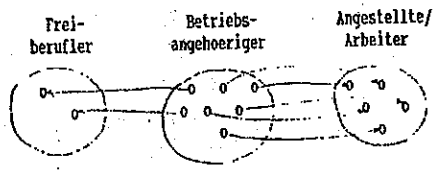
------ fuer alle
---/-- fuer einige = n (n >= 0)
---//- fuer genau ein

Die in der mathematischen Modellbeschreibung (siehe Abschnitt 3.1) dargestellten Beziehungen (d.h. "Beziehungswirkung" (Ortner)) erhalten eine eigenständige Symbolik.

Die Inklusion (Verallgemeinerung) - (1:c)-Beziehung:

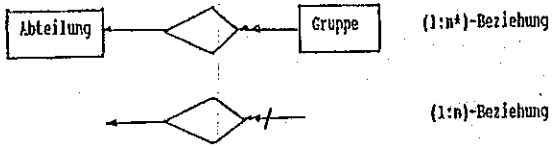


Eine Inklusion verfeinert einen Objekttyp (Betriebsangehöriger) in eine weitere, disjunkte Untergliederung. Dies bedeutet: Ein BA ist entweder ein (Angestellter oder Arbeiter) oder er ist freiberuflich im Unternehmen tätig. Dieses Strukturelement impliziert in der Regel eine (1:c)- Beziehung der Objekte nach folgendem Schema:

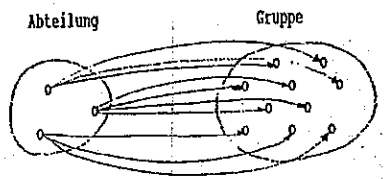


1.9

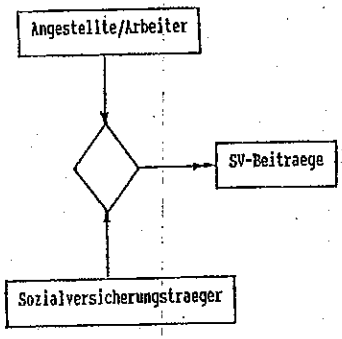
Die Aggregation (Verdichtung) - (1:n*)- / (1:n)-Beziehung:



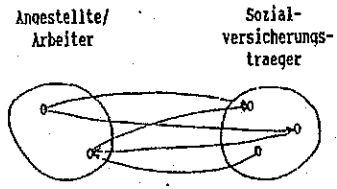
Es werden Objekte (einzelne Datensätze) eines Objekttyps (Gruppe) unter den Objekten eines anderen Objekttyps (Abteilung) funktional zusammengefaßt. Dies bedeutet: Eine Abteilung "X" besteht aus den Gruppen "A","B","C",... . Strukturell wird dadurch eine (1:n)- / (1:n*)-Beziehung definiert, die sich im Mengendiagramm folgendermaßen darstellt:



Die Konnexion (Verknüpfung) - (n:m)-Beziehung:



Durch die Konnexion von Objekten verschiedener Objekttypen z.B. (Angestellte, Arbeiter; SVT) entsteht ein neuer Objekttyp (SV-Beiträge). Dies bedeutet: Ein Angestellter oder Arbeiter leistet Abgaben = SV-Beiträge an verschiedene SV-Träger, wobei ein SV-Träger alle Angestellte und Arbeiter versichert. Das Strukturelement "Konnexion" charakterisiert daher eine (n:m)-Beziehung, die in der Mengenschreibweise wie folgt symbolisiert wird:



Wird das Datenmodell für das reale System "Personalverwaltung" mit dem DSD nach Ortner dargestellt, ergibt sich folgendes Gesamtbild :

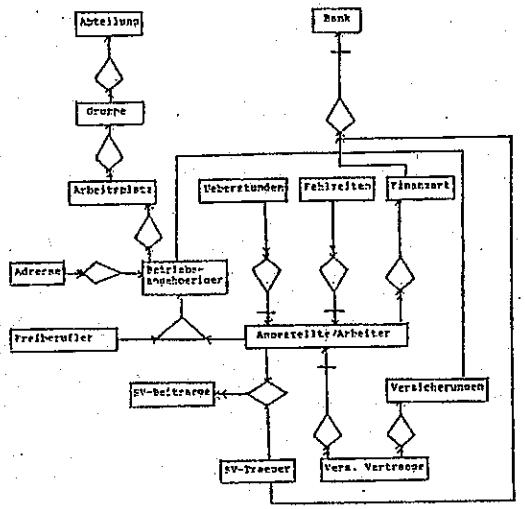


Abb. 3.7: DSD nach Ortner fuer das Beispiel "Personalverwaltung"

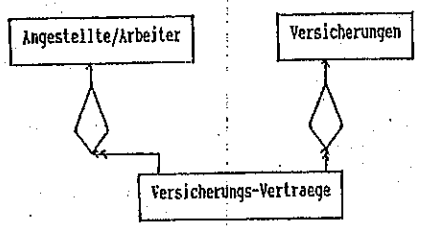
1.10

Wertung :

Unter dem Aspekt, daß die Darstellung des Datenrmodells als DSD als wichtigste visuelle Grundlage für die Kommunikation zwischen den Datenbankdesignern, Anwendern, Programmierern ... dienen soll, scheint das DSD nach Ortner auf den ersten Blick etwas unstrukturiert und unübersichtlich. Die komplizierte Symbolik ist nicht selbsterklärend und verursacht daher immer wieder Unklarheiten.

Redundanzen in der Ortner-Symbolik könnten vermieden werden :
Mit der Pfeilsymbolik werden Beziehungsverhältnisse unterschieden. Zusätzlich wird mit der expliziten Darstellung der Inklusion, Aggregation und Konnexion dies ein zweites Mal getan. Würde man eines der beiden Symboliken weglassen, würde das DSD nach Ortner keine Informationsverluste verbuchen, sondern exakt die gleiche Information vermitteln. Hinweis: Die Symbolik der Entitäten und Beziehungen ist vergleichbar mit der von semantischen Netzen (Objekte in Kästchen, Beziehungen an den Pfeilen), jedoch unterbricht die Beziehungssymbolik nach Ortner die Pfeile. Dadurch sind Fehlschlüsse von unpassender Darstellung zurück auf die Therorie möglich.

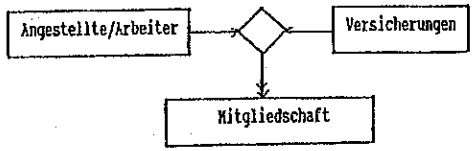
Jedoch weist das Grobdatenmodell einen hohen Detailierungsgrad auf. Man betrachte die beiden (1:n)-Beziehungen zwischen Angestellte/Arbeiter, Versicherungs-Verträge und Versicherungen.



Man könnte diesen Beziehungskomplex durch folgende Argumentation sehr leicht mit einer Konnexion ((n:m)-Beziehung) verwechseln :

"Jeder Angestellte/Arbeiter kann bei mehreren Versicherungen Mtglied sein - eine Versicherung hat mehrere Angestellte/Arbeiter als Mitglieder.

Diese Aussage würde eindeutig eine (n:m)-Beziehung definieren.



Der Objekttyp "Mitgliedschaft" müßte dann einen komplexen Schlüssel erhalten, der sich aus den Einfachschlüsseln der anderen Objekttypen zusammensetzt :

Objekttyp	Schluessel
Ang_Arb	persnr
Versicherung	veschl
Mitgliedschaft	(persnr, veschl)

Diese Aussage wäre jedoch unvollständig und würde in ihrem Beziehungskomplex dem realen System nicht entsprechen. Würde der Fall eintreten, daß ein Angestellter/Arbeiter bei einem Versicherungsträger mehrere Versicherungen abschließt - was durchaus der Wirklichkeit entspricht - könnten diese Objekte des Objekttyps Mitgliedschaft nicht mehr unterschieden werden. Sie würden exakt die gleiche Schlüsselkombination erhalten.

Diese Feinheiten werden bereits optisch im DSD unterschieden, Verwechslungen zwischen (n:m)-Beziehungen und sog. "gekoppelten" (1:n)-Beziehungen wird somit ausgeschlossen.

3.3.2. Das Entity-Relationship-Diagramm (ERD)

Das ERD visualisiert, ähnlich wie das Ortner-Diagramm, ein Datenmodell der konzeptionellen Ebene. Dabei werden folgende Begriffe synonym verwendet:

eigene Terminologie	Ortner	ERD
Ausspraegung	Objekt	Entitaet
Entitaetsmenge/Entitaet	Objekttyp	Entitaetstyp
Beziehung	Beziehung	Beziehungstyp

Die wichtigsten Elemente des ERD werden kurz vorgestellt Flatscher,1990,S34]:

starker Entitaetstyp: Betriebsangehoerige

Ein starker Entitätstyp wird auch als Kernentität bezeichnet [Vetter,1990,S39/40].

Schwacher Entitaetstyp: Ueberstunden

Ein schwacher Entitätstyp wirauch als abhängige Entität bezeichnet [Vetter,1990,S40].

Beziehungstyp: leistet — Explizite Definition einer Beziehung im Strukturdiagramm

Attribut: Personalnummer — Darstellung eines Primaerschluessels eines Entitaettyps

Das ERD unterscheidet bei ihrem Beziehungstyp grundsätzlich nach

- Kardinalverhältnissen
- strukturellen Bedingungen

Die Kardinalverhältnisse entsprechen den (1:1)-, (1:n)- und (n:m)-Beziehungen, gem. der mathematischen Modellbeschreibung (siehe Abschnitt 3.1). Die strukturellen Bedingungen geben die Beziehungshäufigkeit an, d.h. wie oft eine Entität eines bestimmten Entitättyps in einem bestimmten Beziehungtyp enthalten sein kann. Die strukturellen Bedingungen entsprechen so den senkrechten Strichen der Ortner-Symbolik.

Ortner	ERD
-------	(1,1)
*)	(0,1)
-------	(1,n)
----/--	(0,n)
---//--	nicht beruecksichtigt

*) wird mit der Inklusion-Symbolik beschrieben

Vorgehensweise

1. Erstellen eines groben ERD
2. Erstellen eines detaillierten ERD durch schrittweise Verfeinerung

A.12

1. Das grobe ER-Diagramm

Aufgrund einer vorangegangen Datenanalyse werden die benötigten Entitätstypen erfaßt und dann sämtliche Verbindungen (egal welcher Art) zwischen ihnen festgestellt. Das resultierende Diagramm weist sich durch besondere Übersichtlichkeit aus.
Eine Verfeinerung der Entitätstypen bis auf die Attributebene ist bereits in diesem Schritt möglich (jedoch wird diese Feindatenmodellierung gesondert in Abschnitt 4 behandelt).

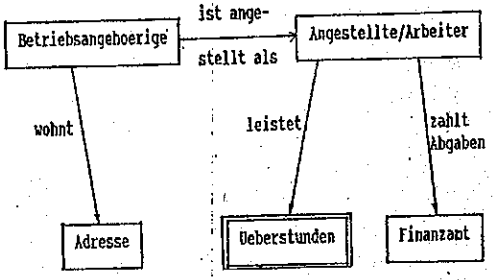


Abb. 3.8: Ausschnitt aus den groben ERD "Personalverwaltung"

2. Erstellen eines detaillierten ERD durch schrittweise Verfeinerung

Aufgrund der im ersten Schritt gefundenen Beziehungen werden diese nun in konkreten Beziehungstypen formuliert. Das Strukturdiagramm erweitert sich um die dafür vorgesehene Symbolik. Es muß die Art des Beziehungstyps genau vermerkt werden, d.h. Bestimmen der Kardinalsverhältnisse (z.B. (1-N), sowie der strukturellen Bedingungen (z.B. (1,n)), desweiteren werden die Primärschlüssel im Strukturdiagramm angegeben.

Das Strukturdiagramm "Personalverwaltung" gestaltet sich als ERD folgendermaßen :

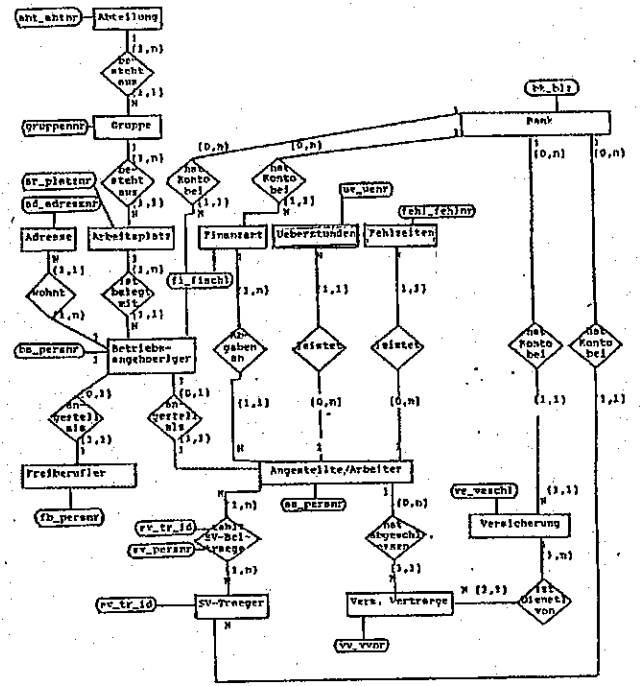


Abb. 3.9: Detailliertes ERD "Personalverwaltung"

Wertung :

Das ERD wirkt durch die explizite Angabe der Beziehungstypen sehr ausladend. Mit der vorgesehenen Symbolik lassen sich die verschiedenen Kardinalverhältnisse nicht differenzieren, so daß eine zusätzliche Beschriftung unerläßlich wird.

1.13

Bereits bei kleinen Datenbanken, wie der "Personalverwaltung" wächst daher das DSD sehr rasch an, muß bei komplexeren DSD oft über mehrere Seiten verteilt werden und verhindert so einen raschen, gezielten Informationsüberblick.

Bei großen Systemen erscheint dieses Modell daher ungeeignet. Es ist nicht gelungen, die Vielzahl an Informationen in übersichtlicher Weise darzustellen, verursacht durch die Differenzierung von Kardinalsverhältnissen und strukturellen Bedingungen bei der Darstellung im ERD. Die Deklaration der strukturellen Bedingungen gewinnt erst bei genauerem Studium des Diagramms an Transparenz.

Eine (n:m)-Beziehung wird im ERD nicht aufgespalten, d.h. die eine aus einer (n:m)-Beziehung resultierende Ergebnisentität (hier: "SV-Beitraege") wird nicht als eigenständige Entität dargestellt, sondern bleibt hinter dem Symbol des Beziehungstyps verborgen.

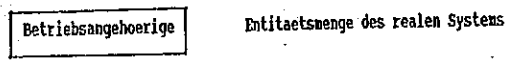
Der Detailierungsgrad, der durch das ERD erreicht wird, genügt sehr hohen Ansprüchen.
Auch in diesem DSD ist eine visuelle Unterscheidung einer (n:m)-Beziehung und einer sog. "gekoppelten" (1:n)-Beziehung möglich (siehe Kapitel 3.3.1, Beziehungskomplex Angestellte/Arbeiter - Vers- Verträge - Versicherung).

3.3.3 Das Datenstrukturdiagramm in Gestalt eines gerichteten Graphen nach Holl

Ein gerichteter Graph ist ebenfalls eine Darstellungsart, um Datenmodelle zu porträtieren. Es wird die eigene Terminologie (siehe Abschnitt 3.0) verwendet.

Mit Hilfe eines gerichteten Graphen werden die Attribute der einzelnen Entitätsmengen, sowie ihre Schlüssel- und Fremdschlüssel definiert (Schlüsseltabelle siehe Kapitel 3.2). Eine vorangegangene Datenanalyse unterstützt eine optimale Feindatenmodellierung, die in Abschnitt 4 dargestellt wird.

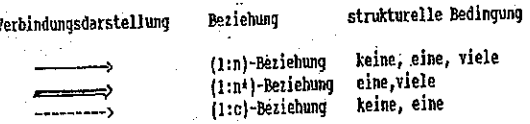
Das Modell weist folgende grafische Elemente auf.



Die dargestellte Verbindung zwischen zwei Entitätsmengen gibt Auskunft über

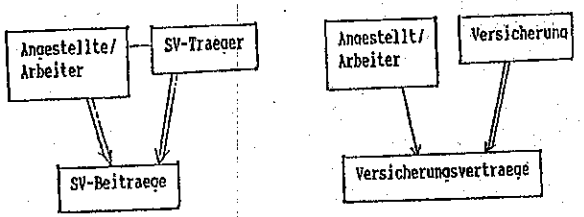
- die Kardinalsverhältnisse
- die strukturellen Bedingungen

einer Beziehung.



A.14

Zusatzregelung für das Kardinalsverhältnis - (n:m)-Beziehung



Darstellung einer (n:m)-Beziehung

Darstellung einer gekoppelten (1:n)-Beziehungen

Diese beiden Beziehungskomplexe charakterisieren eine (n:m)-Beziehung, d.h. eine relationale Abbildung, und eine sog. "gekoppelte" (1:n)-Beziehung, d.h. zwei funktionale Abbildungen. Eine eindeutige Darstellung wird durch die Verbindung der Ausgangsentitäten einer (n:m)-Beziehung erreicht.

Die Beschriftung der Verbindungspfeile mit den sog. Joinattributen gibt über die semantische Verbindung zweier Entitätsmengen Auskunft. Joinattribute sind solche Schlüsselattribute, die aufgrund ihrer Beziehung in anderen Tabellen als Fremdschlüssel oder wiederum als Hauptschlüssel fungieren.

Für die "Personaldatenbank" bietet sich der gerichtete Graph in folgender Form an:

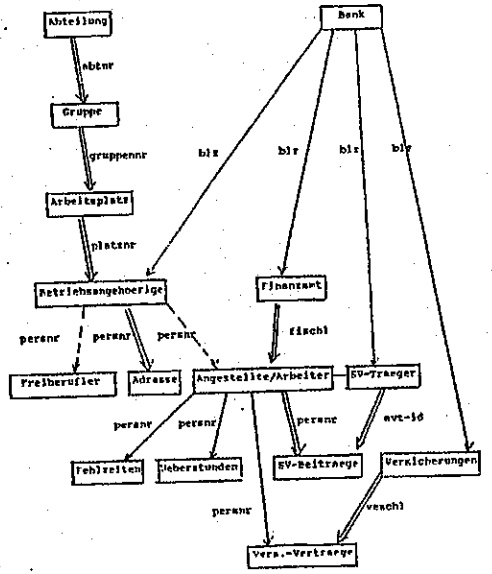


Abb. 3.10: gerichtetes Graphenmodell fuer die Personaldatenbank

Wertung:

Der gerichtete Graph besticht durch Übersichtlichkeit, ist leicht verständlich und eignet sich gut als Kommunikationsgrundlage. Zur detaillierten Darstellung von Datenmodellen bietet die Symbolik des gerichteten Graphen sehr gute Möglichkeiten. Die einzelnen Kardinalsverhältnisse sowie die strukturellen Bedingungen lassen sich eindeutig darstellen, es erfolgt ein Verschmelzen von Kardinalsverhältnissen und strukturellen Bedingungen in der Symbolik. Der gerichtete Graph erreicht eine akzeptable Transparenz. Es zeigt, im Gegensatz zu den bisher vorgestellten DSD, die Joinattribute bzw. Attributkombinationen auf, über die die Verbindung zwischen zwei Entitätsmengen realisiert wird.

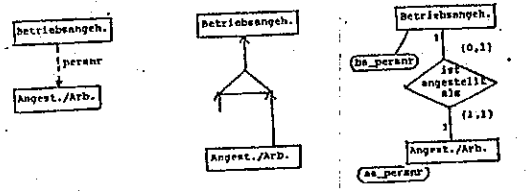
1.15

Resume:

Um einen DSD-Typ auswählen zu können, der für die Problematik - Ableiten von Joinwegen aus dem Relationennetz einer vorliegenden Datenbank - am geeignetsten erscheint, werden die verschiedenen Darstellungsweisen unter dem Aspekt der Informationsdetaillierung und Übersichtlichkeit verglichen.

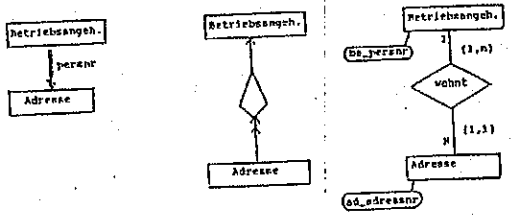
Die (1:c)-Beziehung

gerichteter Graph Ortner-Diagramm ERD



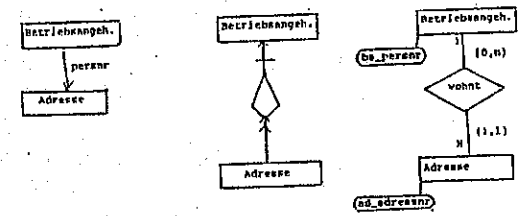
Die (1:n*)-Beziehung

gerichteter Graph Ortner-Diagramm ERD



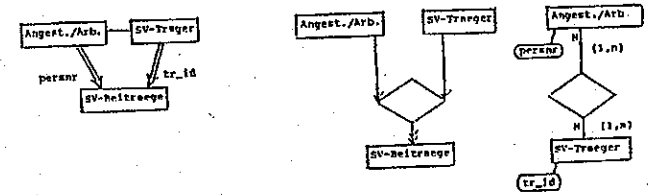
Die (1:n)-Beziehung

gerichteter Graph Ortner-Diagramm ERD



Die (n:m)-Beziehung

gerichteter Graph Ortner-Diagramm ERD



Insgesamt betrachtet, gestaltet sich das direkte Ableiten eines Joinweges aus dem gerichteten Graphen bzgl. der Automatisierung als sehr vorteilhaft. Entitätsmengen, die über eine weite Verbindung kommunizieren, sind mühelos zu überschauen, der Joinweg additiv zusammensetzbar. Dieser Detaillierungsgrad bei der Darstellung von Informationen wird von keinem anderen DSD-Typ erreicht.
Die Verschmelzung der Darstellung von Kardinalsverhältnissen und strukturellen Bedingungen in der Pfeilsymbolik des gerichteten DSD-Typs erhöht die Übersichtlichkeit und Transparenz. Diese Informationen werden zwar von allen DSD-Typen unterstützt, jedoch leidet die Übersichtlichkeit unter den komplizierten Darstellungsvorschriften.

A.16

Kardinalsverhältnisse :

Ortner : durch explizite Symbolik fuer Inklusion, Aggregation und Konnexion
und durch die Pfeilverbindungen
ERD : durch eindeutige Beschriftung der Verbindungskanten

strukturelle Bedingungen :

Ortner : durch die senkrechten Striche auf den Pfeilverbindungen
ERD : durch eindeutige Beschriftung der Verbindungskanten

Der gerichtete Graph weist mit Abstand den höchsten Detaillierungsgrad auf (direktes Ableiten von Joinwegen möglich). Eine weitere Stärke dieses DSD-Typs liegt in der übersichtlichen, prägnanten Darstellungsweise. Aus diesem Grunde erscheint der gerichtete Graph nach Holl als das geeignetste Darstellungsmittel.

4. Die relationale Organisation der Beispieldatenbank

Unter relationaler Organisation versteht man die Umsetzung des Grobdatenmodells in eine äquivalente Relationenstruktur. Jeder Entitätsmenge des gerichteten Graphen wird auf eine Relation (Tabelle) abgebildet.

4.1 Die Feindatenmodellierung

Mit Hilfe einer Informations- bzw. Funktionsanalyse werden die notwendigen Daten beschrieben. Durch stufenweise Verfeinerung des Grobdatenmodells erklärt man die abgeleiteten Tabellen, d.h. die einzelnen Attribute werden nach logischen und funktionellen Gesichtspunkten den Tabellen zugeordnet.

Die Ausführung dieser Punkte aufgrund des gerichteten Graphen für die Beispieldatenbank "Personalverwaltung" resultiert in einer Relationenstruktur, die ausschnittweise in der Abb. 4.2 grafisch dargestellt und in Tabellenform vollständig erhoben wurde. Dabei sind folgende Konventionen zu beachten.

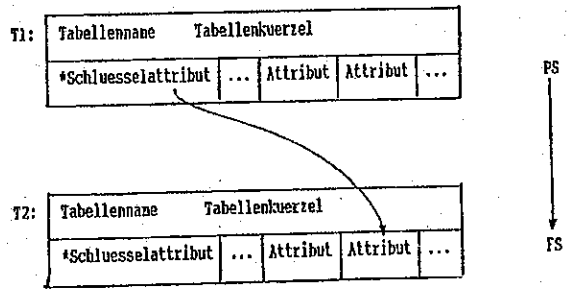


Abb. 4.1: Darstellungsweise des Relationennetze

4.1

Die Tabellen T1, T2 zeigen die Abbildung eines Objekttyps in seine Relation. Die mit einem "*" gekennzeichneten Attribute identifizieren die Primärschlüsselattribute. Sie dienen zur Unterscheidung einzelner Datensätze (Objekte) innerhalb einer Relation. Die Verbindung zwischen zwei Attributen verschiedener Tabellen symbolisiert die Übernahme eines eindeutigen Schlüssels in eine hierarchisch untergeordnete Tabelle (Fremdschlüssel) - dadurch wird auch der Joinweg zwischen zwei benachbarten Tabellen eindeutig definiert. Es entsteht ein sog. Relationennetz.

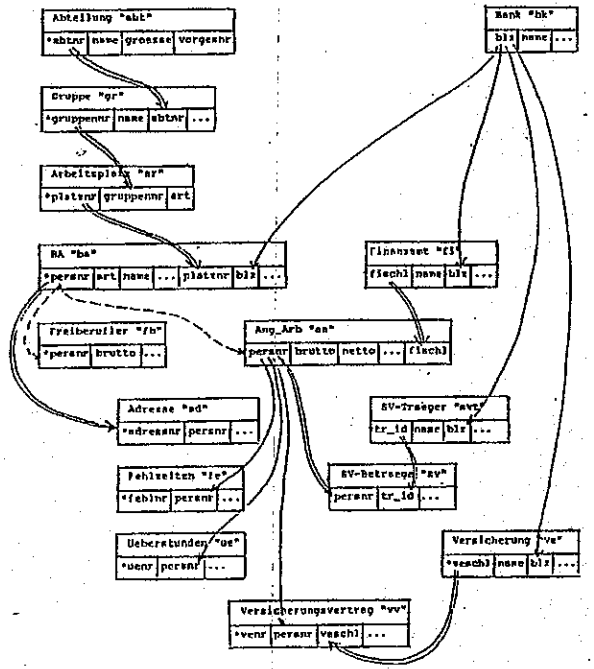


Abb. 4.2 Das Relationennetz der Beispieldatenbank

Namenskonvention:

Relationennamen kürzt man in geeigneter Weise ab. Dem Attributnamen wird das Tabellenkürzel seiner Tabelle vorangestellt, um bereits aus dem Attributnamen die Tabellenzugehörigkeit zu erkennen, z.B.:

Tabelle "abteilung"
Tabellenkürzel "abt"
==> Attribute abt_abtnr
abt_name
abt_groesse ...

Bei der Vergabe der Tabellen- bzw. Attributnamen beachtet man die vom eingesetzten DBMS (INGRES) festgelegten Längengrenzen. Maximal zwölf Zeichen pro Namen sind zulässig. Dabei entstehen oft sehr starke Verkürzungen, die in Problemfällen einer Erläuterung bedürfen. Aus diesem Grunde wurde die tabellarische Erhebung der Relationenstruktur um eine Attributbeschreibungskomponente, sowie Wertebereich für jedes Attribut erweitert. Die vollständige tabellarische Erhebung der Relationenstruktur ist dem Anhang A4 - Feindatenmodellierung zu entnehmen.

A.18

4.2 Die Normalisierung der Relationen

Die Normalisierung von Relationen soll gezielt eine Verbesserung des bestehenden Datenmodells erreichen, sowie logische Widersprüche und Inkonsistenzen aufdecken. Das Datenmodell kann verfeinert werden (siehe Abschnitt 3.2).

Die folgenden Unterabschnitte sollen die Überprüfung und wenn nötig das Durchführen der einzelnen Normalisierungsschritte für die erzeugte Relationenstruktur nachvollziehen.

4.2.1 Die erste Normalform (1NF)

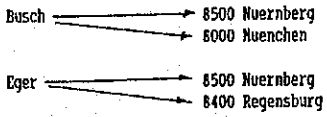
Regel: Alle Attribute eines Tupels (Tabelle) dürfen nur einfache Ausprägungen besitzen, derartige Wiederholgruppen müssen durch Aufspalten der Relation in zwei entfernt werden.

Man überprüfe für jede Tabelle, ob die Attribute der darin vorkommenden Datensätze jeweils nur einen Wert annehmen können. Ein Negativbeispiel läßt sich durch die Relation "BA-ADRESSE" leicht konstruieren.

Tabelle: BA-ADRESSE

persnr	name	Vorname	...	PLZ	Ort	Strasse
001	Busch	Marion		8500	Nuernberg	Seitzstr.
001	Busch	Marion		8000	Muenchen	Humboldweg
002	Eger	Bernd		8500	Nuernberg	Ludwigstr.
002	Eger	Bernd		8400	Regensburg	Hartistr.

An der Tabelle BA-ADRESSE wird das Auftreten von Wiederholgruppen illustriert, d.h. die Betriebsangehörigen "Busch" und "Eger" sind bei mehreren Wohnsitzen gemeldet. Eine Wiederholgruppe signalisiert daher eine (1:n)-Beziehung.



Durch Aufspalten der Tabelle BA-ADRESSE in BA und ADRESSE enfernt man solche Mehrfachausprägungen und stellt sie getrennt dar. Der eindeutig identifizierende Schlüssel (hier persnr) wird in die Tabelle ADRESSE als Fremdschlüssel übernommen - analog der Schlüsselvergabe bei einer (1:n)-Beziehungen (siehe Abschnitt 3.2). Die Tabelle ADRESSE erhält als Primärschlüssel einen Einfachschlüssel (hier: "adressnr").

Tabelle BA

persnr	name	vorname	...
001	Busch	Marion	
002	Eger	Bernd	

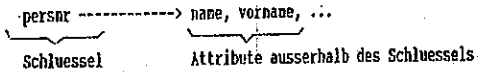
Tabelle Adresse

adressnr	persnr	PLZ	Ort	...
1	001	8500	Nuernberg	
2	001	8000	Muenchen	
3	002	8500	Nuernberg	
4	002	8400	Regensb.	

4.2.2 Die zeite Normalform (2NF)

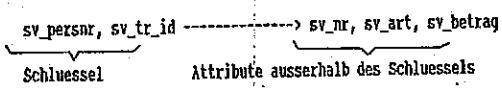
Regel: Alle, nicht dem Schlüssel A angehörenden Attribute B eines Tupels müssen vom gesamten Schlüssel A abhängen, sie dürfen nicht von einem Teil des zusammengesetzten Schlüssels abhängen. Man spricht auch von "voller" funktionaler Abhängigkeit, d.h. ein Attribut B ist dann von einem Schlüssel A oder einer Schlüsselattributkombination A "voll" funktional abhängig, wenn von A auf B geschlossen werden kann, z.B.

1.24



Der zweite Normalisierungsschritt wird nur für Relationen mit zusammengesetztem Schlüssel durchgeführt. Es handelt sich daher um Tabellen, die aus einer (n:m)-Beziehung resultieren.

Für das Datenmodell "Personalverwaltung" ist nur die Überprüfung der Relation "SV-Beiträge" notwendig, da nur hier der Schlüssel eine Attributkombination darstellt.



Diese Relation liegt bereits in der 2NF vor, da die Nichtschlüsselattribute vom Gesamtschlüssel identifiziert werden und die (n:m)-Beziehung bereits bei der Datenmodellierung berücksichtigt wurde.

Ein Negativbeispiel läßt sich mit der Tabelle "SV_BEITRÄGE- SV_TRÄGER" darstellen.

Tabelle "SV_BEITRÄGE-SV_TRÄGER"

sv_persnr	sv_tr_id	sv_nr	sv_art	sv_betrag	sv_name

Dabei erkennt man eindeutig die Situation, daß sv_name funktional nur von sv_tr_id abhängt, d.h. von einem Teil des Schlüssels. Durch Aufspalten der Tabelle erreicht man "volle" funktionale Abhängigkeit.

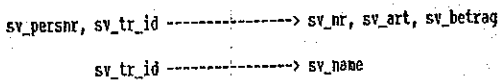


Tabelle "SV_BEITRÄGE"

sv_persnr	sv_tr_id	sv_nr	sv_art	sv_betrag

Tabelle "SV_TRÄGER"

sv_tr_id	sv_name

4.2.3 Die dritte Normalform (3NF)

Regel: Alle Attribute eines Tupels sind nur noch von seinem Schlüssel abhängig. Nicht erlaubt sind Abhängigkeiten zwischen Nichtschlüsselfeldern, d.h. transitive Abhängigkeiten.

Eine transitive Abhängigkeit läßt sich als eine (1:1)-Beziehung zwischen Nichtschlüsselfeldern darstellen, wobei ein Attribut dieser (1:1)-Beziehung in keinem direkten Zusammenhang zum Primärschlüsselattribut der Relation steht.

Dies sei anhand des Beispiels "BETRIEBSANGEHÖRIGE-BANK" gezeigt.

Tabelle "BETRIEBSANGEHÖRIGE-BANK"

persnr	name	...	kto	blz	bankname	...

Diese Tabelle liegt nicht in der dritten Normalform vor. Man betrachte folgende Situationen.

1.20

persnr ----------> blz ----------> bankname

persnr ----------> blz ----------> kto

Es zeigt sich eindeutig die (1:1)-Beziehung zwischen den Attributen "blz" und "bankname". Das Attribut "bankname" steht in keinem direkten Zusammenhang zum Schlüssel "persnr". Daraus folgt eine transitive Abhängigkeit.

Es liegt jedoch keine (1:1)-Beziehung zwischen den Attributen "blz" und "kto" vor. Einer Bankleitzahl ist nicht eindeutig eine Kontonummer zugeordnet. Vielmehr wird zwischen diesen beiden Attributen eine (1:n)-Beziehung definiert.

Wurde eine transitive Abhängigkeit entdeckt, erreicht man die 3NF dadurch, daß man die an der (1:1)-Beziehung beteiligten Attribute isoliert und eine eigenständige Tabelle bildet.

Tabelle "BETRIEBSANGEHÖRIGER"

persnr	name	...	kto	blz	...

Tabelle "BANK"

blz	bankname	...

Eine detaillierte Datenanalyse minimiert bereits das Auftreten transitiver Abhängigkeiten, indem Attribute nach logischen Gesichtspunkten zusammengefaßt werden. Bezüglich des angeführten Beispiels hat der Bankname nichts mit dem Betriebsangehörigen gemein, er müßte schon in diesem Schritt der Tabelle "BANK" zugeordnet werden.

Eine Diskussion daruüber, in welcher Normalform das verwendete Datenmodell "Personalverwaltung" vorliegt, erfolgt in Abschnitt 5.

5. Untersuchung von Datenmodellen

Motivation:

- Untersuchung des DMs allgemein, um die Zusammenhaenge zwischen DM und den darin dargestellten moeglichen Joinwegen zu beschreiben.
- Entwicklung eines zyklenfreien DMs in dem zu jeder Relationenkombination eindeutig ein Joinweg existiert
- Untersuchung nicht zyklenfreier DMe, deren Relationenkombinationen mehrere Joinwege zugeordnet werden koennen
- Loesungsansaetze fuer zyklenfreie und nicht zyklenfreie DMe

Definition eindeutig zugeordneter Joinwege:

Für eine spätere Anwendung soll von jeder Relation A zu jeder anderen Relation E einer vorgebenen Datenbankstruktur jeweils genau eine Joinverbindung existieren. Zur Laufzeit könnte sonst nicht festgestellt werden, welcher Joinweg der richtige ist. Jeder im DM möglichen Relationenverbindung (A ... E) muß eindeutig ein Joinweg zugeordnet sein.

12.1

5.1 Entwicklung zyklenfreier Datenmodelle auf der Basis der Normalisierung

Joinwege können nur in einem zyklenfreien Graphen eindeutig einer Relationenkombination zugeordnet sein, - denn Zyklen verursachen ein Vielwege-Problem, d.h. in einem nicht zyklenfreien Graphen gibt es mehrere Joinwege für eine bestimmte Relationenkombination.

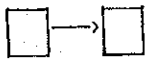
Es wird nun versucht, eine allgemeine Datenanalysemethodik zu finden, die zu DSDen führt, in denen Zyklenfreiheit garantiert und somit jeder Relationenkombination (von A nach E) eindeutig ein Joinweg zugeordnet werden kann.

Zusammenhänge zwischen Normalisierung und Zyklenfreiheit eines Datenmodells

Ausgegangen wird dabei von einer unnormalisierten Datenstruktur (UNF), d.h. das DM besteht aus einer einzigen Gesamtrelation. Es wird gezeigt, wie die einzelnen Normalisierungsschritte die UNF verändern, bzw. wie ein allgemeingültiges, in der 3NF vorliegendes DM gestaltet sein muß.

Folgende Symbolik wird in den Graphendarstellungen verwendet:

: Relation in DSD



: Verbindung zwischen zwei Relationen
wobei gilt: PS -----> FS

1) Ausgangssituation: UNF

Abb. 5.1: Darstellung eine UNF in einen DSD

2) erster Normalisierungsschritt (s. Abschnitt 4.2.1)

Die 1NF sieht das Entfernen von Wiederholgruppen vor. Eine Wiederholgruppe signalisiert entweder eine (1:n)-/(1:n*)-Beziehung oder eine (n:m)-Beziehung.
Bei einer (1:n)-/(1:n*)-Beziehung muß die Ausgansrelation (0) in zwei Tabellen aufgespalten werden. Die Attribute einer Wiederholgruppe werden isoliert in einer eigenen Tabelle dargestellt. Eine (n:m)-Beziehung wird mit zwei (1:n)-/(1:n*)-Beziehung im DSD dargestellt. Deshalb muß die Ausgangsrelation einer (n:m)-Beziehung in drei Tabellen aufgespalten werden. Die Aufspaltung der Ausgangsrelation erfolgt in der 1NF (Auflösung der ersten (1:n)-/(1:n*)-Beziehung). Die Auflösung zweiten (1:n)-/(1:n*)-Beziehung wird in der 2NF berücksichtigt.
In eine Relation, die Attribute einer Wiederholgruppe darstellen, erhalten einen eigenen Primärschlüssel, der Primärschlüssel der Ausgangsrelation wird als Fremdschlüssel übernommen. Der erste Normalisierungsschritt wird solange wiederholt, bis alle (1:n)-/(1:n*)- Beziehungen aufgelöst sind.

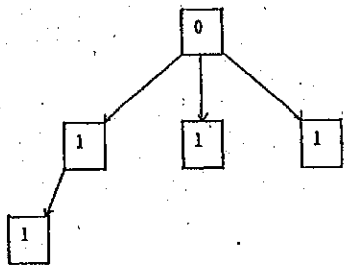
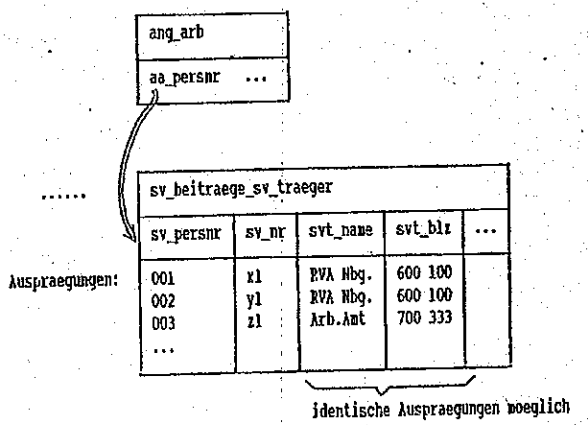


Abb. 5.2: Darstellung eines DMs der 1NF in einen DSD

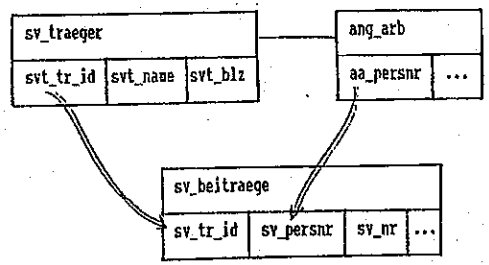
3) zweiter Normalisierungsschritt (s. Abschnitt 4.2.2)

Die 2NF verlangt "volle" funktionale Abhängigkeit vom Schlüssel. Sie ist nur für Relationen mit zusammengesetztem Schlüssel relevant, also für Tabellen, die aus einer (n:m)-Beziehung resultieren. Bei der Durchführung des zweiten Normalisierungsschrittes werden Relationen einer (n:m)-Beziehung vollständig aufgespalten, d.h. die zweiten (1:n)-/(1:n*)-Beziehungen werden berücksichtigt.

Man erkennt diese (1:n)-/(1:n*)-Beziehungen daran, daß Ausprägungen einzelner Attributkombinationen einer Relation - entstanden beim ersten Normalisierungsschritt - identisch sind, z.B.:



Attributkombinationen einer Relation, die beim ersten Normalisierungsschritt entstanden sind, lassen identische Ausprägungen zu, d.h. sie können n-mal vorkommen. Eine solche Attributkombination muß isoliert werden. Sie wird in einer eigenen Tabelle dargestellt, die einen identifizierenden Schlüssel (hier: "svt_tr_id") erhält. Der identifizierende Schlüssel wird mit Primär- und Fremdschlüsselfunktion in die Relation - entstanden bei der 1NF - übernommen (siehe Abschnitt 3.1).



Im DM entstehen dabei neue Wurzeln :

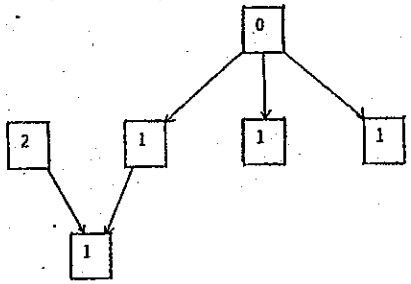


Abb. 5.3: Darstellung eines DM der 2NF in einem DSD

1.23

4) dritter Normalisierungsschritt (s. Abschnitt 4.2.3)

Die 3NF verbietet transitive Abhängigkeiten, d.h. (1:1)- Beziehungen zwischen Nichtschlüsselfeldern. Der dritte Normalisierungsschritt führt zur Isolation der an einer (1:1)- Beziehung beteiligten Attribute (neue Relation). Der Schlüssel dieser neuen Relation fungiert in der bisherigen Tabelle als Fremdschlüssel. Im Datenmodell entstehen dabei neue "Wurzeln":

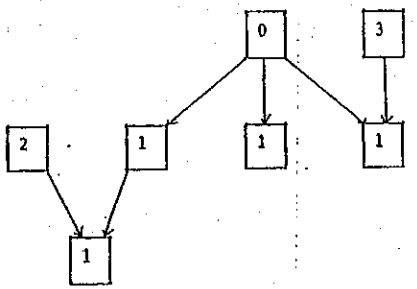


Abb. 5.4: Darstellung eines DM der 3NF in einem DSD

Ein DSD, das ein Datenmodell in der 3NF abbildet, stellt einen zyklenfreien gerichteten und zusammenhaengenden Graphen dar, fuer den eindeutige Joinwege, d.h. genau ein Joinweg fuer jede Relationenkombination, festgestellt werden koennen.

Es stellt sich nun die Frage, warum das DM "Personalverwaltung", Zyklen enthält. Zu diesem Zweck wird ein zyklenfreies DM "Personalverwaltung" in einem gerichteten Graphen (ohne Angabe der Joinfelder) dargestellt:

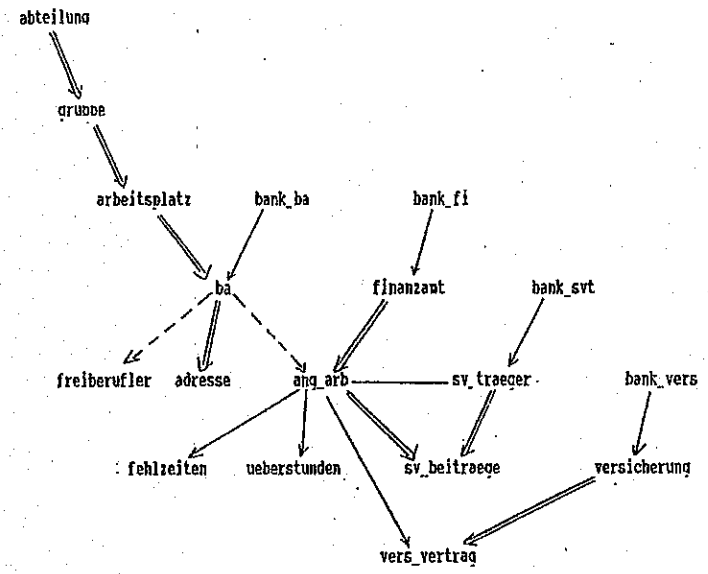


Abb. 5.5 : Darstellung des zyklenfreien DSD "Personalverwaltung" (exakt 3NF)

Data-base (ACM, New York) 1 (1969) 4-10

1.25

DATA STRUCTURE DIAGRAMS

By Charles W. Bachman

Successful communication of ideas has been and will continue to be a limiting factor in man's endeavors to survive and to better his life. The invention of algebra, essentially a graphic technique for communicating truths with respect to classes of arithmetic statements, broke the bond that slowed the development of mathematics.

Whereas "12+13=25" and "3+7=10" and "14+(-2)=12" are arithmetic statements, "a+b=c" is an algebraic statement. In particular, it is an algebraic statement controlling an entire class of arithmetic statements such as those listed.

Data Structure Diagrams

The Data Structure Diagram is also a graphic technique. It is based on a type of notation dealing with classes—specifically, with classes of entities and the classes of sets that relate them. For example, individual people and automobiles are entities. When they are taken collectively, they make two quite different classes of entities. On the other hand, all the automobiles belonging to a particular person constitute a set of entities that are subordinate to the owner entity.

The Data Structure Diagram has been used fruitfully over a period of five years by a limited but rapidly growing audience. This audience (where the technique originated) consists of the users of General Electric's Integrated Data Store (I-D-S) data management system. I-D-S employs language statements that directly support the relationships implied by the Data Structure Diagrams. The technique is now being used to study, document, and communicate information structures, even in those cases where no mechanized implementation is intended. The purpose of this paper is to document the technique of data structure diagramming so that it may be studied, evaluated, and put to work where it appears useful.

Definitions

Four terms: *entity, entity class, entity set,* and *set class* are central to the understanding of Data Structure Diagrams. This text will use the term *entity* to mean a particular object being considered; the term *entity class* will mean an entire group of entities which are sufficiently similar, in terms of the attributes that describe them, to be considered collectively. Many different entity classes may exist. The text will use the term *entity set* to mean a different kind of entity grouping—one that associates a group of entities of one entity class with one entity of a different entity class in a subordinate relationship. The concepts of entity class and entity set are independent of each other and can be thought of as being at right angles or orthogonal. Figure 1 illustrates this point.

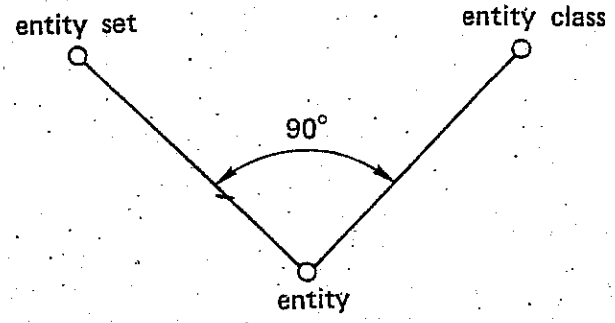


Figure 1

The term *set class* will be used in the text to mean an entire group of entity sets which are sufficiently similar, in terms of the attributes that describe them, to be considered collectively. Specifically, it is limited to those groups of sets in which the same entity-to-entity subordinate relationship exists. Figure 2 expands on Figure 1 to put all four of these terms into a spatial relationship.

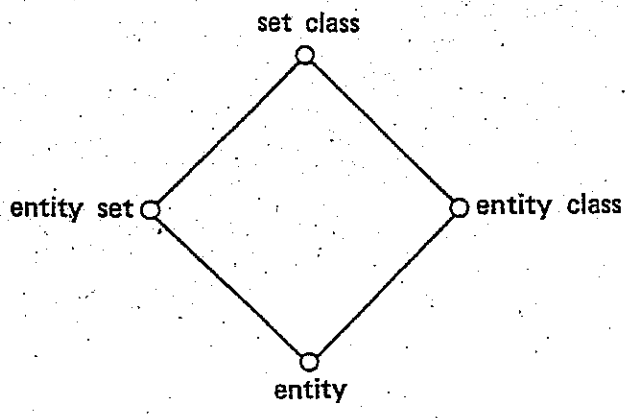


Figure 2

Many different *set classes* may exist. For example, the entities that we might consider in a management information system are the employees and the departments. All the employees in the company, when considered together, would make one entity class, while all the departments would make another entity class. Although the departments and employees may be considered independently of each other for some purposes, the relationship between the group of employees who work for the same department and that department may also be very important. Insofar as a department has a set of employees currently assigned to it, these employees can legitimately be considered as subordinate entities or sub-entities of that department. Each department is considered to be the *owner* of the set in which its employees are the *members.* When all of these

CHARLES W. BACHMAN is Manager, Applications Technology, for General Electric, Phoenix, Arizona. He is the creator of G.E.'s Integrated Data Store (I-D-S), a generalized data storage and retrieval system. He is also a member of the COBOL Data Base Task Force.

sets of employees are considered collectively they constitute a set class.

In a like manner, if employees, as an entity class, were considered in conjunction with their spouses and children, which comprise yet another entity class, then a set class could be established on the basis of the sets with employee entities as owners and their spouse and children as members. The concept of owner and member, the one owner to many member ratio, and the fact that these may be treated on a class basis, are central to the purpose and graphics of the Data Structure Diagram.

Graphic Symbols

The Data Structure Diagram technique uses two basic graphic symbols: the block, to represent an entity class; and the arrow, to represent a set class and to designate the roles of owner/member established by that set class. The arrow points from the entity class that owns the sets to the entity class that makes up the membership of the sets.

The diagram in Figure 3 states that an entity class exists and that an entity class name is to be assigned.

No information is implied as to how many entities make up the entity class. The only implication is that the entity class has been declared and is subject to such operations as may be defined.

CLASS OF ENTITIES

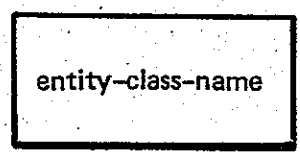


Figure 3

The diagram in Figure 4 states that two entity classes have been defined and that their entity class names are: "department" and "employee." If a particular company were being studied, there would be as many department entities and employee entities as that company had departments and people.

TWO CLASSES OF ENTITIES

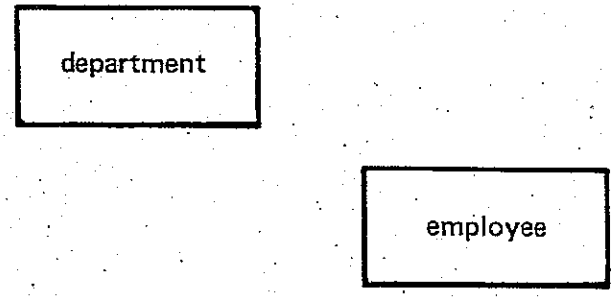


Figure 4

The diagram in Figure 5 states not only that two entity classes exist, but also that they are related by a set class named "assignment." The direction of the arrow is read to mean that each employee is a member of a set of employees that belong to a particular department, and further, that each department has such a set of employees.

SET ASSOCIATION OF ENTITIES

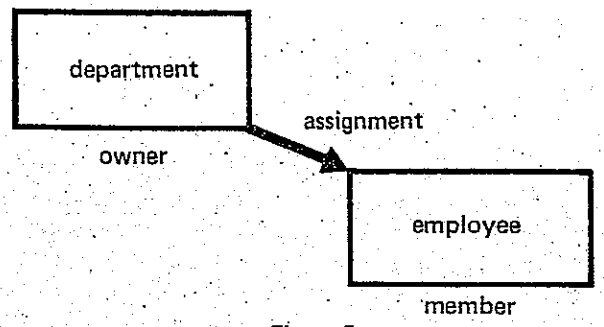


Figure 5

The Data Structure Diagram is topological in nature. Only the blocks, arrows, and names have meaning. The size, position, and proportion are selected for readability. Figure 6 is exactly equivalent to Figure 5, even if somewhat contorted.

TOPOLOGICAL GRAPHICS

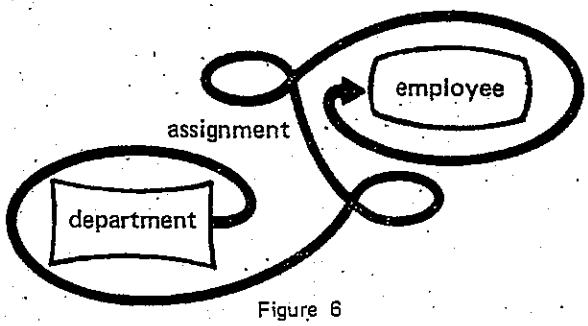


Figure 6

A Data Structure Diagram may contain as many blocks and arrows as necessary to establish the particular information structures under study. Any two entity classes may be associated as entity class/sub-entity class by zero, one, two, or more different set classes with the same or opposite ownership.

Hierarchies

The term hierarchy has been used rather ambiguously in the field of information technology. Data Structure Diagrams provide one possibility for non-ambiguous definition, i.e., an information hierarchy can be said to exist wherever there is a set-class relationship. Therefore, an information hierarchy exists whenever there are two or more levels of associated entity classes. Figure 7 integrates the department/employee association of Figure 5 with the employee/dependent association mentioned earlier to provide an example of a three-level hierarchy.

HIERARCHY

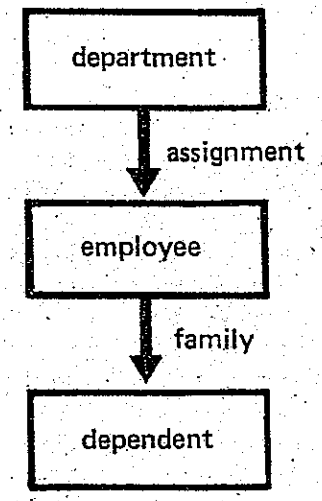
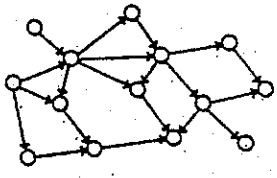


Figure 7

Many actual structures can be modeled either as a hierarchy, network or tree. When the elements in a real world hierarchy are like entities (i.e.,all people, all organizations) and their reporting level is subject to change, then a network or tree structure may prove to be more satisfactory in modeling the situation than a hierarchical structure.

Networks

Many information models involve networks of information. PERT or CPM diagrams are examples of such networks. Another example is the "T" account double-entry accounting system in which every entry affects the debit side of one account and the credit side of another account. Figure 8 is a generalized picture of a network with nodes connected to each other in a directed sense, or as a directed graph.



NETWORK

Figure 8

The Data Structure Diagram that defines a network is seen in Figure 9.

SIMPLE NETWORK STRUCTURE

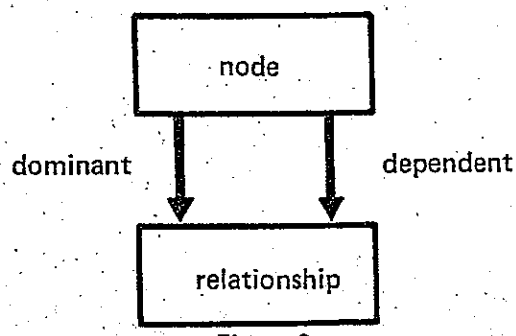


Figure 9

The two entity classes labeled "node" and "relationship" relate to the nodes and lines between nodes in Figure 8. By setting up a table of equivalences (Table 1), several different models which are network-oriented can be quickly defined. Please do not confuse the arrow direction in the network (Figure 8)—meaning the dominance of one node over another with the arrow direction in the network Data Structure Diagram (Figure 9) meaning an owner/member role.

APPLICATION	ENTITY CLASS NAME		SET CLASS NAME	
	NODE	RELATIONSHIP	DOMINANT	DEPENDENT
PERT/CPM	Event	Activity	Prior	Succeeding
GENERAL ACCOUNTING	Account	Transaction	Debit	Credit
PARTS LISTS	Material Item	Component	Call-Out	Where Used
GENEALOGICAL CHARTS	Subject	Relationship	Parent	Child
SUBROUTINE STRUCTURE	Subroutine	Call	Enter	Return
ORGANIZATION CHARTS	Organization	Component	Sub-Unit	Report-To

Table 1

The similarity of the PERT/CPM diagrams to a network is obvious. That of the general accounting model may be less obvious. But what are the transactions, except directed quantitative relationships between accounts (nodes); the trail balance should always be zero. Manufacturing parts lists consists of material items, which are made out of material items. Genealogical charts are similar to manufacturing parts lists except that each item is made from only two other things, its parents.

The interrelationship of a set of subroutines that call on each other is also a network because each subroutine may call many subroutines or be called by many subroutines.

Tree Structures

Organization charts usually are special cases of a network, the tree structure, in which each node has one

dependent relationship and many dominant relationships. I say usually, because military organizations are rife with situations where units are assigned one place for command purposes and another for rations and quarters. Figure 10 illustrates a tree structure.

TREE

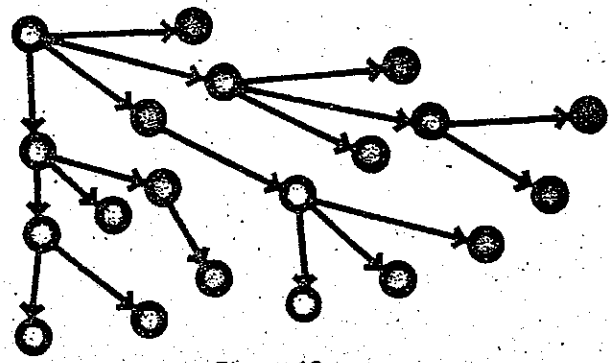


Figure 10

The Data Structure Diagram in Figure 9 is equally good for modeling a network or a tree. The Data Structure Diagram in Figure 11 is more specialized in that it supports a tree model but does not support a network.

TREE STRUCTURE

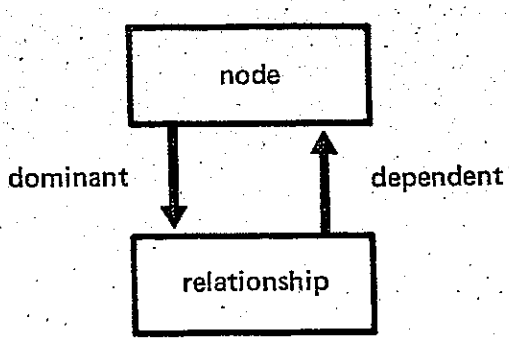


Figure 11

In the Tree Structure Diagram the owner/member relationship (arrow) of the "dependent" set class has been reversed. The freedom to make the reversal is based upon the fact that the tree allows only one or no relationships on the dependent side of the node. Modelling the tree with a Data Structure Diagram permits two options: (1) one relationship entity owning a one-member set of node entities (Figure 11), or (2) one node entity owning a one-member set of relationship entities (Figure 9). Therefore, the direction of the entity/sub-entity relationship is somewhat arbitrary. From the Data Structure Diagram in Figure 11, it is a short step in structure simplification to reach the diagram in Figure 12, which still represents the tree.

TREE STRUCTURE DIAGRAM

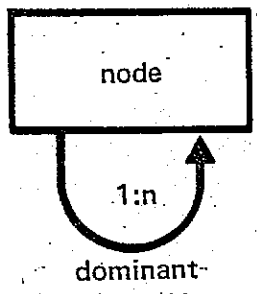


Figure 12

The "dependent" set had been limited by definition of a tree to a 1:1 relationship between the "node" and "relationship" entities. This was the fact which permitted reversing the dependent entity/sub-entity association in Figure 11 and still further supported merging the "node" entity class with the "relationship" entity class. The diagram simply illustrates that each "node" has a "dominant" relationship with other nodes that, in turn, are limited to one relationship on what is considered their "dependent" side.

The Data Structure Diagrams in Figures 11 and 12 create a chicken and egg situation. A member entity cannot exist until there is a set in which to insert it. In Figures 11 and 12 the "node" entity is both owner and member. Therefore, one such entity cannot exist alone unless it is its own owner. Two different structure solutions are available for this dilemma. These are the sometime member entity classes, described in the next section, and the alternate owner set classes, which will be discussed later in the text.

Sometime Member Entity Classes

When it is necessary to document a set class in which the member relationship may or may not exist, a dashed arrow is used. Figure 13 illustrates the graphic convention. Entities of the owner entity class always own a set of the set class specified. Entities of the class at the head of the arrow in Figure 13 individually may (or may not) be members.

SOMETIME MEMBERSHIP

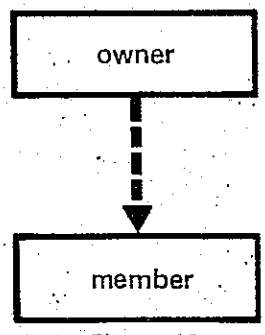


Figure 13

An example of a sometime member can be drawn from the banking industry, where demand deposit accounts entities have a sometime relationship with an overdrawn account entity. Whenever an account's balance is less than zero, the relationship is invoked until paid up. Otherwise, the relationship doesn't exist.

Compound Networks

We have just used some examples of simple networks to illustrate the usage of Data Structure Diagrams. By simple network, I mean networks of like entities, i.e., all events, or all "T" accounts. Compound networks are the result of the association of unlike entities within a network. One example of a compound network is developed by the author association between books and the people who wrote them. If you were to examine the books in any library, you would find that some books had one author while other books, especially in the technical and educational fields, had two and maybe three or four authors. If you considered the people who were authors of these books, you would find that some were authors of more than one book. Certain prolific authors would have many books to their credit. Therefore, the network created by the book/people/author relationship would consist of nodes for books (book entities), nodes for people (people entities), and a relationship between a book entity and a person entity that records authorship. Figure 14 is the Data Structure Diagram which represents this compound network.

COMPOUND NETWORK STRUCTURE

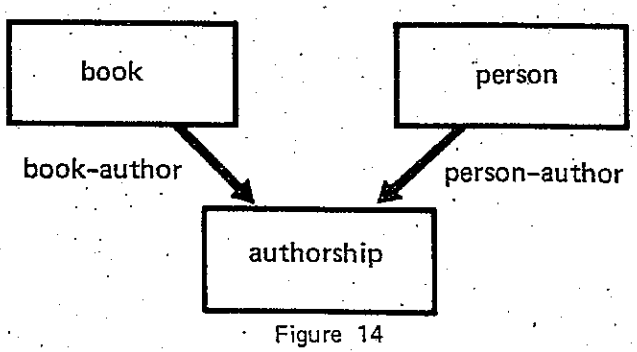


Figure 14

Information models of any complexity usually are compound networks in one or more ways. If our original department/employee model had represented a university, then the "employee" entity might have been the same entity class name as the "person" entity with a new association. Figure 15 combines the Data Structure Diagrams of Figure 14 and one similar to Figure 7 to create a more comprehensive information model.

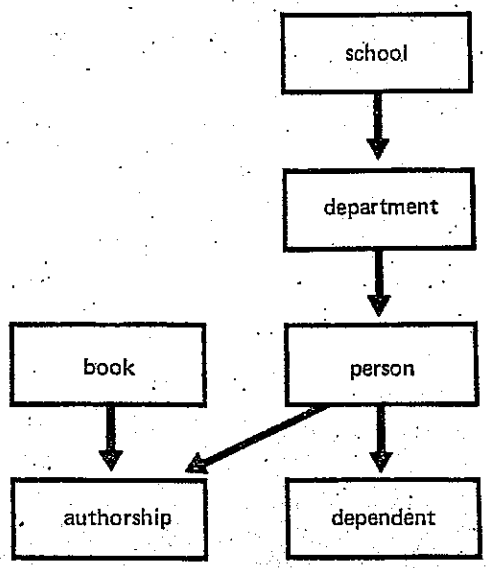


Figure 15

This model includes a school/department/person organization-oriented hierarchy, the person/dependent personnel-oriented hierarchy and the person/authorship/book compound (publish-or-perish) network. Given the necessary computer hardware and software, and access to any one entity in a specific data base created according to this Data Structure Diagram, then all other associated entities could be determined by moving through the sets. Both General Electric's Integrated Data Store (I-D-S) and General Motors' Associative Programming Language (APL) are software systems specifically designed and implemented to encourage and support the construction, maintenance, and use of data bases organized around such complex structures.

Multimember Set Classes

When it is necessary to document a set class with more than one class of entities in the role of member, a multi-headed arrow is used. Figure 16 illustrates the graphic convention.

MULTIMEMBER SET CLASSES

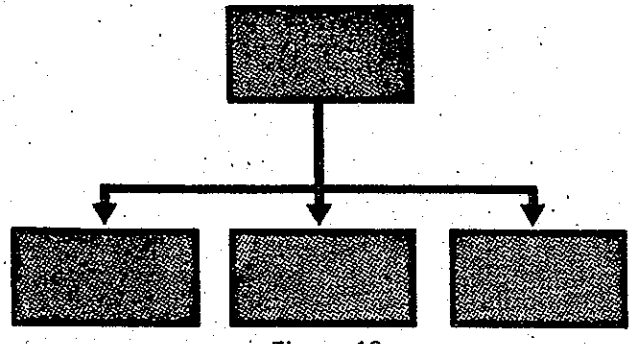


Figure 16

An example of multimember set classes can be drawn from the banking industry, where each customer may have several different types of business with the bank. Different entity classes are established for demand deposit accounts, savings accounts, loan accounts, and trust accounts. Figure 17 illustrates this data structure. With this structure, each customer can have multiple numbers of accounts of the same or different types.

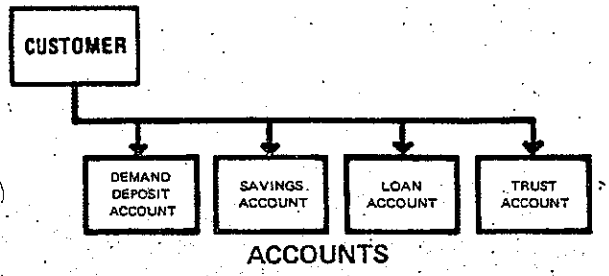


Figure 17

The need to provide for multicustomer associations with specific accounts, i.e., joint accounts, usually leads banks to work with a structure that includes joint account entities. Figure 18 shows this extension. With this data structure, each customer can have multiple numbers of accounts, and each account has one prime customer. In addition, each account may have any number of additional joint account customers. Thus, again a compound network is described.

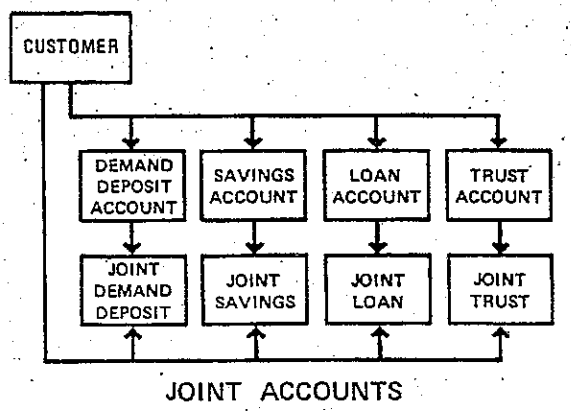


Figure 18

Alternate Owner Set Classes

When it is necessary to document a set class with potentially more than one class of entities in the role of owner, a multitail arrow is used. Figure 19 illustrates the graphic convention.

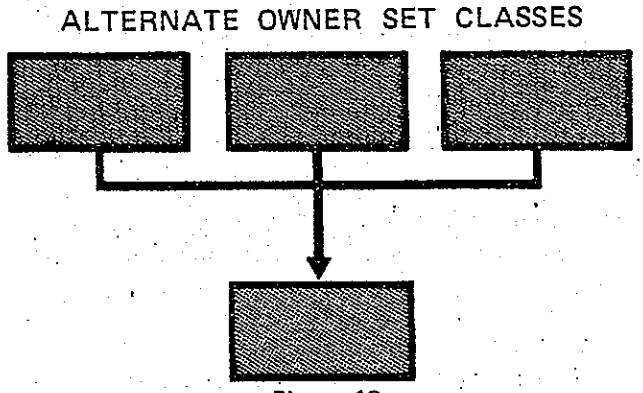


Figure 19

A particular set has only one owner. Each owner entity has its own set, but the set class name and the member entity classes of its set are the same regardless of which entity is the owner.

An example of alternate owner set classes can be drawn from the manufacturing industry, where a manufacturing order may be placed on the shop by an internal organization, a distributing organization, or a customer. For accounting and reporting purposes, different entity classes are established for each type of organization because different information must be maintained or because they may be involved in other and different set classes. However, from the shop's point of view, they are the "customer" for an order regardless of their other nature.

Another example relates back to the Tree Structure Diagram in which initiation of the tree posed a problem. Figure 20 is an alternate solution. If the "dominant" set class had alternate owners (either a primary node entity that is owner but not member or a secondary node entry that is both owner and member), then the problem is trunk of the tree. The corollary of this structure, if imposed on a data management system, is as follows, "If the primary node were removed, then all of the secondary nodes and thus the entire tree goes with it."

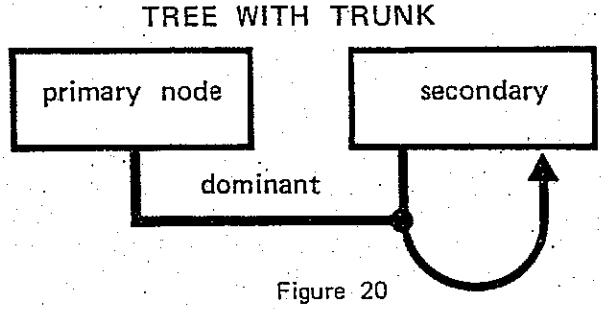


Figure 20

Complex Structures

Very large Data Structures Diagrams have been designed and used in the last five years in the design and implementation of various information systems. "Large" in this case is measured in terms of the number of entity classes and set classes. Figure 21 illustrates the Data Structure Diagram underlying one manufacturing information and control system.

MANUFACTURING STRUCTURE

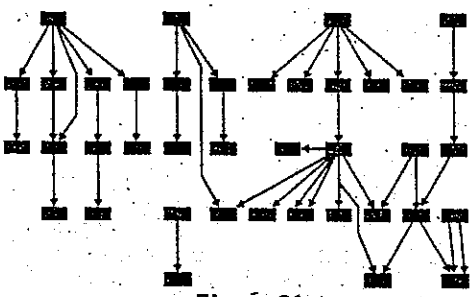


Figure 21

It has 39 entity classes and 38 set classes. Can you, in examining the diagram, find a five-level hierarchy? Two simple networks? Five compound networks? A simple network with an extra hierarchical level in one leg? Note that there are no tree structures. Without any rigorous definition, a complex structure is one composed of many entity classes and many set classes. Within a complex structure, we typically find multilevel hierarchies, simple networks, compound networks and trees.

Large Data Structure Diagrams, in terms of the number of elements, should be clearly distinguished from large data bases with many entities (records), which have been built in response to a Data Structure Diagram. Although each entity in a data base needs an entity class to define and control it, that one entity class may represent zero, one, ten thousand, or a million records in storage. The largest system in operation today contains 60 entity classes controlling over half a million data records. Larger systems are being installed.

Summary of Structure Types

Simple and compound networks are differentiated by the fact that one, the simple network, is constructed of node entities of a single entity class, while the other is constructed of node entities of several different entity classes. Trees and networks are differentiated by the fact that one, the tree, is constructed under the rule that each node has only one "dominant" node while the other is constructed with unlimited association between nodes. The two concepts: tree vs. networks, and simple vs. compound, are independent of each other and can be thought of as being at right angles or orthogonal. Figure 22 illustrates this point and brings the hierarchial structure into perspective. It is a compound tree, i.e. with more than one entity class as node and each node with only one dependent association.

COMPOUND NETWORK

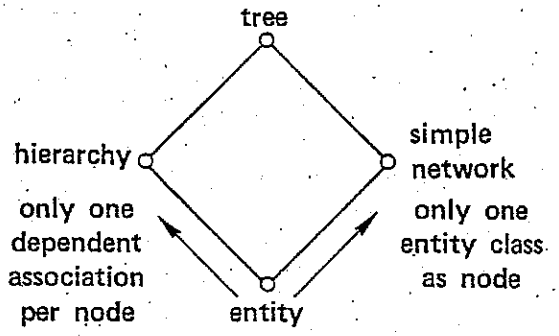


Figure 22

Summary

The Data Structure Diagrams, consisting of two kinds of elements (blocks representing entity classes and arrows representing set classes), have been defined. Examples have illustrated their application. Their practical usage in the design and study of mechanized information systems has been described. Two data management languages, I-D-S and APL, provide for the description and manipulation of data bases with the characteristics that are describable through the Data Structure Diagrams. In addition, there is a new Data Description Language and Data Manipulation Language currently being specified by the Data Base Task Group of the COBOL Programming Languages Committee. This offers promise for an industry standard data management language that would operate in conjunction with FORTRAN, ALGOL, PL/I, as well as COBOL allowing a DATA BASE to be built in one language while being accessed in yet another.

Bibliography

(1) Bachman, C. W., Williams, S. B., "A General Purpose Programming System For Random Access Memories," *Proceedings of the Fall Joint Computer Conference,* San Francisco, California. October 1964.

(2) Bachman, C. W., "Software For Random Access Processing" *Datamation,* April 1965.

(3) "Integrated Data Store, A New Concept in Data Management," AS-CPB-483A General Electric Information Systems Group, Phoenix, Arizona.

(4) Bachman, C. W., "Integrated Data Store Data Base Study" *Second Symposium on Computer-Centered Data Base Systems,* also available as CPB-481A General Electric Information System Group, Phoenix, Arizona.

(5) Dodd, G. G., "APL—A Language for Associative Data Handling in PL/I," Fall Joint Computer Conference, 1966.

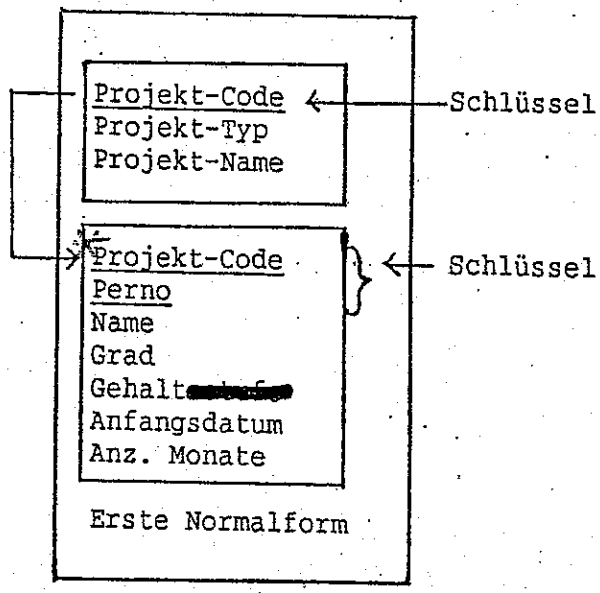
(6) "Report to the CODASYL COBOL Committee, January 1968. COBOL Extensions to Handle Data Base" prepared by Data Base Task Group.

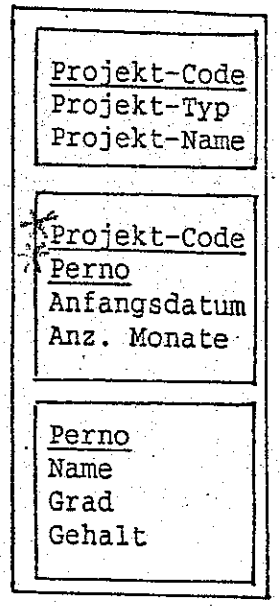
(7) "Data Description Language and Data Manipulation Language Report," April 1969. Prepared as a report to the CODASYL COBOL Programming Language Committee by the Data Base Task Group.

Projekt-Code
Projekt-Typ
Projekt-Name
Perno
Name
Grad
Gehalt
Anfangsdatum
Anfangsdatum
Anzahl Monate

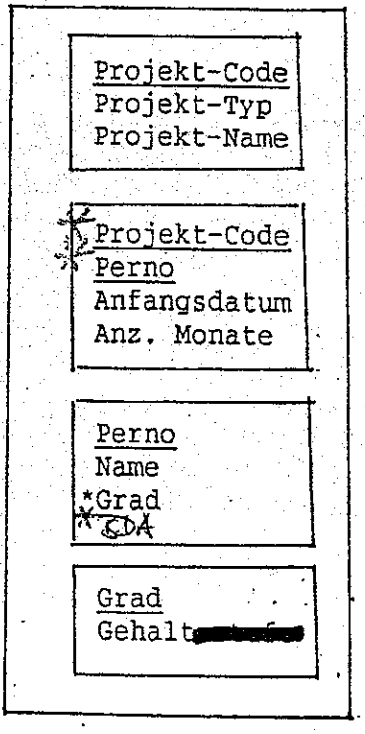
sich wiederholende Gruppe

Unnormalisierte Form



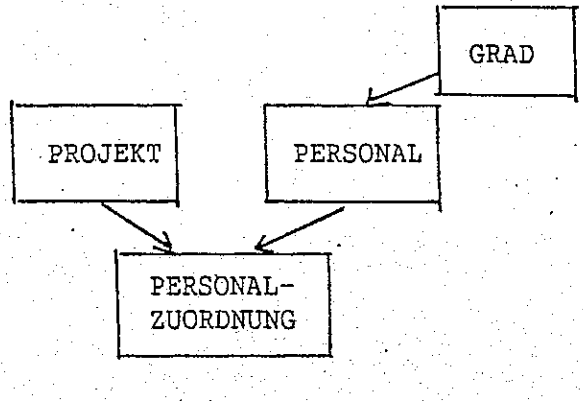


Zweite Normalform



dritten Normalform

Datenstrukturdiagramm



wenige einzelne
1
:
m
viele einzelne

EINZELHÄNDLER

1. D a t e n e l e m e n t e sammeln und objektbezogen im Datenkatalog vorgruppieren.

D a t e n s a m m l u n g

Artikel-Nummer
Artikel-Bezeichnung
Artikel-Gruppe
Farbe
Preis

Angebots-Lieferant, Nummer
Angebots-Lieferant, Name
Angebots-Lieferant, Telefon
Angebots-Lieferant, Preis
Angebots-Lieferant, Lieferzeit
} können mehrmals vorkommen!

Letzter Lieferant, Nummer
Letzter Lieferant, Name
Letzter Lieferant, Ort

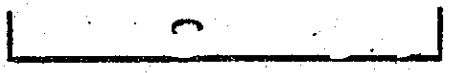
Verkäufer-Name
Verkaufsdatum
Verkaufsmenge
} können mehrmals vorkommen!

2. Bestimmung der Schlüssel für die Primärdatei

Der erste Schritt der eigentlichen Datenanalyse ist die Bestimmung des Datenfeldes bzw. der Kombination von Datenfeldern, die die vorliegende Sammlung von Daten (=Primärdatei) eindeutig identifiziert.

Dieses Feld bzw. diese Kombination von Feldern wird Schlüssel der Primärdatei genannt.

Die im letzten Schritt dargestellte Karteikarte, die ja gerade die Sammlung der Primärdaten darstellt, wird von dem Begriff



eindeutig identifiziert.

3. Herauslösen von Mehrfachfeldern

Der nächste Schritt der Datenanalyse ist das Herauslösen von Mehrfachfeldern. Diese werden mit einem zusammengesetzten Schlüssel in eine eigene Datei gebracht.

Der Schlüssel wird zusammengesetzt aus:

- Schlüssel der Primärdatei,
- Mehrfachfelder (so viele wie nötig sind, um den Begriff eindeutig zu machen).

Es entstehen damit so viele neue Tabellen oder Dateien wie es Gruppen von Mehrfachfeldern in der Primärdatei gibt.

In unserem Beispiel tauchen folgende Gruppen von Mehrfachfeldern in der Primärdatei auf:

-
-
-
-
-

Damit ergibt sich folgendes Resultat:

Artikel-Datei	Angebots-Datei	Verkaufs-Datei

4. Herauslösung von Feldern, die nur von einem Teil des Schlüssels abhängig sind.

Zur Vermeidung von Inkonsistenzen durch Datenveränderungen und zur Erleichterung des Pflegeaufwandes werden Elemente, die nur von einem Teil des Schlüssels abhängig sind, mit diesem als eigene Tabelle abgelegt. Der Teilschlüssel ist dann Schlüssel dieser neuen Tabelle.
Diese Maßnahme wird auch funktionale Zergliederung genannt.

In unserem Beispiel hängen die Felder

"_____" und "_____"

nur von dem Teil "_____" des zusammengesetzten Schlüssels ab.

Herauslösen und Ablegen in einer eigenen Datei ergibt folgendes Ergebnis:

5. Herauslösung von Feldern, die von einem Nicht-Schlüsselfeld abhängen.

Um unnötige Redundanzen zu vermeiden und um die Effizienz der Datenpflege zu optimieren, werden Datenelemente, die von Nicht-Schlüsselfeldern abhängen, aus der Tabelle entfernt und mit denjenigen, von denen sie abhängig sind, in einer anderen Tabelle abgelegt (transitive Zergliederung).

Die Felder

"_____" und "_____"

in der Artikeldatei hängen von dem Nicht-Schlüsselfeld "_____" ab.

Diese werden herausgelöst und müßten eigentlich mit dem Feld "_____" (als Schlüssel) in eine eigene Tabelle gebracht werden.

Da jedoch schon eine Lieferanten-Datei mit genau dem Schlüssel und dem Lieferanten-Namen existiert, erübrigt sich die Definition einer neuen Datei.

Resultat:

6. Herauslösung von Datenelementen mit sich wiederholenden Inhalten.

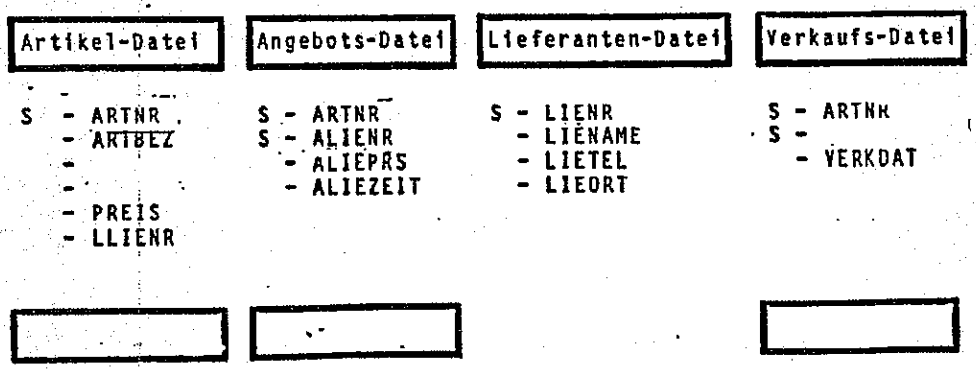
Datenelemente, deren Inhalt sich über eine Reihe von Objekten in einer Datei wiederholt, können mit einem neuen Schlüssel als eigene Tabelle angelegt werden. Anstelle des bisherigen Datenelementes wird der neue Schlüssel in der bisherigen Datei eingefügt, um eine Zuordnung sicherzustellen.

Dieser Schritt hilft, die Dateneingabe zu erleichtern, läßt leicht Plausibilitäten zu, ermöglicht eine Klassifizierung und kann mögliche Sortierungen berücksichtigen.

In unserem Beispiel bieten sich dazu die Felder

"_____", "_____" und "_____" an.

Ergebnis:



Bei der Herauslösung von Datenelementen mit sich wiederholenden Inhalten können auch die neuen Schlüssel in den bestehenden Tabellen hinzugefügt werden, um dann eine Herauslösung entsprechend Schritt 5 (transitive Zergliederung) zu erreichen.

2. Eine Fortbildungsinstitution will die Verwaltung von Prüfungsteilnehmern mit einer relationalen Datenbank abwickeln.
Es gelten die folgenden Beziehungen:

- Ein Teilnehmer muß mindestens eine, kann mehrere Prüfungen ablegen.
- Eine Firma kann mehrere Teilnehmer anmelden.
- Ein Dozent kann mehrere Prüfungen stellen.
- Eine Prüfung wird von einem Dozenten abgehalten.

a) Führen Sie ausgehend von folgenden Datenelementen (UNF) eine Datenanalyse bis zur 3NF mit dem Normalisierungskalkül durch.
Bezeichnen Sie bei den Tabellen jeder Normalform Primärschlüssel durch Unterstreichen und Fremdschlüssel durch *.
Finden Sie sinnvolle EDV-Namen für die Datenfelder in den 3NF-Tabellen.

Teilnehmer-Nummer
Teilnehmer-Name
Teilnehmer-Anrede
Teilnehmer-Privatanschrift

Prüfungs-Nummer
Prüfungs-Bezeichnung
Prüfungs-Ort
Prüfungs-Datum
Dozenten-Schlüssel
Dozenten-Name
Note
Platzziffer
Gesamtzahl der Prüfungsteilnehmer

} kann mehrmals vorkommen

Firmen-Schlüssel
Firmen-Bezeichnung
Firmen-Anschrift
Firmen-Telefon

b) Skizzieren Sie in Stichworten die Herleitung von 1NF (2 Zeilen), 2NF (1 Zeile) und 3NF (1 Zeile).

c) Stellen Sie die Abhängigkeiten der 3NF-Tabellen in einem Datenstrukturdiagramm als gerichteten Graph dar.

Aufgabe: Erstellung eines logischen Datenmodells

Ausgangsstruktur des konventionellen Systems:

	Artikel-Nr.	Artikel-Bez.	Bestand	
Wurzel	ANR	ABESCHR	BEST	Stufe 0

Auftrags-Nr.	Sonderfertigung (JA/NEIN)	Kunden-Nummer	Kundenpriorität	Anzahl	
AUFNR	S	KNR	KPR	ANZ	Stufe 1

Ein Betrieb bietet Artikel an, die eine Artikelnr. (ANR) und eine Beschreibung besitzen. Es ist ein Bestand vorhanden und mehrere Aufträge sind eingegangen (Wiederholungsgruppe). Die Aufträge haben eine Nummer, eine Angabe, ob Sonderanfertigung vorliegt oder nicht, die Kundennummer des Auftrages (KNR kann redundant existieren), die Kundenpriorität, d.h. ob ein Kunde bevorzugt beliefert wird, und die Bestellmenge. Definieren Sie ein unnormalisiertes Datenmodell und leiten Sie davon ausgehend über die FNF und SNF die TNF her.

Kundenpriorität: nicht auftragsabhängig
Sonderfertigung: auftragsabhängig

DATENBANK-DESIGN

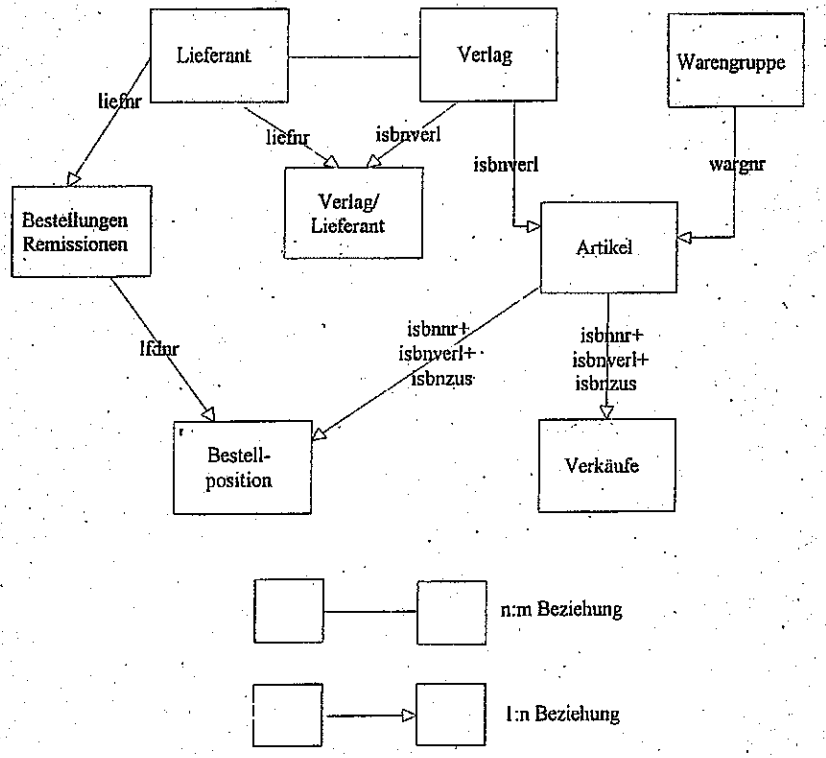
~~2.8.7~~ 2.7

Minimelten

VERARBEITUNGS-ANFORDERUNG	SUCH-BEGRIFF	AUSGABEN	
1. KUNDEN EINES VERTRETERS [Außendienst]	VERTRETER-Nr. (VNR)	ADRESSE DES VERTR.	(VAN)
		PROVISION	(PRO)
		KUNDEN-Nr	(KNR)
		KUNDEN-ANSCHRIFT	(KAN)
2. AUFTRÄGE EINES KUNDEN [Auftrags-verwaltung]	KUNDEN-Nr (KNR)	KUNDEN-ANSCHRIFT	(KAN)
		AUFTRAGS-Nr	(BNR)
		ARTIKEL-Nr	(ANR)
		ARTIKEL-BEZ	(ABE)
		BESTELLMENGE	(BEM)
		PREIS/DM	(PDM)
		LIEFERDATUM	(LDT)
3. OFFENE POSTEN EINES KUNDEN [Fibü]	KUNDEN-Nr (KNR)	KUNDEN-ANSCHRIFT	(KAN)
		BELEG-Nr	(BEL)
		KONTO-Nr	(KTO)
		DATUM	(DAT)
		BETRAG	(BET)
4. UMSÄTZE JE ARTIKEL [Unternehmer]	ARTIKEL-Nr (ANR)	ARTIKEL-BEZ	(ABE)
		PREIS/DM	(PDM)
		UMSÄTZE (ein Feld pro Monat)	(UMS)
5. AUFTRÄGE JE ARTIKEL [Produktion, Versand]	ARTIKEL-Nr (ANR)	ARTIKEL-BEZ	(ABE)
		BESTAND	(BES)
		AUFTRAGS-Nr	(BNR)
		BESTELLMENGE	(BEM)
		LIEFERDATUM	(LDT)

2.4.2. ERM

ERM-Diagramm Comicladen

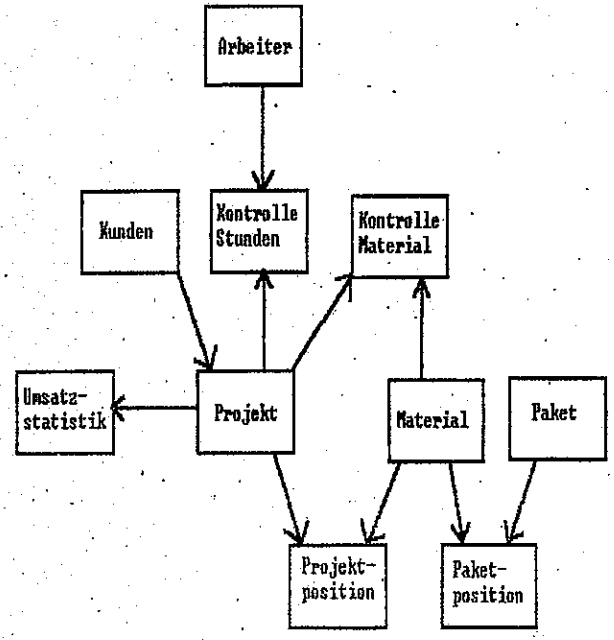


Beschreibung des ERM:

Es soll ein Überblick über die Tabellen und Besonderheiten gegeben werden. Die detaillierten Informationen sind im folgenden Unterkapitel zu finden.

Die Verwaltung der Rechnungsdaten wird im Neuentwurf nicht mitberücksichtigt, da davon ausgegangen wird, daß eine eigenständige Finanzbuchhaltung diese Aufgaben erfüllt.

2.2.2.2 Datenstrukturdiagramm Elektrohandwerk



Das Datenstrukturdiagramm des Elektrohandwerks unterscheidet sich von dem des Stukkateurhandwerks durch die Zuordnung des Materials zur Projektposition und nicht der Leistungs-tabelle.

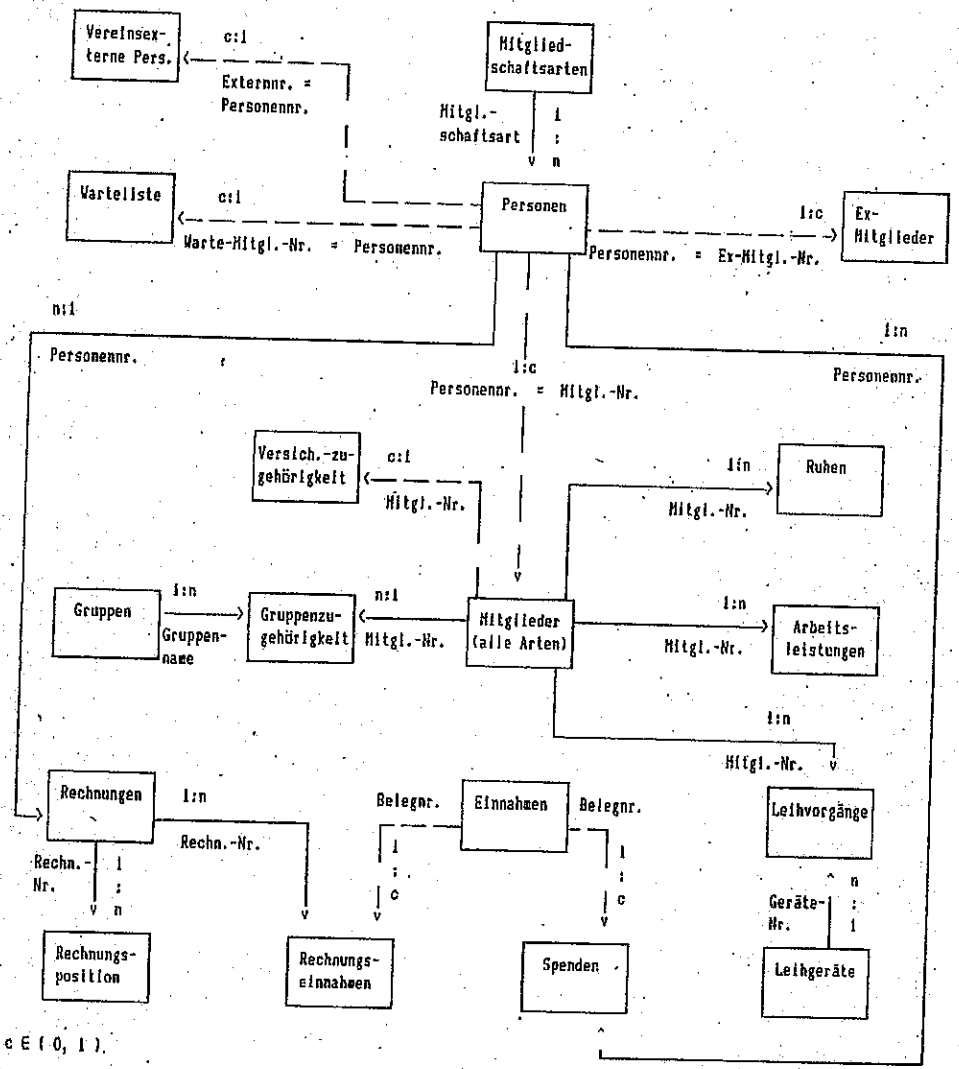
Dies kommt daher, daß der Elektriker anhand von Materialien kalkuliert und nicht mit Leistungen. Die Tabelle Leistungen wird nicht benötigt. Desgleichen besteht ein Paket nicht aus einzelnen Leistungen, sondern aus Materialien.

Zu erwähnen ist noch, daß bei der Beziehung Material-Projektposition die Materialnummer in der Tabelle Projektposition auftritt, da hier der Verweis auf die Materialtabelle genügt. Denn hier ist in verschiedenen Leistungsverzeichnissen die Bezeichnung eines Materiales immer dieselbe.

4.3. Feinkonzept

Vereinsverwaltung

4.3.1. Datenstrukturdiagramm



c ∈ { 0, 1 }.

4.9 Das ERM-Modell in 3. Normalform

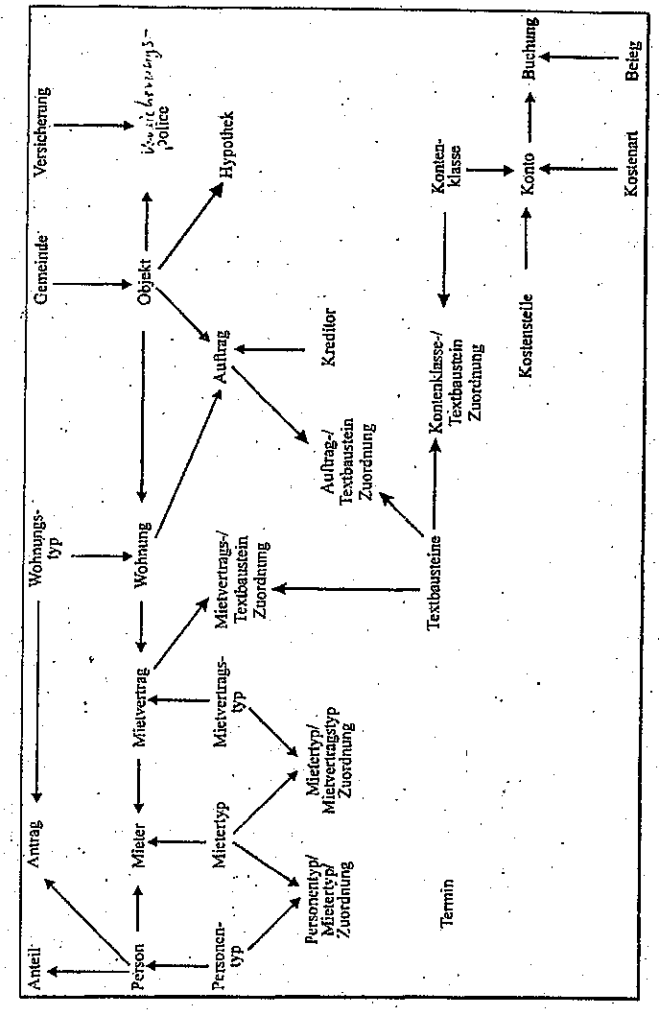


Abbildung 53: Das ERM in 3. Normalform

3.2

3.3

5.3.3. Datenstrukturdiagramme (Vereinsverwaltung)

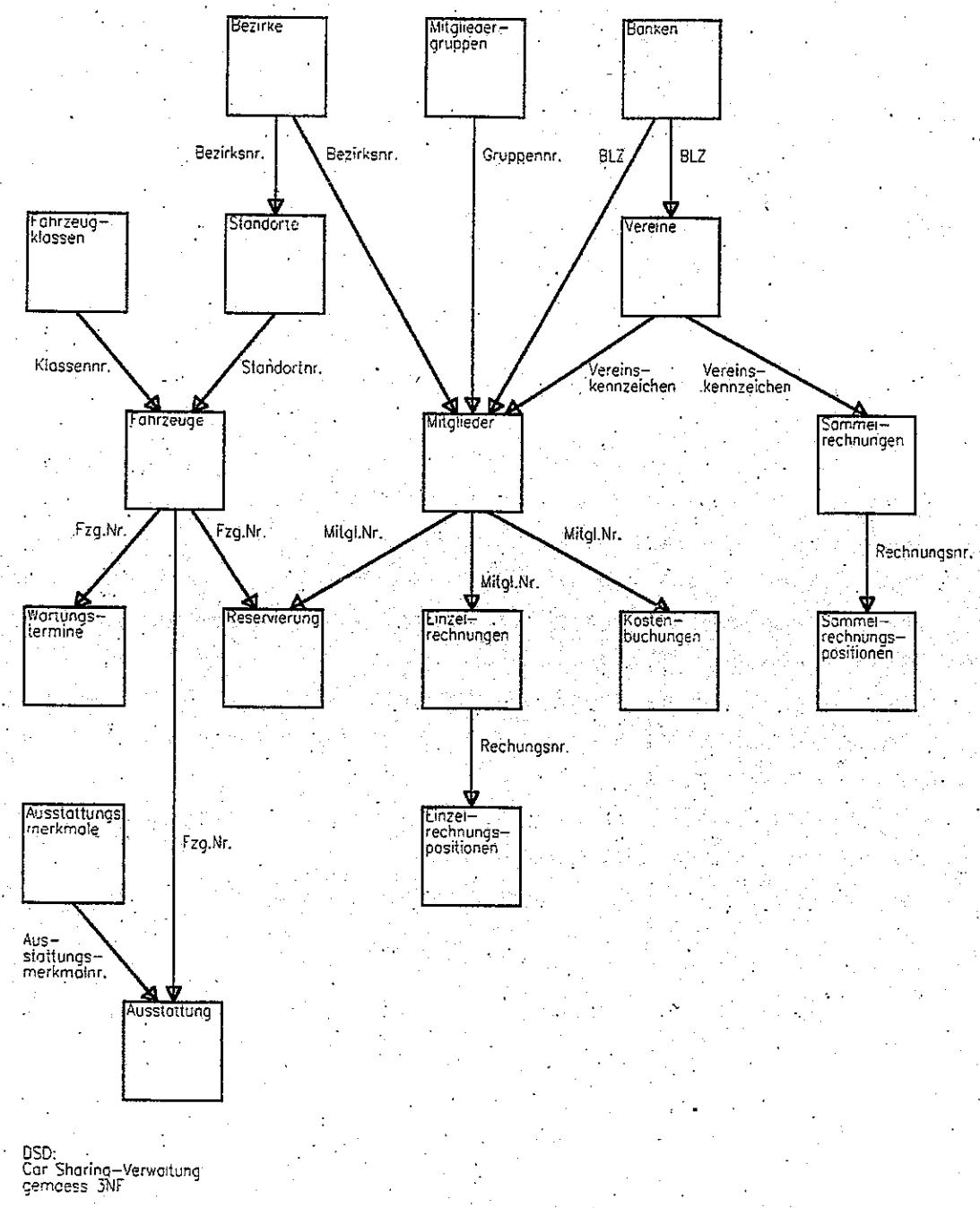


Abb. 23: Datenstrukturdiagramm gemäß 3NF.

4.1

3NF-basierte objektorientierte Modellbildung in der Wirtschaftsinformatik

Abkürzungen

3NF dritte Normalform
ER Entity-Relationship
OO objektorientiert

These

Ein relationales 3NF-Datenmodell muß die Grundlage eines statischen OO-Modells (Klassenmodells) der Wirtschaftsinformatik bilden.

Ableitung eines aufwärtskompatiblen OO-Modells aus einem 3NF-Modell:

1. Jeder 3NF-Entitätstyp enthält speichernswerte Daten und wird deshalb auf genau einen persistenten Objekttyp des OO-Modells abgebildet.
2. Konditionale Abhängigkeiten (1:c-Abhängigkeiten) und 1:1-Abhängigkeiten, die beide in einer 3NF unberücksichtigt bleiben, werden zur semantischen Verfeinerung in das OO-Modell aufgenommen.
3. Beziehungen (relationships) werden je nach Semantik interpretiert als:
   3.1 einfache Assoziationen/Instanzenverbindungen (Objektrelationen/-verbindungen ohne semantische Differenzierung) oder
   3.2 kompositionelle Strukturen (whole-part, Aggregation/Komposition) oder
   3.3 taxonomische Strukturen (is a, Generalisierung-Spezialisierung, Vererbung; nicht bei 1:n-Beziehungen).
4. Eine zusätzliche Erweiterung des OO-Modells um transiente Objekttypen (ohne speichernswerte Daten) ist möglich.
5. Die Objekttypen werden um Elementarfunktionen (Methoden, Services) ergänzt.

Begründung

Ich will zunächst die Grundprinzipien der relationalen und 3NF-Modellierung kurz skizzieren, um ihre mathematische und erkenntnistheoretische Bedeutung hervorzuheben. Im Anschluß daran werde ich meine These zusammenfassend begründen. OO-Modellierung kann - richtig eingesetzt - in der Wirtschaftsinformatik viele Vorteile bringen, die ich abschließend auflisten werde.

Übersicht

1. Die mathematischen Prinzipien der relationalen Datenmodellierung
2. Die mathematischen Prinzipien der 3NF-Datenmodellierung
3. Zusammenfassende Begründung 3NF-basierter OO-Modellierung
4. Vorteile 3NF-basierter OO-Modellierung in der Wirtschaftsinformatik

Der Text ist für Leser geschrieben, die die 3NF-Modellierung sehr gut kennen und die Grundlagen der OO-Modellierung beherrschen.

1. Die mathematischen Prinzipien der relationalen Datenmodellierung

Motivation der mathematischen Defintionen

Im folgenden will ich von bekannten Eigenschaften von Entität, Entitätstyp und Entitätsmenge ausgehend induktiv zu sinnvollen mathematischen Definitionen dieser Termini gelangen.

Das Grundprinzip relationaler Modellierung besteht darin, reale Objekte (was immer das sein mag; ich vernachlässige hier die erkenntnistheoretische Dimension) als Entitäten in Gestalt von Tupeln zu modellieren. (Um es wegen ungenauer Definitionen in der Literatur eigens zu betonen: Eine Entität ist kein reales Objekt (das gleiche gilt für ein OO-Objekt), sondern ein Modell eines realen Objektes.) Ein Tupel ist ein Element eines kartesischen Produktes, dessen Faktormengen Attributwertebereiche (kurz Attribute) darstellen. Die Komponenten eines Tupels sind die Attributwerte der zugehörigen Entität.

Ich betrachte als erstes die mathematische Definition von Entitätstyp. Daß und wie Tupel, Attribute und Relationen dort einzubeziehen sind, ist klar. Einen Entitätstyp aber lediglich als Menge gleichartiger Entitäten zu definieren, wäre mathematisch zu schwach. Es darf nämlich nicht jede Entität mit jeder anderen gleichartigen in einer Entitätsmenge vorkommen. Ein Entitätstyp ist also zu beschreiben durch eine Struktureigenschaft und eine Verträglichkeitseigenschaft: Er wird konstituiert von allen Mengen, die Entitäten umfassen, die gleiche Attributwertebereiche haben und die miteinander verträglich sind. Diese sollen Entitätsmengen heißen. Sie haben die Gestalt von Relationen auf dem kartesischen Produkt der Attribute, daher der Ausdruck "Relationenmodell".

Die obige verbale Beschreibung einer Verträglichkeitseigenschaft ist für eine mathematische Definition noch zu wenig aussagekräftig, schlecht formalisierbar und bedarf einer genaueren Betrachtung. Zunächst soll aber anhand zweier Beispiele erläutert werden, daß überhaupt eine Verträglichkeitseigenschaft benötigt wird:

1. Angenommen es soll beim Entitätstyp "Kunde" pro Kunde genau eine Adresse geben, dann darf in jeder Entitätsmenge jeder Kunde auch nur mit einer Adresse vertreten sein. Am Fall einer Anschriftenänderung sieht man, daß es zwar zu einem Kunden zwei Entitäten (mit alter und neuer Anschrift) geben kann, jedoch nur eine mit der jeweils aktuellen Adresse. Alte und neue Anschrift eines Kunden können nicht gleichzeitig in einer Entitätsmenge vorkommen, sonst wäre die Voraussetzung verletzt, daß es pro Kunde genau eine Anschrift geben soll; sie können aber sehr wohl in verschiedenen Entitätsmengen vorkommen.

2. Zum Entitätstyp "Kunde" mit den Attributen "Nummer" (Primärschlüssel) und "Name" gehören verschiedene Entitätsmengen, etwa

{(10000,Meier), (20000,Huber)}

und {(10000,Moser), (20000,Huber)},

nicht aber {(10000,Meier), (10000,Moser)},

da die beiden Tupel den gleichen Primärschlüsselwert haben und daher nicht gleichzeitig Elemente einer Entitätsmenge sein können.

Es ist wichtig zu sehen, daß nicht jede Menge von Entitäten eines Entitätstyps auch eine Entitätsmenge des Entitätstyps ist.

Ich fasse kurz zusammen:

Die Struktureigenschaft beschreibt, welche Attribute ein Entitätstyp hat.
Zwei Entitäten haben die gleiche (Entitäts)Struktur, wenn sie die gleichen Attribute aufweisen.
(Die Sprechweise "zwei Entitäten sind vom gleichen Typ" ist sehr unglücklich, da sie nur besagen soll, daß zwei Entitäten die gleiche Struktur haben, aber keine Aussage bezüglich der Verträglichkeit trifft, also mit einem Entitätstyp nur halb etwas zu tun hat.)

Die Verträglichkeitseigenschaft beschreibt, welche Entitäten gleichzeitig in einer Entitätsmenge des Entitätstyps auftreten können.
Zwei Entitäten >>gleichen Typs<< sind miteinander verträglich, wenn sie gleichzeitig Elemente einer Entitätsmenge sein können, bzw. miteinander unverträglich, wenn sie nicht gleichzeitig Elemente einer Entitätsmenge sein können.

Wie ist nun eine solche Verträglichkeitseigenschaft mathematisch zu fassen? Die Menge aller Entitäten eines Entitätstyps ist trotz ihrer Tupelstruktur sicher kein Vektorraum, da keine Verknüpfungsoperation angenommen werden kann. Also muß ich ein ganz einfaches mathematisches Konzept zur Formulierung solcher Verträglichkeitseigenschaften heranziehen, die Äquivalenzrelation. Welche der beiden Eigenschaften "verträglich" oder "unverträglich" ist nun für eine mathematische Definition von Entitätstyp und Entitätsmenge brauchbar?

Die Verträglichkeitseigenschaft ist nicht geeignet, eine Äquivalenzrelation zu fordern, da man leicht ein Beispiel findet, in dem die Transitivitätsbedingung nicht erfüllt ist:
Denn: (10000,Meier) ~ (20000,Huber) , (20000,Huber) ~ (10000,Moser)
==> (10000,Meier) ~ (10000,Moser) ist eine falsche Aussage (v. s.).

Die Negation der Verträglichkeitseigenschaft ist aber sehr gut geeignet, eine Äquivalenzrelation zu fordern.
Denn: Reflexivität und Symmetrie gelten trivialerweise.
Transitivität: (10000,Meier) ~ (10000,Moser) , (10000,Moser) ~ (10000,Bauer)
==> (10000,Meier) ~ (10000,Bauer) ist eine richtige Aussage,
da zwei Elemente mit gleichem Primärschlüsselwert nicht gleichzeitig Elemente einer Entitätsmenge sein können.

Postuliert man für einen bestimmten Entitätstyp die Existenz dieser Äquivalenzrelation, so zerfällt die Menge aller Entitäten des Entitätstyps in disjunkte Äquivalenzklassen, die jeweils miteinander unverträgliche Entitäten enthalten. Da jedes vollständige Repräsentantensystem ausschließlich verträgliche Entitäten enthält, läßt sich sinnvoll definieren, daß jede Entitätsmenge des Entitätstyps eine Teilmenge eines vollständigen Repräsentantensystems ist.

Nun kann man aber bei einer so allgemeinen Definition einer Äquivalenzrelation in einem konkreten Einzelfall gar keine Aussage über die Verträglichkeit machen. Es gilt also, eine naheliegende, weniger abstrakte Definition einer Äquivalenzrelation zu finden, die sich aus dem Konzept eines Kreuzproduktes natürlich ergibt. Diese findet sich in Gestalt einer Projektion auf eine bestimmte Teilmenge des Kreuzproduktes.

Ich postuliere also, daß zwei Tupel eines Entitätstyps genau dann äquivalent sein sollen, wenn ihre Bilder unter der Projektion gleich sind. Es ist sinnvoll, einen Schlüsselkandidaten (einen möglichen Primärschlüssel) des Entitätstyps so zu definieren, daß das Kreuzprodukt der Attribute des Schlüsselkandidaten gerade gleich dem Wertebereich der Projektion ist. Denn in diesem Fall sind zwei Entitäten eines Entitätstyps genau dann unverträglich, wenn ihre Projektionen, also ihre Schlüsselkandidatwerte, gleich sind; und ein Schlüsselkandidat ist ja gerade so definiert, daß er in allen Entitäten einer Entitätsmenge verschiedene Werte annimmt.

Mathematische Formulierung der Definition von Entitätstyp und Entitätsmenge

Ein Entitätstyp ist ein Tupel (A,~) mit

(1) $n \in \mathbf{N}$ (Grad des Entitätstyps) und

$$A := \bigtimes_{i=1}^{n} A(i)$$ (Menge aller möglichen Entitäten des Entitätstyps) mit

A(i) (i=1,...,n) Mengen, den sogenannten Attributwertebereichen, und

(2) es existiert (mindestens) eine Auswahl P(I) von Faktormengen von A mit

$$P(I) := \bigtimes_{\substack{i=1 \\ i \in I}}^{n} A(i) \quad \text{mit} \quad I \subset \{1,\ldots,n\} \text{ derart, daß}$$

$$x,y \in A \implies [\, x \sim y \iff pr(x) = pr(y) \,] \quad \text{mit}$$

pr: A --> P(I) die kanonische Projektion von A auf P(I).

P(I) und jedes weitere P(J) mit (2) heißt Schlüsselkandidat des Entitätstyps (A,~).
Sei M c A Teilmenge eines vollständigen Repräsentantensystems von ~.
Dann heißt M Entitätsmenge des Entitätstyps (A,~).
Die Elemente von A heißen Entitäten des Entitätstyps (A,~).
A beschreibt die Struktur des Entitätstyps und ~ die Unverträglichkeit seiner Entitäten.

Bemerkung 1 (Attribute)

Es ist mir wichtig festzustellen, daß im Rahmen relationaler Modellierung die Entitätstypen durch ihre Attribute definiert werden. Bei klassischer ER-Modellierung ist dies nicht notwendig so. Es werden - zumindest am Anfang des Modellierungsprozesses - ungenau spezifizierte Entitätstypen nur mit Namen etikettiert, die nicht selten mißverständlich sind und zu Fehlinterpretationen führen.
Erst die Festlegung von Attributen führt aber zu Klarheit und Eindeutigkeit.
Nicht Etiketten, sondern Attribute sind also konstitutiv für Entitätstypen.
Bei der OO-Modellierung müssen zwar Attribute verwendet werden, sie werden aber mitunter (wie bei ER auch) erst sehr spät und nicht sofort bei der Definition eines Objekttyps festgelegt.

Bemerkung 2 (Gleichheit und Verträglichkeit von Entitäten)

Durch die obige Definition von Entitätstypen wird in einem konkreten Anwendungsfall klar entscheidbar, ob zwei Entitäten gleich sind oder nicht (komponentenweise Gleichheit der Tupel) und ob eine Entität zu einer Entitätsmenge gehören kann oder nicht (Untersuchung der Projektion).
Diese beiden sehr nützlichen Eigenschaften werden in der Objektmodellierung ohne Not aufgegeben, wenn zugelassen wird, daß gleiche Tupel verschiedene reale Objekte repräsentieren, und wenn Objekte eines Typs nicht auf ihre Verträglichkeit hin untersucht werden.

Bemerkung 3 (Schlüsselkandidaten und Primärschlüssel)

Zu jedem Entitätstyp existiert mindestens ein Schlüsselkandidat; bei der Entscheidung für einen bestimmten spricht man von einem Primärschlüssel.

Primärschlüssel sind also keine künstlichen Instrumente, die nur zum Zweck der Anwendung relationaler DBMS eingeführt werden müssen. Sie ergeben sich vielmehr ganz natürlich aus der Darstellung von Entitäten als Attributwerttupel, die nur unter ganz bestimmten Bedingungen miteinander verträglich sind. Obwohl auch die OO-Modellierung auf dieser Tupeldarstellung basiert, werden Primärschlüssel dort gerne verworfen.

Bemerkung 4 (Entitätsmengen)

Sei M eine Entitätsmenge von (A,~) und P(I) ein Schlüsselkandidat von A. Dann ist

pr: M --> pr(M) c P(I) bijektiv.

Dies ist klar, denn M ist Teilmenge eines vollständigen Repräsentantensystems, enthält also pro Äquivalenzklasse höchstens ein Element, und Elemente verschiedener Äquivalenzklassen haben verschiedene Bilder unter pr.

Eine Menge soll Entitätsmenge heißen, wenn sie aus jeder Äquivalenzklasse unverträglicher Entitäten höchstens ein Element enthält.
Etwas weniger mathematisch formuliert: Eine Entitätsmenge ist eine (konkrete) Ausprägung eines (abstrakten) Entitätstyps. Es gibt in der Regel mehrere Entitätsmengen eines Entitätstyps, die etwa zu verschiedenen Zeitpunkten gültig sind.

Bemerkung 5 (Relationen)

Der Datenbank-Theoretiker mag eine Coddsche Spitzfindigkeit vermissen, die besagt: Bei einer Datenbank-Relation (Datenmodellierungs-Relation) komme es im Unterschied zu einer mathematischen Relation nicht auf die Reihenfolge der Komponentenmengen an, sie sei kommutativ. Ich halte diese Unterscheidung für überflüssig, da zwei Kreuzprodukte der gleichen Komponentenmengen jederzeit über eine Permutation der Indizes als isomorph betrachtet werden können:

$$\bigtimes_{i=1}^{n} A(i) \tilde{=} \bigtimes_{j=1}^{n} A(j)$$

genau dann, wenn eine n-stellige Permutation p existiert, derart daß

$$j = p(i) \quad \text{für alle} \quad i = 1,\ldots,n .$$

Damit erübrigt sich also eine Unterscheidung von zwei Relationstypen, und man kann ausschließlich die mathematische Relation verwenden.

2. Die mathematischen Prinzipien der 3NF-Datenmodellierung

Ein 3NF-Modell ist ein Relationenmodell mit verschärften mathematischen Anforderungen. Es ist von unkontrollierten Redundanzen befreit und durch Axiomatisierung mathematisch optimiert. Grundsätzlich kann jedes Relationenmodell in ein 3NF-Modell übergeführt werden, was sich mit einem mathematischen Beweis zeigen läßt, den ich jedoch hier nicht ausführen möchte.

Natürlich wird bei der Erstellung eines 3NF-Modells in der Praxis der Normalisierungskalkül nicht sklavisch vollzogen (analytische Vorgehensweise), sondern es werden zunächst intuitiv Entitätstypen mit ihren Attributen festgelegt, und anschließend wird geprüft, ob sie wohldefiniert sind und die drei Normalisierungsbedingungen (Freiheit von Wiederholgruppen, funktionalen Schlüsselteilabhängigkeiten und transitiven Abhängigkeiten) erfüllen, und das Datenmodell ggf. korrigiert (synthetische Vorgehensweise).

Durch die 3NF-Normalisierung eines relationalen Datenmodells werden die erfaßten Entitätstypen und Beziehungen so festgelegt, daß sie die folgenden beiden mathematischen Eigenschaften aufweisen:

Eigenschaft der Entitätstypen

1. Bei jedem beliebigen Entitätstyp (A,~) gibt es zu einem Primärschlüsselwert pro Attribut nur genau einen Wert (inkl. NULL); Wiederholgruppen sind ausgeschlossen, d. h. die Komponentenmengen von A sind paarweise verschieden.
2. Es bestehen (bis auf Primärschlüsselabhängigkeiten) keine funktionalen Abhängigkeiten von Attribut(grupp)en eines Entitätstyps untereinander.
Der Normalisierungskalkül ist in der Literatur so gut dargestellt, daß ich an dieser Stelle die mathematischen Details übergehen kann.

Eigenschaft der Beziehungen zwischen Entitätstypen

Verschiedene Entitätstypen stehen nur in 1:n-Beziehungen zueinander. Das heißt, daß zwei Entitätstypen nicht unabhängig voneinander verschiedene Ausprägungen (sprich Entitätsmengen) haben können, sondern daß je zwei Ausprägungen durch eine nicht-injektive Funktion verbunden werden können.
Mathematisch gesprochen bedeutet das, daß die Beziehungen zwischen Entitätsmengen (die grundsätzlich beliebig komplexe Formen annehmen können) auf nicht-injektive funktionale Abhängigkeiten reduziert werden.

Zwei Entitätsmengen M(1) und M(2) von zwei Entitätstypen, die in einer 1:n-Beziehung zueinander stehen, sind also genau dann miteinander verträglich, wenn gilt: Die Funktion

f:M(2) --> M(1),

die jeder Entität von M(2) die zugehörige Entität von M(1) zuordnet, ist wohldefiniert und nicht-injektiv.

Denn: Wohldefiniertheit: Sie ergibt sich aus der Fremdschlüsselreferenz.
Nicht-Injektivität: Es gibt mindestens zwei Entitäten von M(2), die auf die gleiche Entität von M(1) abgebildet werden, sonst würde keine 1:n-Beziehung vorliegen, die gerade so definiert wird.

Bewertung der beiden mathematischen Eigenschaften

Die beiden mathematischen Eigenschaften eines 3NF-Modells (und die zugrundeliegenden Eigenschaften eines Relationenmodells) sind nicht willkürlich gewählt, sondern spiegeln mathematische Struktureigenschaften der Realwelt wider, so daß ein solches Modell einen Weltausschnitt mathematisch besser verständlich macht (mathematische Realitätsadäquatheit). Und im Erklärungswert eines Modells liegt ja stets auch sein Erkenntniswert.

Der Erkenntniswert wird durch folgende Überlegung noch deutlicher:
Zu einem Weltausschnitt gibt es grundsätzlich nicht nur ein mathematisches Modell, sondern mehrere, die alle gleichzeitig richtig sind. Sie sind insoweit äquivalent, unterscheiden sich aber in ihrer Einfachheit und Kürze, Eleganz und Ästhetik, die ihrerseits eine ökonomische Bedeutung haben.
Man betrachte zum Vergleich physikalische Bewegungsgleichungen (auch hierbei handelt es sich um mathematische Modelle), die in Abhängigkeit von der Wahl des Koordinatenursprungs völlig verschiedene Gestalt haben können, obwohl sie alle richtig sind. Was zeichnet nun eine von ihnen besonders aus? Es ist ihre mathematische Einfachheit, die die Lösungsfindung erleichtert. Nicht ohne Grund legt man zur Beschreibung der Planetenbewegung im Sonnensystem den Koordinatenursprung in den Mittelpunkt (Schwerpunkt) der Sonne und nicht auf den Nordpol der Erde, obwohl dies nicht falsch wäre, aber sehr komplexe Bewegungsgleichungen zur Folge hätte. Äquivalente Bewegungsgleichungen können durch Koordinatentransformationen ineinander übergeführt werden.
So wie man mit einer speziellen Koordinatentransformation eine Bewegungsgleichung in ihre mathematisch einfachste Form überführen kann, so transformiert der Normalisierungskalkül (im Sinne einer Standardisierung) ein relationales Datenmodell in seine mathematisch einfachste Form, die 3NF. Die Schwierigkeiten beim Vergleich von OO-Modellen verschiedener Entwickler rührt m. E. daher, daß sie ihre Modelle nicht so weit wie möglich mathematisch vereinfachen. Aber gerade die Einfachheit eines Modells trägt wegen ihrer Ökonomie und leichten Verständlichkeit ganz erheblich zu seinem Erkenntniswert bei. So hat ein ästhetisches Merkmal einen ökonomischen Effekt.
Ein 3NF-Modell ist die mathematisch einfachste Form eines Relationenmodells und hat daher einen Erkenntniswert, der über den eines solchen hinausgeht.

Eine weitere interessante Beobachtung sei hier angeführt:
Die mathematischen Anforderungen an eine 3NF sind für einen Mathematiker so natürlich, so intuitiv einleuchtend, daß er sie unbewußt und automatisch bei einer relationalen Datenmodellierung verwendet, auch ohne die Terminologie und den Kalkül der Normalisierung explizit zu kennen. Ich selbst bin dafür ein lebendes Beispiel (und nicht das einzige): In meiner Berufstätigkeit als Programmierer waren alle meine COBOL-Dateibeschreibungen bereits in den 70er Jahren 3NF-Modelle, ohne daß ich damals eine Ahnung vom theoretischen Hintergrund hatte.

Ich betone zusammenfassend:
3NF-Modelle besitzen einen hohen Erkenntniswert, da sie
- mathematische Realitätsadäquatheit
- mathematische Einfachheit, Optimiertheit und Axiomatisiertheit

- mathematische Natürlichkeit

aufweisen.

Ich will noch auf zwei mögliche Einwände antworten:

Einwand 1

"Die 3NF-Normalisierung auf der Basis der Relationenmodellierung ist historisch aus der Anwendung relationaler DBMS entstanden und ihr Wert ist daher beschränkt auf Datenbankanwendungen."

Unabhängig von ihrer Entstehungsgeschichte hat die 3NF-Modellierung aufgrund ihrer mathematischen Anforderungen einen erheblichen Wert für die Datenmodellierung überhaupt. Ein 3NF-Modell kann so abgewandelt werden, daß es bei einer 3GL-, 4GL- oder sogar bei einer OO-Implementierung Anwendung findet. (Ebenso ist die OO-Analyse (OOA) über das OO-Design (OOD) aus der Anwendung von OO-Programmiersprachen entstanden und beansprucht für sich, durchaus begründet, einen erkenntnistheoretischen Wert.)
Ein 3NF-Modell hat also einerseits erkenntnistheoretischen Wert (mathematische Realitätsadäquatheit, Einfachheit und Natürlichkeit) und ist andererseits an verschiedene DV-Entwicklungsumgebungen anpaßbar (Computeradäquatheit).
Diese beiden Eigenschaften sollte ein Modell in der angewandten Informatik aufweisen.

Einwand 2

"Ein 3NF-Modell muß bei der Implementierung mitunter zum Zweck der Laufzeitoptimierung verändert werden."

Ein Modell in der angewandten Informatik steht immer im Spannungsfeld zwischen Realitätsadäquatheit und Computeradäquatheit. Die Tatsache, daß ein 3NF-Modell, das aufgrund seiner besonderen mathematischen Eigenschaften realitätsadäquat ist, auch noch leicht an die Erfordernisse einer Implementierung angepaßt werden kann, erhöht seinen Wert für die Informatik. Computeradäquate Modelle, die der Realität kaum angemessen sind, haben keinen Wert.

4.11

3. Zusammenfassende Begründung 3NF-basierter OO-Modellierung

Ein Modell eines Weltausschnitts in der Wirtschaftsinformatik beinhaltet naturgemäß ein komplexes Datenmodell, da viele Daten für eine spätere Weiterverwendung und Auswertung zu speichern sind. Das zugehörige Funktionsmodell ist in der Regel weit weniger aufwendig.

Die relationale 3NF-Datenmodellierung hat sich als Konzept für die Entwicklung von Informationssystemen in der Wirtschaftsinformatik bestens bewährt. Beim derzeitigen Kenntnisstand verfügen wir über kein besseres Datenmodellierungsprinzip. Die mathematischen Anforderungen an solche Modelle sind so streng, daß wenig Interpretationsspielraum bleibt und daß verschiedene Entwickler bei der Modellierung des gleichen Weltausschnitts zu zumindest ähnlichen, wenn nicht gleichen Ergebnissen gelangen. Mathematische Optimierung ist eindeutig möglich. Diese angenehme Situation findet sich bei OO-Modellen keineswegs, wenn verschiedene OO-Analysten den gleichen Weltausschnitt teilweise in nicht vergleichbarer und nicht nachvollziehbarer Weise modellieren. Semantische Optimierung ist nicht eindeutig möglich. Ich sehe keinen Grund, den bewährten Qualitätsanspruch an 3NF-Modelle nur wegen der Verfolgung eines neuen, "modernen" Modellierungskonzepts zu erweichen oder gar aufzugeben, zumal ein aufwärtskompatibles Einbeziehen bisheriger Modelle ohne weiteres möglich ist.

Ich glaube, die OO-Gemeinde muß erkennen, daß an Objekttypen Ansprüche mathematisch-formaler Wohldefiniertheit zu richten sind. Will man einen Weltausschnitt so modellieren, daß er in einem Computer abbildbar wird, muß man es mit mathematisch-formalen Spezifikationen tun, da der Computer eben eine mathematisch-formale Maschine ist.

Von Seiten der Erkenntnistheorie ist also gegen den naiven Realismus in OO-Kreisen einzuwenden, daß OO-Objekttypen genauso wenig wie Entitätstypen vom Himmel fallen und auch nicht wie platonische Ideen präexistent sind. Was intuitiv und aufgrund seiner natürlichsprachlichen Bezeichnung als möglicher Objekttyp erscheint, muß erst auf seine mathematische Wohldefiniertheit geprüft werden.

Nun verfügt man aber mit einem relationalen 3NF-Modell über ein mathematisch-formales Modell, das bereits hohen Anforderungen genügt. Also liegt nichts näher, als eben dieses als Grundlage für ein statisches OO-Modell zu wählen.

Terminologisch ist dies auch ganz einfach. Ich stelle die beiden Begriffswelten gegenüber:

Entität	Objekt (oder schlechter Objektinstanz)
Entitätsmenge	Objektmenge
Entitätstyp	Objekttyp (oder schlechter Objektklasse)
Beziehung	Assoziation
Kardinalität	Multiplizität

Diese Parallelisierung findet sich auch in Barkow, Georg et al.: Begriffliche Grundlagen für die frühen Phasen der Software-Entwicklung; Information Management 4(1989)4, 54-60.

Man definiere also die Objekttypen eines OO-Modells entsprechend den Entitätstypen des zugrundeliegenden 3NF-Modells. Damit besitzen sie nicht nur irgendwelche obskuren Namen, sondern von Anfang an die sie konstituierenden Attribute. Weiter verfahre man im Sinne meiner These, ohne sich sklavisch an die dort gewählte Reihenfolge zu halten.

Festlegung der OO-Methoden/Funktionen/Services

Eine einfache Vorgehensweise zur Ermittlung von OO-Methoden ist der Entwurf eines Modells nach SA (einer strukturierten Methode), dessen Speicher 3NF-Entitätstypen sind. Verarbeitungsfunktionen eines Informationsflußdiagramms auf niedrigster Abstraktionsebene bzw. terminale Knoten eines Funktionsbaums sind atomisierte Funktionen, die als OO-Methoden bestens geeignet sind. Sie werden bei der OO-Modellierung einfach an die entsprechenden Objekttypen "gehängt". Zudem ist ein in mehreren Abstraktionsebenen entworfenes SA-Modell ein hervorragendes Werkzeug, um sich einen Überblick über einen Weltausschnitt zu verschaffen, an dem man m. E. auch bei OO-Modellierung nicht vorbeikommt.

Abgesehen von OO-Erwägungen stellt sich bei SA ständig das Problem der Definition der Speicher. Entweder man beläßt es bei einem unstrukturierten Datenpool, oder man arbeitet von Anfang an mit 3NF-Entitätstypen. Letzteres ist, wie gesagt, auch in bezug auf ein darauf aufbauendes OO-Modell vorzuziehen. Die unangenehmste Vorgehensweise ist sicher die Verwendung beliebiger Speicher, die auf der untersten Abstraktionsebene eines SA-Modells zu 3NF-Entitätstypen verfeinert und zusammengefaßt werden müssen. Diese Aufgabe kann zur Umstellung größerer Teile eines SA-Modells führen, was auf jeden Fall vermieden werden sollte.

4. Vorteile 3NF-basierter OO-Modellierung für die Wirtschaftsinformatik

1. Ein OO-Modell baut nach meinen Überlegungen auf einem Datenmodell in Form eines 3NF-Modells auf, das eine mathematisch wohldefinierte, mathematisch realitätsadäquate, computeradäquate, transparente, nachvollziehbare und wenig subjektive Basis bildet.

Datenmodelle sind als Grundlage von umfassenden Realitätsmodellen den Funktionsmodellen vorzuziehen, da bei letzteren grundsätzlich wesentlich mehr subjektive Interpretation einfließt als bei ersteren. Funktionsmodelle sind in diesem Sinne weniger stabil als Datenmodelle. Dies liegt daran, daß für das menschliche Gehirn verschiedene Kontinua unterschiedlich gut diskretisierbar sind: zeitliche Abläufe (als Grundlage von Funktionsmodellen) sind stets weniger leicht strukturierbar als räumliche Bilder (als Grundlage von Datenmodellen). Der Entitäts- und Objektbegriff entsteht ja als Abstraktion von räumlich wahrnehmbaren Gegenständen. Ein Raumkontinuum ist optisch wahrnehmbar und so für das menschliche Gehirn, das ja eine Erweiterung des optischen Wahrnehmungsapparates ist, leicht strukturierbar. Ein räumliches Bild kann nämlich als zeitlicher Schnappschuß festgehalten (konstant gehalten) und dann in Ruhe analysiert werden. Die Details hierzu sind Gegenstand der evolutionären Erkenntnistheorie.

2. In einem OO-Modell ist die Erweiterung eines 3NF-Modells um taxonomische und kompositionelle Strukturen möglich, was häufig zu einem besseren Verständnis der Modellsemantik beiträgt.

3. Ein OO-Modell zwingt zur Erfassung der beteiligten Funktionen, die wegen ihrer vermuteten Einfachheit in der Wirtschaftsinformatik häufig vernachlässigt werden. (Die Erstellung eines 3NF-Modells reicht ja zur Modellierung eines Wirklichkeitsausschnitts nicht aus; eine gleichzeitige Untersuchung der Funktionsstrukturen ist erforderlich.) Weiterhin beschränkt sich ein OO-Modell auf die Beschreibung von Elementarfunktionen, was die unter 1 geschilderten Schwierigkeiten bei der Funktionsmodellierung (die sich insbesondere bei einer hierarchischen Funktionsstrukturierung zeigen) erheblich reduziert.

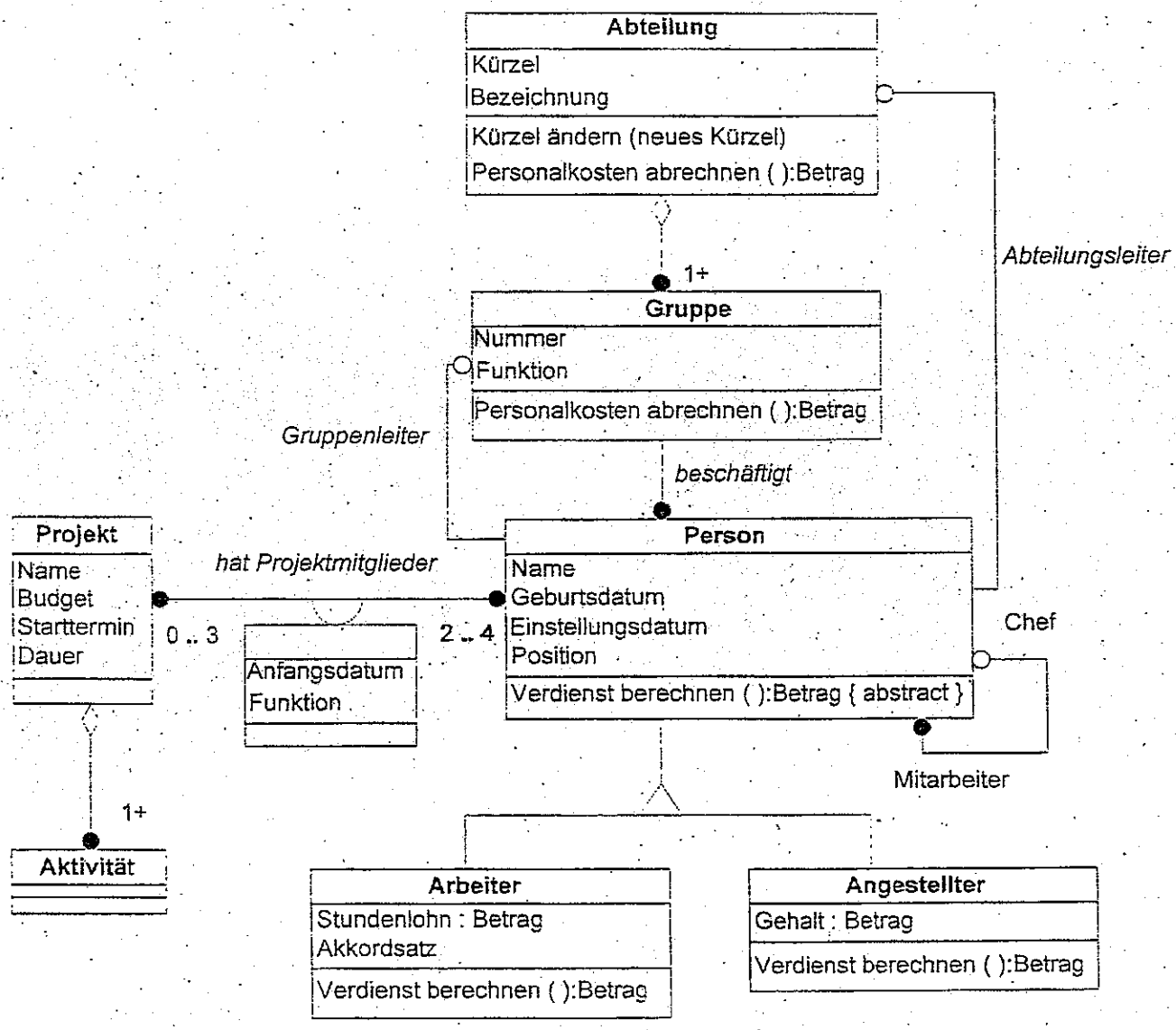
4. Mit der Verwendung von OO-Modellen ist die Durchgängigkeit der Entwicklungswerkzeuge von der Analyse über das Design zur Implementierung zu erwarten, wenn OODBMS eines Tages kommerziell verfügbar sein werden.

4.14

Das Objekt-Paradigma in der Wirtschaftsinformatik

Seminararbeit im Fach DV-Anwendungen in der Wirtschaft
im Sommersemester '96

Fachbereich Allgemeinwissenschaften und Informatik



Bearbeitet von: Udo Pelikan, Robert Scharold, Peter Söllner
Betreut von: Prof. Dr. A. Holl

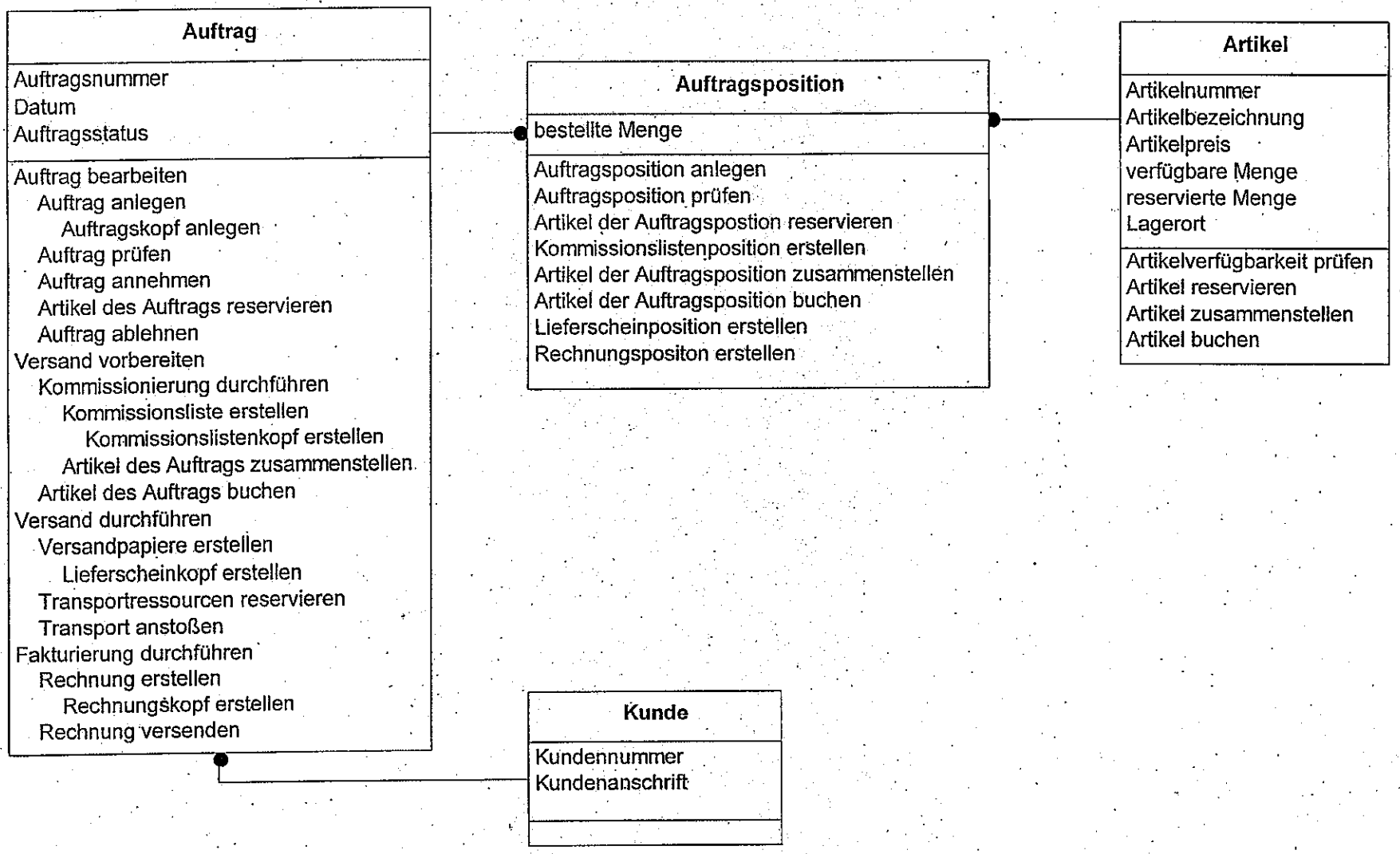


Abbildung 39: Abgeleitetes objektorientiertes Modell

4.16

Die Klasse Lieferung erstellt im Rahmen der Kommissinierung eine Kommissionsliste und veranlaßt die Kommissionierung des Auftrags. Für die Versendung werden die Versandpapiere erstellt, die Transportressourcen reserviert und der Transport angestoßen.

6.3.2.2 Dynamisches Objektmodell

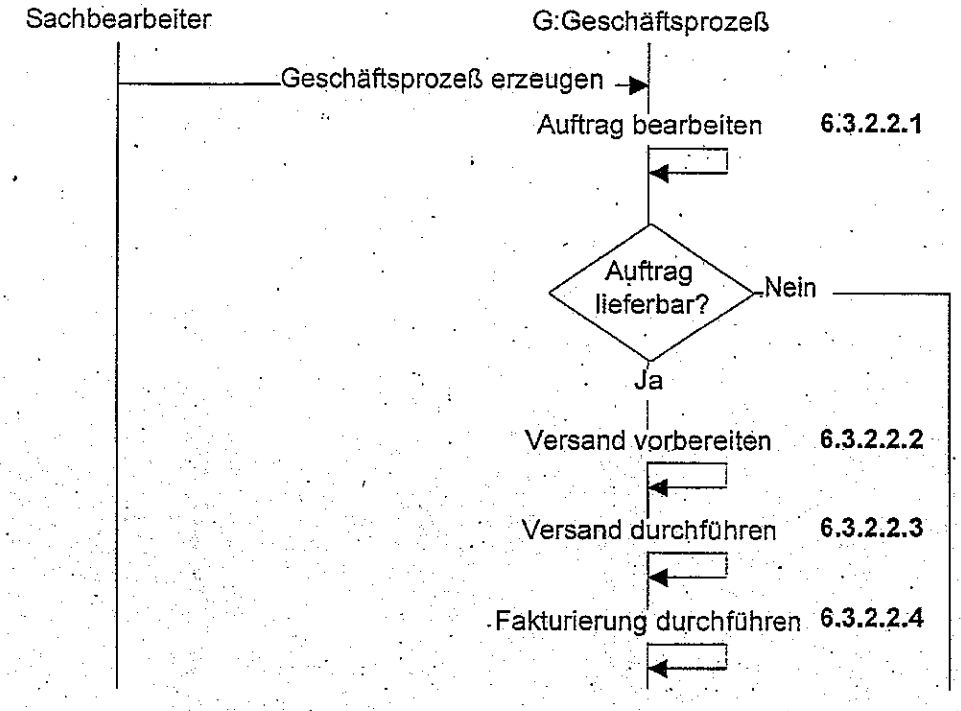


Abbildung 41: Szenario Auftragsabwicklung

Das Szenario *Auftragsabwicklung* zeigt die oberste Ebene des Geschäftsprozesses und wird an anderer Stelle noch weiter verfeinert.

Der Geschäftsprozeß wird durch einen Sachbearbeiter für einen bestimmten Kunden erzeugt. Hierauf wird die dem Geschäftsprozeß eigene Methode *Auftrag bearbeiten* aufgerufen. Hat die darin ausgeführte Artikelverfügbarkeitsprüfung des Auftrags ergeben, daß dieser lieferbar ist, wird der Auftragsstatus auf *angenommen* gesetzt, anderfalls erhält er den Wert *abgelehnt*. Für angenommene Aufträge werden die Methoden *Versand vorbereiten*, *Versand durchführen* und *Fakturierung durchführen* ausgeführt, die ebenfalls zur Funktionalität des Objekts Geschäftsprozeß gehören. Jeweils am Ende einer Methode wird der Auftragsstatus fortgeschrieben. Er wird also auf *versandfertig*, *versendet* bzw. *fakturiert* gesetzt. In den entsprechenden Szenarien wird dies nicht explizit dargestellt.

6.3.2.2.1 Auftrag bearbeiten

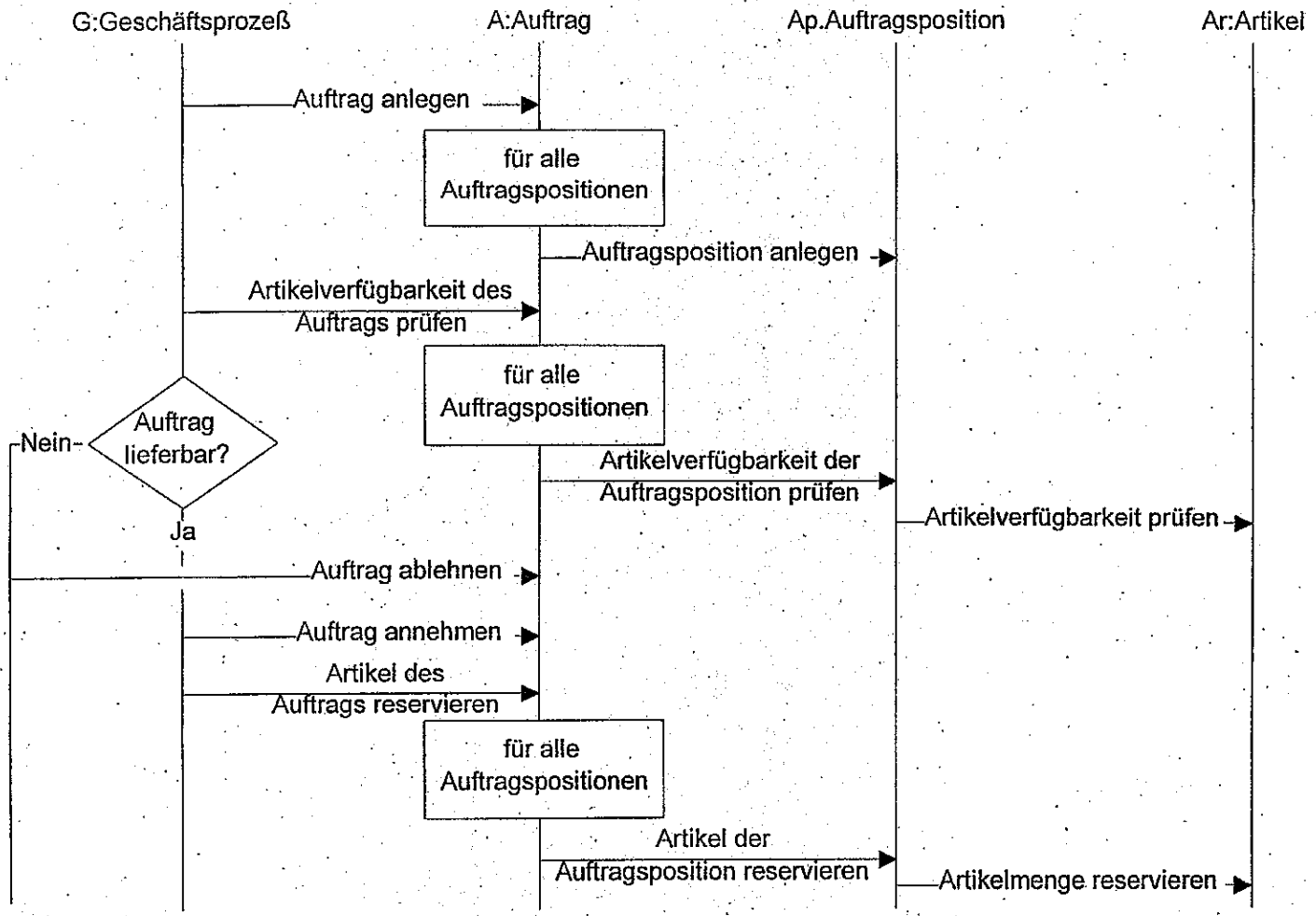


Abbildung 42: Szenario Auftrag bearbeiten

4.1 Grundform der SELECT-Anweisung

```
SELECT  [ALL|DISTINCT] *|{ausdr__1} ....
        FROM tab__liste
       [WHERE bedingung__1]
       [GROUP BY ausdr__3,...]
       [HAVING bedingung__2]
       [ORDER BY ausdr__5 [ASC|DESC],...];
```

4.2 WHERE-Klausel

Beispiel 4.5

Finden Sie alle Projekte, deren Finanzmittel mehr als 60000$ betragen. Der augenblickliche Kurs soll bei 0,55 Dollar für 1 DM liegen.

```
SELECT pr_name
   FROM projekt
   WHERE mittel * 0.55 > 60000;
```

4.8.1 Der Operator UNION

```
select__1 UNION [ALL] select__2
```

Beispiel 4.44

Finden Sie alle Wohnorte der Mitarbeiter und alle Standorte der Abteilungen.

```
SELECT wohnort
   FROM mitarbeiter
UNION
SELECT stadt
   FROM abteilung;
```

Damit zwei SELECT-Anweisungen mit dem UNION-Operator verbunden sein können, müssen folgende Voraussetzungen erfüllt sein:

a) die Anzahl der Spalten in den beiden Projektionen muß gleich sein;
b) die entsprechenden Spalten müssen denselben Typ haben.

Falls die Ausgabe sortiert sein soll, darf die ORDER BY-Klausel nur in der letzten SELECT-Anweisung angegeben werden.

Beispiel 4.45

Finden Sie die Personalnummer der Mitarbeiter, die entweder der Abteilung a1 zugehören oder vor dem 1.1.1989 in das Projekt eingetreten sind. Die Personalnummern sollen sortiert ausgegeben werden.

```
SELECT m_nr
   FROM mitarbeiter
   WHERE abt_nr = 'a1'
UNION
SELECT m_nr
   FROM arbeiten
   WHERE einst_dat < '01.01.1988'
   ORDER BY 1;
```

Beispiel 4.46

Finden Sie alle Mitarbeiter, die entweder der Abteilung a1 oder a2 oder beiden angehören.

```
SELECT m_nr, m_name, m_vorname
   FROM mitarbeiter
   WHERE abt_nr = 'a1'
UNION
SELECT m_nr, m_name, m_vorname
   FROM mitarbeiter
   WHERE abt_nr = 'a2';
```

Dieselbe Aufgabe kann mit dem OR-Operator einfacher gelöst werden:
(Reihen aus einer Tabelle)

Beispiel 4.47

```
SELECT m_nr, m_name, m_vorname
   FROM mitarbeiter
   WHERE abt_nr = 'a1'
   OR    abt_nr = 'a2';
```

4.2.2 Die Operatoren IN und BETWEEN

5.1

Beispiel 4.11

Finden Sie alle Mitarbeiter, deren Personalnummer entweder 29346, 28559 o 25348 ist.

```
SELECT *
   FROM mitarbeiter
   WHERE m_nr IN (29346, 28559, 25348);
```

Der IN-Operator kann durch eine Reihe vom OR-Operatoren ersetzt werden. B spiel 4.12 ist mit Beispiel 4.11 identisch.

Beispiel 4.12

```
SELECT *
   FROM mitarbeiter
   WHERE m_nr = 29346
   OR    m_nr = 28559
   OR    m_nr = 25348;
```

Beispiel 4.14

Nennen Sie Namen und Mittel aller Projekte, deren finanzielle Mittel zwisch 95000DM und 120000DM liegen.

```
SELECT pr_name, mittel
   FROM projekt
   WHERE mittel BETWEEN 95000 AND 120000;
```

Der BETWEEN-Operator kann auch durch Vergleichsoperatoren ersetzt we den. Folgendes Beispiel entspricht dem Beispiel 4.14:

Beispiel 4.15

```
SELECT pr_name, mittel
   FROM projekt
   WHERE mittel <= 120000
   AND   mittel >=  95000;
```

Die Anweisung in 4.14 ist transparenter und deswegen leichter lesbar. Zude bringt die Verwendung des BETWEEN-Operators Zeitvorteile bei der Abarbeitu der SELECT-Anweisung.

4.3.2 Unterabfragen und IN-Operator

Beispiel 4.24

Nennen Sie die Daten aller Mitarbeiter, die in München arbeiten.

```
SELECT *
   FROM mitarbeiter
   WHERE abt_nr IN
   (SELECT abt_nr
       FROM abteilung
       WHERE stadt = 'Muenchen');
```

4.2.4 Der Operator LIKE

Das Zeichen „%" kennzeichnet eine beliebige Zeichenfolge von n Zeichen, w n eine nichtnegative ganze Zahl ist, also auch 0 sein kann. Das Zeichen „_" ke zeichnet ein beliebiges alphanumerisches Zeichen. Jedes andere alphanumeris Zeichen kennzeichnet sich selbst.

Beispiel 4.18

Finden Sie Namen und Personalnummer aller Mitarbeiter, deren Name mit Buchstaben „K" beginnt.

```
SELECT m_name, m_nr
   FROM mitarbeiter
   WHERE m_name LIKE 'K%';
```

Beispiel 4.20

Nennen Sie Namen, Vornamen und Personalnummer aller Mitarbeiter, deren N name als zweiten Buchstaben „a" hat.

```
SELECT  m_nr, m_name, m_vorname
   FROM mitarbeiter
   WHERE m_vorname LIKE '_a%';
```

4.4 GROUP BY-Klausel

5,2

Beispiel 4.29

Welche Aufgaben üben die Mitarbeiter der Firma aus?

```
SELECT aufgabe
   FROM arbeiten
   GROUP BY aufgabe;
```

Beispiel 4.30

Gruppieren Sie die Mitarbeiter nach Projektnummer und Aufgabe.

```
SELECT pr_nr, aufgabe
   FROM arbeiten
   GROUP BY pr_nr, aufgabe;
```

4.6 HAVING-Klausel

Die HAVING-Klausel hat dieselbe Funktion für die GROUP BY-Klausel wie die WHERE-Klausel für die SELECT-Anweisung. Mit anderen Worten: Die HAVING-Klausel definiert die Bedingung nach der die Reihengrupen ausgewählt werden.

Beispiel 4.40

Nennen Sie alle Projekte, mit denen weniger als vier Mitarbeiter befaßt sind.

```
SELECT pr_nr
   FROM arbeiten
   GROUP BY pr_nr
   HAVING COUNT(*) < 4;
```

Die HAVING-Klausel kann auch ohne die GROUP BY-Klausel benutzt werden, obwohl dies in der Praxis selten vorkommt. In diesem Fall wird die gesamte Tabelle als eine einzige Gruppe betrachtet.

4.7 ORDER BY-Klausel

ORDER BY {ausdruck|ganzzahl [ASC|DESC]},...

Beispiel 4.41

Geben Sie Personalnummer, Namen und Vornamen aller Mitarbeiter, sortiert nach Personalnummern, an.

```
SELECT m_nr, m_name, m_vorname
   FROM mitarbeiter
   ORDER BY m_nr;
```

Jede Spalte in der ORDER BY-Klausel kann durch eine ganze Zahl ersetzt werden, die die Position dieser Spalte in der Projektion definiert.
Die Verwendung der ganzen Zahlen in der ORDER BY-Klausel ist die einzige Alternative, falls der Sortierbegriff durch eine Aggregatfunktion definiert ist, wie das folgende Beispiel verdeutlicht.

Beispiel 4.43

Finden Sie die Anzahl aller Mitarbeiter in jedem Projekt und sortieren Sie sie anschließend in absteigender Reihenfolge.

```
SELECT pr_nr, COUNT(*)
   FROM arbeiten
   GROUP BY pr_nr
   ORDER BY 2 DESC;
```

5.3

4.5 Aggregatfunktionen

- MIN;
- MAX;
- SUM;
- AVG und
- COUNT.

Die Aggregatfunktionen können in einer SELECT-Anweisung mit oder ohne GROUP BY-Klausel erscheinen. Falls die SELECT-Anweisung die GROUP BY-Klausel nicht enthält, dürfen in der Projektion nur die Spaltennamen angegeben werden, die als Parameter der Aggregatfunktion erscheinen. Deswegen ist das folgende Beispiel falsch:

keine Ergänzung eines Skalaren Subselect durch Spaltenanzahl

Beispiel 4.31

```
SELECT m_name, MIN(m_nr)
   FROM mitarbeiter;
```

Alle Spaltennamen, die nicht Parameter der Aggregatfunktion sind, dürfen in der SELECT-Anweisung erscheinen, falls sie zur Gruppierung verwendet werden.

4.5.1 Die Funktionen MIN und MAX

Beispiel 4.33 cf. 4.28, 5.26

Nennen Sie Personalnummer und Namen des Mitarbeiters mit der kleinsten Personalnummer.

```
SELECT m_nr, m_name
   FROM mitarbeiter
   WHERE m_nr =
   (SELECT MIN(m_nr)
       FROM mitarbeiter);
```

4.5.2 Die Funktion SUM

Beispiel 4.35

Berechnen Sie die Summe der finanziellen Mittel aller Projekte.

```
SELECT SUM(mittel)
   FROM projekt;
```

4.5.3 Die Funktion AVG

Beispiel 4.36

Berechnen Sie das arithmetische Mittel der Geldbeträge, die höher als 100000 DM sind.

```
SELECT AVG(mittel)
   FROM projekt
   WHERE mittel > 100000;
```

4.5.4 Die Funktion COUNT

Die Aggregatfunktion COUNT hat zwei verschiedene Formen. Die erste F
sieht wie folgt aus:

COUNT (DISTINCT sp__name)

Sie berechnet die Anzahl der Werte der Spalte sp__name, wobei alle mehrf
vorhandenen Werte nicht berücksichtigt werden. DISTINCT muß in diesem
angegeben werden.

Beispiel 4.37

Finden Sie heraus, wieviele verschiedene Aufgaben in jedem Projekt ausge
werden.

```
SELECT pr_nr, COUNT(DISTINCT aufgabe)
   FROM arbeiten
   GROUP BY pr_nr;
```

Die Funktion COUNT DISTINCT entfernt alle NULL-Werte aus der betreffen
Spalte.

Die zweite Form der Funktion COUNT sieht folgendermaßen aus:

COUNT (*)

Sie berechnet die Anzahl der Reihen.

Beispiel 4.38

Finden Sie heraus, wieviele Mitarbeiter in jedem Projekt arbeiten.

```
SELECT pr_nr, COUNT(*)
   FROM arbeiten
   GROUP BY pr_nr;
```

Im Unterschied zu der ersten Form der COUNT-Funktion, berücksicht
COUNT(*) auch NULL-Werte.

4.3.3 Die Operatoren ANY und ALL

ausdruck vergl_op [ANY | ALL] (unterabfrage)

Beispiel 4.27

Nennen Sie die Namen und Vornamen aller Mitarbeiter, die im Projekt p1 arbeiten.

```
SELECT m_name, m_vorname
   FROM mitarbeiter
   WHERE m_nr = ANY
   (SELECT m_nr
       FROM arbeiten
       WHERE pr_nr = 'p1');
```

oder Join:

```
SELECT m_name, m_vorname
   FROM mitarbeiter, arbeiten
   WHERE mitarbeiter.m_nr = arbeiten.m_nr
   AND pr_nr = 'p1';
```

Beispiel 4.28 cf. 4.33

Nennen Sie die Aufgabe des Mitarbeiters, der die kleinste Personalnummer hat.

```
SELECT aufgabe
   FROM arbeiten
   WHERE m_nr <= ALL
   (SELECT m_nr
      FROM mitarbeiter);
```

4.3.4 Der Operator EXISTS

Beispiel 5.25

Nennen Sie die Namen und Vornamen aller Mitarbeiter, die im Projekt p1 arbeiten.

```
SELECT m_name, m_vorname
   FROM mitarbeiter
   WHERE EXISTS
   (SELECT *
       FROM arbeiten
       WHERE pr_nr = 'p1'
       AND mitarbeiter.m_nr = arbeiten.m_nr);
```

Beispiel 5.23

Finden Sie die Städte, die sowohl die Wohnorte der Mitarbeiter als auch die Standorte der Abteilungen sind.

```
SELECT DISTINCT wohnort
   FROM mitarbeiter
   WHERE EXISTS
   (SELECT stadt
       FROM abteilung
       WHERE stadt = wohnort);
```

oder Join:

```
SELECT DISTINCT wohnort FROM mitarbeiter, abteilung
   WHERE stadt = wohnort
```

Beispiel 5.26 cf. 4.33

Nennen Sie die Aufgabe des Mitarbeiters, der die kleinste Personalnummer hat.

```
SELECT DISTINCT aufgabe
   FROM arbeiten
   WHERE NOT EXISTS
   (SELECT *
       FROM mitarbeiter
       WHERE NOT arbeiten.m_nr >= mitarbeiter.m_nr);
```

Beispiel 5.24

Finden sie die Wohnorte der Mitarbeiter, die nicht an den Firmenstandorten wohnen.

```
SELECT DISTINCT wohnort
   FROM mitarbeiter
   WHERE NOT EXISTS
   (SELECT stadt
       FROM abteilung
       WHERE wohnort = stadt);
```

4.8.2 Der Operator INTERSECT

4.8.3 Der Operator MINUS

Beispiel 4.48 (nur ORACLE)

Finden Sie die Städte, die sowohl die Wohnorte der Mitarbeiter als auch die Standorte der Abteilungen sind.

```
SELECT wohnort
   FROM mitarbeiter
INTERSECT
SELECT stadt
   FROM abteilung;
```

Der Operator INTERSECT kann durch den Operator EXISTS ersetzt werden. (siehe Beispiel 5.23).

$A \cap B = \{a \in A : \exists b \in B : a = b\}$

Beispiel 4.49 (nur ORACLE)

Finden Sie die Wohnorte der Mitarbeiter, die nicht an den Firmenstandorten wohnen.

```
SELECT wohnort
   FROM mitarbeiter
MINUS
SELECT stadt
   FROM abteilung;
```

Der Operator MINUS kann durch den Operator NOT EXISTS ersetzt werden (siehe Beispiel 5.24).

$A \setminus B = \{a \in A : \nexists b \in B : a = b\}$

- Equijoin,
- Kartesisches Produkt,
- natürlicher Join,
- Thetajoin
- Outer Join.

5.1.1 Equijoin

Beispiel 5.1

Finden Sie für jeden Mitarbeiter, zusätzlich zu seiner Personalnummer, Namen und Vornamen, auch die Abteilungsnummer und den Standort der Abteilung. Die doppelten Spalten beider Tabellen sollen ausgegeben werden.

```
SELECT mitarbeiter.*, abteilung.*
   FROM mitarbeiter, abteilung
   WHERE mitarbeiter.abt_nr = abteilung.abt_nr;
```

5.1.3 Natürlicher Join

Der natürliche Join entsteht aus einem Equijoin, wenn die doppelte Spalte aus der Projektion entfernt wird.

Beispiel 5.5

```
SELECT mitarbeiter.*, abt_name, stadt
   FROM mitarbeiter, abteilung
   WHERE mitarbeiter.abt_nr = abteilung.abt_nr;
```

5.1.4 Thetajoin

Der Thetajoin kennzeichnet jene SELECT-Anweisung, bei der die Joinspalten in der WHERE-Klausel mit einem der Vergleichsoperatoren verglichen werden. Die allgemeine Form eines Thetajoins sieht folgendermaßen aus:

```
SELECT tabelle_1.spalten, tabelle_2.spalten
FROM tabelle_1, tabelle_2
WHERE join_spalte_1 ϕ join_spalte_2;
```

wobei „ϕ" einen beliebigen Vergleichsoperator darstellt.

Beispiel 5.7

```
SELECT miterbeiter.*, abteilung.*
   FROM miterbeiter, abteilung
   WHERE wohnort > stadt;
```

5.1.6 Eine Tabelle mit sich selbst verknüpfen

Wird eine Tabelle mit sich selbst verknüpft, erscheint ihr Name doppelt in der FROM-Klausel einer SELECT-Anweisung. Damit der Tabellenname in beiden Fällen unterschieden werden kann, müssen Aliasnamen benutzt werden. Gleichzeitig müssen alle Spalten dieser Tabelle in der SELECT-Anweisung gekennzeichnet sein und zwar mit dem entsprechenden Aliasnamen als Präfix.

Beispiel 5.11

Finden Sie alle Abteilungen, die sich an demselben Standort befinden.

```
SELECT a.abt_nr, a.abt_name, a.stadt
   FROM abteilung a, abteilung b
   WHERE a.stadt = b.stadt
   AND a.abt_nr <> b.abt_nr;
```

Beispiel 5.12

Finden Sie Personalnummer, Namen und Wohnort der Mitarbeiter, die im gleichen Ort wohnen und zu derselben Abteilung gehören.

```
SELECT a.m_nr, a.m_name, a.wohnort
   FROM mitarbeiter a, mitarbeiter b
   WHERE a.wohnort = b.wohnort
   AND a.abt_nr = b.abt_nr
   AND a.m_nr <> b.m_nr;
```

5.5

5.1.7 Outer Join

Beispiel 5.14 (nur ORACLE)

Finden Sie alle Kombinationen der Mitarbeiter- und Abteilungsdaten für die C die entweder nur Wohnorte der Mitarbeiter oder gleichzeitig Wohnorte der M beiter und Standorte der Abteilungen sind.

```
SELECT miterbeiter.*, abteilung.abt_nr
   FROM mitarbeiter, abteilung
   WHERE wohnort = stadt(+);
```

Das Ergebnis ist:

m_nr	m_name	m_vorname	abt_nr	wohnort	abt_nr
29346	Probst	Andreas	a2	Augsburg	
9031	Meier	Rainer	a2	Augsburg	
10102	Huber	Petra	a3	Landshut	
25348	Keller	Hans	a3	Muenchen	a1
25348	Keller	Hans	a3	Muenchen	a2
2581	Kaufmann	Brigitte	a2	Muenchen	a1
2581	Kaufmann	Brigitte	a2	Muenchen	a2
18316	Mueller	Gabriele	a1	Rosenheim	
28559	Mozer	Sibille	a1	Ulm	

Beispiel 5.15 (nur INFORMIX)

```
SELECT mitarbeiter.*, abteilung.abt_nr
   FROM mitarbeiter, OUTER abteilung
   WHERE wohnort = stadt;
```

Im Unterschied zu Equi-, Theta- und natürlichem Join befinden sich die Tab im Outer Join auf verschiedenen Ebenen. Die Tabelle mitarbeiter befindet auf der oberen und die Tabelle abteilung auf der unteren Ebene. Die Ebene der sich eine Tabelle im Outer Join befindet, ist sehr wichtig, weil dadurch Ergebnis maßgeblich beeinflußt wird. Die Anzahl der Ebenen für den Outer ist nicht auf zwei beschränkt, sondern theoretisch unbegrenzt. Auf einer E können mehrere Tabellen angegeben werden.

Outer Join kann mit Hilfe des UNION-Operators und der Unterabfrage EXISTS-Operator ersetzt werden. Dies ist besonders hilfreich für die Systeme den Outer Join nicht direkt unterstützen. Das folgende Beispiel zeigt, wie ma Aufgabe in Beispiel 5.14, bzw. 5.15 noch anders lösen kann.

Beispiel 5.16

```
SELECT mitarbeiter.*, abteilung.abt_nr
   FROM mitarbeiter, abteilung
   WHERE wohnort = stadt
UNION
SELECT mitarbeiter.*, '     '
   FROM mitarbeiter
   WHERE NOT EXISTS
      (SELECT *
         FROM abteilung
         WHERE stadt = wohnort);
```

5.6

6.1 Die INSERT-Anweisung

(1) INSERT INTO tab [(spalte_1,...)]
VALUES (wert_1,...);

(2) INSERT INTO tab [(spalte_1,...)]
SELECT-Anweisung;

Beispiel 6.6

```
INSERT INTO mitarbeiter (m_nr, m_name, m_vorname)
   VALUES (15201,'Lang','Viktor');
```

Beispiel 6.7

```
INSERT INTO mitarbeiter (m_name, m_vorname, abt_nr, m_nr)
       VALUES ('Lotter', 'Wolfgang', 'ei', 8413);
```

Beispiel 6.8

Erstellen Sie die Tabelle aller Abteilungen, die in München ihren Standort haben, und laden Sie sie mit den entsprechenden Reihen aus der Tabelle abteilung.

```
CREATE TABLE muench_abt
    (abt_nr CHAR(4) NOT NULL,
     abt_name CHAR(20) NOT NULL);

INSERT INTO muench_abt(abt_nr, abt_name)
       SELECT abt_nr, abt_name
          FROM abteilung
          WHERE stadt = 'Muenchen';
```

6.2 Die UPDATE-Anweisung

UPDATE tab
SET spalte_1=ausdr_1,[spalte_2=ausdr_2],...
[WHERE bedingung];

Beispiel 6.11

Ändern Sie die Aufgabe des Mitarbeiters mit der Personalnummer 18316 im Projekt p2. Er soll Gruppenleiter dieses Projektes werden.

```
UPDATE arbeiten
   SET aufgabe = 'Gruppenleiter'
   WHERE m_nr = 18316
   AND   pr_nr = 'p2';
```

Beispiel 6.12

Die Finanzmittel aller Projekte sollen geändert und in Schweizer Franken dargestellt werden. (Der augenblickliche Währungskurs soll bei 0,89 SFR für 1 DM sein.).

```
UPDATE projekt
   SET mittel = mittel * 0.89;
```

6.3 Die DELETE-Anweisung

DELETE FROM tab
[WHERE bedingung];

Beispiel 6.16

Die Mitarbeiterin namens Mozer scheidet aus der Firma aus. Löschen Sie zun alle Reihen in der Tabelle arbeiten, die diese Mitarbeiterin betreffen, und d auch die entsprechende Reihe der Tabelle mitarbeiter.

```
DELETE FROM arbeiten
   WHERE m_nr =
   (SELECT m_nr
      FROM mitarbeiter
      WHERE m_name = 'Mozer');

DELETE FROM mitarbeiter
   WHERE m_name = 'Mozer';
```

Beispiel 6.17

Löschen Sie alle Reihen der Tabelle arbeiten.

```
DELETE FROM arbeiten;
```

7.1.1 Erstellen von Views

```
CREATE VIEW view_name [(view_spalte_1,...)]
    AS select_anweisung
    [WITH CHECK OPTION];
```

Beispiel 7.1

Erstellen Sie ein View, das alle Sachbearbeiter der Firma beinhaltet.

```
CREATE VIEW v_sach_arb
    AS SELECT m_nr, pr_nr, einst_dat
    FROM arbeiten
    WHERE aufgabe = 'Sachbearbeiter';
```

Die Abfragen, die ein View betreffen, werden tatsächlich auf der zugrundeliegenden Basistabelle durchgeführt. (Wir erinnern noch einmal daran, daß ein View physikalisch nicht existiert; es existieren nur Einträge in Systemtabellen, die es definieren.). Folgende Abfrage auf das View **v_sach_arb**

```
SELECT m_nr
   FROM v_sach_arb
   WHERE pr_nr = 'p2';
```

wird vom System auf Grund der Definition des Views in die Abfrage auf die zugrundeliegende Basistabelle **arbeiten** umgewandelt:

```
SELECT m_nr
   FROM arbeiten
   WHERE pr_nr = 'p2'
   AND aufgabe = 'Sachbearbeiter';
```

Die Anweisung ALTER VIEW existiert in der SQL-Sprache nicht. Das Ändern einer Basistabelle mit der Anweisung ALTER TABLE kann u.U. die Views beinflussen, die aus dieser Basistabelle abgeleitet wurden. Falls ein View mit der Anweisung

```
CREATE VIEW view_name
   AS SELECT *
   FROM tab_name;
```

erstellt wurde, werden alle Änderungen der Basistabelle dazu führen, daß die Struktur des Views nicht mehr der Struktur der zugrundeliegenden Basistabelle entspricht. Deswegen ist es grundsätzlich empfehlenswert:

- SELECT in der CREATE VIEW-Anweisung immer mit der Liste aller Spalten zu schreiben;
- nach der Änderung einer Tabelle alle dazugehörigen Views zu überprüfen und diejenigen, die nicht mehr der Basistabelle entsprechen, zu löschen und wieder zu erstellen.

Beispiel 7.2

Leiten Sie aus der Basistabelle **projekt** ein View ab, bei dem die Spalte **mittel** nicht sichtbar ist.

```
CREATE VIEW v_teil_pr (nummer, name)
  AS SELECT pr_nr, pr_name
       FROM projekt;
```

Beispiel 7.6

Erstellen Sie ein View, das die Personalnummern aller Mitarbeiter enthält, die im Projekt Apollo arbeiten. Diese Aufgabe soll mit Hilfe des Views **v_teil_pr** (Beispiel 7.2) gelöst werden.

```
CREATE VIEW v_arb_teilpr
   AS SELECT m_nr
   FROM arbeiten, v_teil_pr
   WHERE arbeiten.pr_nr = v_teil_pr.pr_nr
   AND pr_name = 'Apollo';
```

5.7

7.3 Ändern eines Views

Die optionale Angabe „WITH CHECK OPTION" in der CREATE VIEW-Anweisung prüft alle Reihen, die über das View in der Basistabelle eingefügt oder geändert werden. Dabei wird die Bedingung in der SELECT-Anweisung innerhalb CREATE VIEW überprüft und, falls sie nicht erfüllt ist, das Ändern, bzw. Einfügen der Reihen mit einer Fehlermeldung abgewiesen. (SQL/DS unterstützt die Angabe „WITH CHECK OPTION" nicht.).

Falls die Angabe „WITH CHECK OPTION" fehlt, werden alle Reihen ohne Überprüfung über das View in der Basistabelle eingefügt bzw. geändert. Es werden also auch die Reihen eingefügt bzw. geändert, die die Bedingung in der SELECT-Anweisung nicht erfüllen. Diese Reihen können anschließend mit demselben View nicht abgefragt werden. Die Beispiele mit der Angabe „WITH CHECK OPTION" befinden sich in Abschnitt 7.3.

Falls das Einfügen der Reihen in der zugrundeliegenden Basistabelle mit Hilfe eines Views durchgeführt wird, gelten dafür folgende Einschränkungen:

a) das View darf nur aus einer einzigen Tabelle abgeleitet werden;
b) keine Spalte des Views darf aus einer Aggregatfunktion abgeleitet werden;
c) keine Spalte des Views darf aus einer skalaren Funktion, einer Konstante oder einem arithmetischen Ausdruck abgeleitet werden;
d) die SELECT-Anweisung innerhalb CREATE VIEW darf die Angabe DISTINCT nicht enthalten;
e) die SELECT-Anweisung innerhalb CREATE VIEW darf die GROUP BY-Klausel nicht enthalten.

7.3.1 INSERT-Anweisung und View

Beispiel 7.12

```
CREATE VIEW v_arb_1988
 AS SELECT m_nr, pr_nr, einst_dat
     FROM arbeiten
     WHERE einst_dat BETWEEN '01.01.1988' AND '31.12.1988'
        WITH CHECK OPTION;

INSERT INTO v_arb_1988
   VALUES (22334, 'p2', '15.04.1989');
```

In Beispiel 7.12 wird die Rolle der „WITH CHECK OPTION"-Angabe gezeigt. In der INSERT-Anweisung dieses Beispiels wird überprüft, ob der Datenwert der Spalte **einst_dat** ('15.04.1989') die Bedingung in der WHERE-Klausel der SELECT-Anweisung erfüllt. Da dies nicht der Fall ist, wird die INSERT-Anweisung mit einer Fehlermeldung abgewiesen.

7.3.2 UPDATE-Anweisung und View

Beispiel 7.16

```
CREATE VIEW v_pr_100
   AS SELECT pr_nr, mittel
         FROM projekt
         WHERE mittel > 100000
         WITH CHECK OPTION;

UPDATE v_pr_100
   SET mittel = 92500
   WHERE pr_nr = 'p2';
```

Die UPDATE-Anweisung in Beispiel 7.16 wird abgewiesen, weil der geänderte Datenwert der Spalte **mittel** in Projekt **p2** die WHERE-Bedingung nicht erfüllt.

7.3.3 DELETE-Anweisung und View

Beispiel 7.18

```
CREATE VIEW v_arb_p1
   AS SELECT m_nr, aufgabe
         FROM arbeiten
         WHERE pr_nr = 'p1';

DELETE FROM v_arb_p1
   WHERE aufgabe = 'Sachbearbeiter';
```

Die DELETE-Anweisung in Beispiel 7.18 wird vom System in folgende Anweisung umgewandelt:

```
DELETE FROM arbeiten
   WHERE pr_nr = 'p1'
   AND aufgabe = 'Sachbearbeiter';
```

8.1.1 Die Anweisungen zur Zugriffssteuerung

```
CREATE [UNIQUE] INDEX index_name
  ON tabelle (sp_1 [ASC|DESC] [,sp_2 [ASC|DESC] ...])
      [weitere_optionen];
```

Beispiel 8.1

Erstellen Sie einen Index für die Spalte **m_nr** der Tabelle **mitarbeiter**.

```
CREATE INDEX i_mit_mnr
    ON mitarbeiter (m_nr);
```

Beispiel 8.2

Erstellen Sie einen zusammengesetzten Index für die Spalten **m_nr** und **pr_nr** der Tabelle **arbeiten**. Die Werte in den zusammenhängenden Spalten **m_nr** und **pr_nr** dürfen nicht mehrfach vorkommen.

```
CREATE UNIQUE INDEX i_arb_mpr
    ON arbeiten (m_nr, pr_nr);
```

9.1.1 Die Anweisung GRANT

```
GRANT tab_recht_liste ON tab TO kennung_liste|PUBLIC
                [WITH GRANT OPTION];
```

genommt

```
GRANT db_recht TO kennung|PUBLIC
                [WITH GRANT OPTION];
```

nicht genommt

- SELECT,
- UPDATE [(spalte_1,...)],
- INSERT,
- DELETE,
- ALTER,
- INDEX und
- ALL [PRIVILEGES].

genommt

- CONNECT,
- RESOURCE und
- DBA.

CONNECT definiert ein sehr eingeschränktes Datenbank-Zugriffsrecht. Der Benutzer, der dieses Recht hat, kann:

- die Verbindung zur Datenbank, für welche dieses Recht gilt, aufbauen;
- Views erzeugen und
- alle einzelnen Tabellen-Zugriffsrechte durchführen, mit Ausnahme von INDEX.

Zwischen Datenbank- und Tabellen-Zugriffsrecht existiert ein enger Zusammenhang. Ein Tabellen-Zugriffsrecht reicht allein nicht aus, um einem Benutzer den Zugriff auf eine Tabelle zu ermöglichen. Dieser Benutzer muß mindestens über das Datenbank-Zugriffsrecht CONNECT verfügen, um die vergebenen Tabellen-Zugriffsrechte verwenden zu können. Dementsprechend reicht das Tabellen-Zugriffsrecht INDEX zusammen mit dem Datenbank-Zugriffsrecht CONNECT nicht aus, um einen Index zu erstellen, weil CONNECT das Erstellen von Indexen nicht umfaßt. Auf diese Weise ist es möglich, eine differenzierte Vergabe von Zugriffrechten für verschiedene Benutzer zu erreichen.

RESOURCE umfaßt alle Datenbank-Zugriffsrechte wie CONNECT, sowie zwei weitere:

- das Erzeugen der Basistabellen und
- das Erzeugen von Indexen.

Das Datenbank-Zugriffsrecht DBA ist das umfangreichste Zugriffsrecht, das existiert. Der Benutzer, dem dieses Zugriffsrecht vergeben wurde, kann alle (legalen) Operationen auf einer Datenbank durchführen.

9.1.2 Die Anweisung REVOKE

Mit der Anweisung REVOKE können die vergebenen Zugriffsrechte entz werden. Diese Anweisung hat zwei Formen, die den Formen der GR/ Anweisung entsprechen:

```
REVOKE tab_recht_liste ON tab
    FROM kenn_liste | PUBLIC;

REVOKE db_recht FROM kenn_liste;
```

9.3 Transaktionen

Mit der Anweisung

```
COMMIT WORK;
```

wird eine Transaktion beendet, und alle Änderungen, die innerhalb der Transakt angegeben sind, werden durchgeführt.

Mit der Anweisung

```
ROLLBACK WORK;
```

werden alle Anweisungen innerhalb einer Transaktion rückgängig gemacht.

Die beiden soeben beschriebenen Anweisungen zur Beendigung einer Tran tion gelten für alle Systeme. Der Beginn einer Transaktion wird bei jedem Sys unterschiedlich behandelt

aus:
Petković, Dušan:
SQL – die Datenbanksprache.
Hamburg: McGrawHill 1990

Anhang A

Die Beispieldatenbank

Die Beispieldatenbank beinhaltet die Datenwerte einer Firma, deren Mitarbeiter gewissen Abteilungen zugehören und gleichzeitig in Projekten, die unabhängig von den Abteilungen sind, arbeiten. Dementsprechend existieren in der Beispieldatenbank insgesamt vier Tabellen:

- **abteilung,**
- **projekt,**
- **arbeiten** und
- **mitarbeiter.**

Die Tabelle **abteilung** beinhaltet drei Spalten und drei Reihen:

abteilung

abt_nr	name	stadt
a1	Beratung	Muenchen
a2	Diagnose	Muenchen
a3	Freigabe	Stuttgart

Die Tabelle **projekt** enthält genauso drei Spalten und drei Reihen:

projekt

pr_nr	pr_name	Mittel
p1	Apollo	120000
p2	Gemini	95000
p3	Merkur	186500

Die Tabelle **arbeiten** beinhaltet 4 Spalten und 11 Reihen:

arbeiten

m_nr	pr_nr	aufgabe	einst_dat
10102	p1	Projektleiter	01.10.1988
10102	p3	Gruppenleiter	01.01.1989
25348	p2	Sachbearbeiter	15.02.1988
18316	p2		01.06.1989
29346	p2		15.12.1987
2581	p3	Projektleiter	15.10.1989
9031	p1	Gruppenleiter	15.04.1989
28559	p1		01.08.1988
28559	p2	Sachbearbeiter	01.02.1989
9031	p3	Sachbearbeiter	15.11.1988
29346	p1	Sachbearbeiter	01.04.1989

Die Tabelle **mitarbeiter** enthält 4 Spalten und 7 Reihen:

mitarbeiter

m_nr	m_name	m_vorname	abt_nr
25348	Keller	Hans	a3
10102	Huber	Petra	a3
18316	Mueller	Gabriele	a1
29346	Probst	Andreas	a2
9031	Meier	Rainer	a2
2581	Kaufmann	Brigitte	a2
28559	Mozer	Sibille	a1

In einigen Beispielen des Buches ist die erweiterte Beispieldatenbank benutzt worden. Der einzige Unterschied zwischen der erweiterten und der ursprünglichen Datenbank ist, daß die erste eine zusätzliche Spalte in der Tabelle **mitarbeiter** hat:

mitarbeiter

m_nr	m_name	m_vorname	abt_nr	wohnort
25348	Keller	Hans	a3	Muenchen
10102	Huber	Petra	a3	Landshut
18316	Mueller	Gabriele	a1	Rosenheim
29346	Probst	Andreas	a2	Augsburg
09031	Meier	Rainer	a2	Augsburg
02581	Kaufmann	Brigitte	a2	Muenchen
28559	Mozer	Sibille	a1	Ulm

A.4.1 Wählen Sie alle Reihen der Tabellen **arbeiten** und **mitarbeiter** aus.

A.4.2 Finden Sie die Personalnummer aller Sachbearbeiter.

A.4.3 Finden Sie die Personalnummer der Mitarbeiter, die in Projekt **p2** arbeiten und deren Personalnummer kleiner als 10000 ist.

A.4.4 Finden Sie die Personalnummer der Mitarbeiter, die nicht im Jahr 1988 in ihr Projekt eingesetzt sind.

A.4.5 Finden Sie Personalnummer aller Mitarbeiter, die in Projekt **p1** eine leitende Aufgabe (Gruppen- oder Projektleiter) haben.

A.4.6 Finden Sie das Einstellungsdatum der Mitarbeiter in Projekt **p2**, deren Aufgabe noch nicht festgelegt ist.

A.4.7 Finden Sie Personalnummer, Namen und Vornamen aller Mitarbeiter, deren Name mit „M", bzw. „H" anfängt und mit „er" endet.

A.4.8 Nennen Sie die Personalnummer aller Mitarbeiter, dessen Standort Stuttgart ist.

A.4.9 Finden Sie Namen und Vornamen aller Mitarbeiter, die ab 01.04.1989 eingesetzt worden sind.

A.4.10 Gruppieren Sie alle Abteilungen auf Grund ihres Standortes.

A.4.11 Nennen Sie die größte Personalnummer eines Mitarbeiters.

A.4.12 Welche Aufgabe üben mehr als zwei Mitarbeiter aus?

A.4.13 Finden Sie die Personalnummer aller Mitarbeiter, die entweder Sachbearbeiter sind oder der Abteilung **a3** gehören.

A.4.14 Warum ist folgende Aufgabe falsch

```
SELECT pr_name
   FROM projekt
   WHERE pr_nr =
   (SELECT pr_nr
      FROM arbeiten
      WHERE aufgabe = 'Sachbearbeiter');
```

Wie sollte diese Aufgabe richtig lauten?

A.5.1 Erstellen Sie ein:
- Equijoin,
- Natürlicher Join und ein
- Kartesisches Produkt

für die Tabellen **projekt** und **arbeiten**.

A.5.2 Finden Sie Personalnummer und Aufgabe aller Mitarbeiter, die im Projekt Gemini arbeiten.

A.5.3 Finden Sie Namen und Vornamen aller Mitarbeiter, die entweder Beratung oder Diagnose durchführen.

A.5.4 Finden Sie das Einstellungsdatum der Mitarbeiter, die zu Abteilung **a2** gehören und in ihrem Projekt Sachbearbeiter sind.

A.5.5 Finden Sie die Namen des Projekts, in dem zwei oder mehrere Sachbearbeiter arbeiten.

A.5.6 Nennen Sie Namen und Vornamen der Mitarbeiter, die Gruppenleiter sind und im Projekt Merkur arbeiten.

A.5.7 Finden Sie in der erweiterten Beispieldatenbank die Personalnummer der Mitarbeiter, die im gleichen Ort wohnen und zu derselben Abteilung gehören.

A.5.8 Finden Sie die Personalnummer aller Mitarbeiter, die zur Abteilung Freigabe gehören. Lösen Sie diese Aufgabe mit Hilfe
a) des Join-Operators
b) der korrelierten Unterabfrage.

A.6.1 Erstellen Sie eine neue Tabelle aller Mitarbeiter, die in den Projekten **a1** und **a2** arbeiten, und laden Sie sie mit den entsprechenden Reihen der Tabelle **mitarbeiter**.

A.6.2 Erstellen Sie eine neue Tabelle aller Mitarbeiter, die im Jahr 1989 eingestellt worden sind, und laden Sie sie mit den entsprechenden Reihen aus der Tabelle **mitarbeiter**.

A.6.3 Ändern Sie die Aufgabe aller Gruppenleiter in Projekt **p1**. Sie sollen ab sofort als Sachbearbeiter tätig sein.

A.6.4 Die Mittel aller Projekte sind bis auf weiteres nicht festgelegt. Weisen Sie den Mitteln den NULL-Wert zu.

A.6.5 Ändern Sie die Aufgabe der Mitarbeiterin mit der Personalnummer 28559. Sie soll ab sofort in allen Projekten Gruppenleiterin werden.

A.6.6 Löschen Sie alle Reihen der Tabelle **abteilung**, deren Standort München ist.

A.6.7 Das Projekt **p3** ist beendet worden. Löschen Sie zunächst alle Daten der Mitarbeiter in der Tabelle **mitarbeiter**, die in diesem Projekt gearbeitet haben und danach auch die entsprechende Reihe der Tabelle **projekt**.

A.7.1 Erstellen Sie ein View, das die Daten aller Mitarbeiter enthält, die im Jahr 1988 eingestellt worden sind.

A.7.2 Erstellen Sie ein View, das die Daten aller Mitarbeiter enthält, die der Abteilung **a3** angehören.

A.7.3 Erstellen Sie ein View, das Namen und Vornamen aller Mitarbeiter beinhaltet, die im Projekt **p3** arbeiten.

A.7.4 Überprüfen Sie, ob Ihre Lösung der Aufgabe A.7.3 ein View darstellt, das modifizierbar ist. Ist das nicht der Fall, erstellen Sie ein solches neues View.

A.7.5 Erstellen Sie ein View, das Personalnummer und Aufgabe aller Mitarbeiter enthält, die im Projekt Merkur arbeiten.

A.7.6 Erstellen Sie ein View, das Namen und Vornamen aller Mitarbeiter enthält, deren Personalnummer kleiner als 10000 ist. Das View soll die WHERE-Klausel jeder UPDATE-, bzw. DELETE-Anweisung überprüfen.

A.7.7 Schreiben Sie für das in A.7.6 erstellte View eine INSERT-Anweisung, die vom System akzeptiert wird, und eine DELETE-Anweisung, die abgewiesen wird.

A.8.1 Erstellen Sie, unter der Annahme, daß alle Tabellen der Beispieldatenbank eine sehr große Anzahl von Reihen haben, die notwendigen Indexe für folgende SELECT-Anweisungen:

a)
```
SELECT m_nr, m_name, m_vorname
   FROM mitarbeiter
   WHERE m_name = 'Kaufmann';
```

b)
```
SELECT m_nr, m_name, m_vorname
   FROM mitarbeiter
   WHERE  m_name = 'Meier'
   AND m_vorname = 'Rainer';
```

c)
```
SELECT aufgabe
   FROM arbeiten, mitarbeiter
   WHERE arbeiten.m_nr = mitarbeiter.m_nr;
```

d)
```
SELECT m_name, m_vorname
   FROM mitarbeiter, abteilung
   WHERE mitarbeiter.abt_nr=abteilung.abt_nr
   AND abt_name = 'Beratung';
```

A.9.1 Die Tabelle **systeme** wird mit der folgenden CREATE TABLE-Anweisung erstellt:

```
CREATE TABLE systeme
  (s_name CHAR(15) NOT NULL,
   version CHAR(5) NOT NULL,
   hersteller CHAR(20),
   ort CHAR(20),
   preis DECIMAL (9));
```

Vergeben Sie den genannten Benutzern folgende Zugriffsrechte:

a) Dem Benutzer **gabi** alle Rechte für die obige Tabelle.
b) Dem Benutzer **rainer** SELECT-Rechte für die ganze Tabelle.
c) Dem Benutzer **alex** INSERT und DELETE-Rechte für die ganze Tabelle.
d) Dem Benutzer **peter** UPDATE-Rechte für die Spalten **hersteller** und **preis**. Dieser Benutzer darf diese Rechte weiteren Benutzern vergeben.
e) Dem Benutzer **juergen** die Rechte zur Erstellung der Indexe für die Tabelle **systeme**.

A.9.2 Welche Datenbankrechte müssen die Benutzer **gabi**, **rainer**, **alex**, **peter** und **juergen** wenigstens haben, damit sie ihre Tabellen-Zugriffsrechte auch verwenden können?

A.9.3 Entziehen Sie den Benutzern **gabi** und **juergen** die in A.9.1 vergebenen Zugriffsrechte.

5.11

.3.3
```
CREATE TABLE systeme
(sys__name CHAR(15) NOT NULL,
 version INTEGER NOT NULL,
 hersteller CHAR(20),
 ort CHAR(20));
```

.4.1
```
SELECT *
    FROM arbeiten;

SELECT *
    FROM mitarbeiter;
```

.4.2
```
SELECT m__nr
    FROM arbeiten
    WHERE aufgabe = 'Sachbearbeiter';
```

.4.3
```
SELECT m__nr
    FROM arbeiten
    WHERE pr__nr = 'p2'
    AND m__nr < 10000;
```

.4.4
```
SELECT m__nr
    FROM arbeiten
    WHERE einst__dat NOT BETWEEN
    '01.01.1988' AND '31.12.1988';
```

.4.5
```
SELECT m__nr
    FROM arbeiten
    WHERE pr__nr = 'p1'
    AND (aufgabe='Projektleiter'
    OR   aufgabe='Gruppenleiter');
```

A.4.6
```
SELECT einst__dat
    FROM arbeiten
    WHERE pr__nr='p2'
    AND aufgabe IS NULL;
```

A.4.7
```
SELECT m__nr, m__name, m__vorname
    FROM mitarbeiter
    WHERE (m__name LIKE 'M%'
    OR m__name LIKE 'H%')
    AND m__name LIKE '%er';
```

A.4.8
```
SELECT m__nr
    FROM mitarbeiter
    WHERE abt__nr =
    (SELECT *
        FROM abteilung
        WHERE stadt = 'Stuttgart');
```

A.4.9
```
SELECT m__name, m__vorname
    FROM mitarbeiter
    WHERE m__nr =
    (SELECT m__nr
        FROM arbeiten
        WHERE einst__dat ='01.04.1989');
```

A.4.10
```
SELECT stadt
    FROM abteilung
    GROUP BY stadt;
```

A.4.11
```
SELECT max(m__nr)
    FROM mitarbeiter;
```

A.4.12
```
SELECT aufgabe
    FROM arbeiten
    GROUP BY aufgabe
    HAVING COUNT(*) > 1;
```

A.4.13
```
SELECT m__nr
    FROM arbeiten
    WHERE aufgabe = 'Sachbearbeiter';
```

A.4.14 Die innere SELECT-Anweisung darf eine einzige Reihe als Ergebnis liefern, falls sie im Zusammenhang mit einem Vergleichsoperator erscheint (in diesem Fall „="). Das Gleichheitszeichen muß durch den IN-Operator ersetzt werden.

A.5.1
```
SELECT *
    FROM projekt, arbeiten
    WHERE projekt.pr__nr=arbeiten.pr__nr;

SELECT projekt.*,m__nr,aufgabe, einst__dat
    FROM projekt, arbeiten
    WHERE projekt.pr__nr=arbeiten.pr__nr;

SELECT *
    FROM projekt,arbeiten;
```

A.5.2
```
SELECT m__nr, aufgabe
    FROM arbeiten, projekt
    WHERE arbeiten.pr__nr = projekt.pr__nr
    AND pr__name='Gemini';
```

A.5.3
```
SELECT m__name,m__vorname
    FROM mitarbeiter, abteilung
    WHERE mitarbeiter.abt__nr=abteilung.abt__nr;
```

A.5.4
```
SELECT einst__dat
    FROM arbeiten, mitarbeiter
    WHERE arbeiten.m__nr=mitarbeiter.m__nr
    AND aufgabe = 'Sachbearbeiter'
    AND abt__nr = 'a2';
```

A.5.5
```
SELECT pr__name
    FROM projekt
    WHERE pr__nr =
    (SELECT pr__nr
        FROM arbeiten
        WHERE aufgabe='Sachbearbeiter'
        GROUP BY pr__nr
        HAVING COUNT(*) > 1);
```

A.5.6
```
SELECT m__name, m__vorname
    FROM mitarbeiter,arbeiten,projekt
    WHERE mitarbeiter.m__nr=arbeiten.m__nr
    AND arbeiten.pr__nr=projekt.pr__nr
    AND pr__name = 'Merkur'
    AND aufgabe = 'Gruppenleiter';
```

A.5.7
```
SELECT m__nr
    FROM mitarbeiter a, mitarbeiter b
    WHERE a.wohnort = b.wohnort
    AND a.abt__nr = b.abt__nr;
```

A.5.8
```
SELECT m__nr
    FROM mitarbeiter,abteilung
    WHERE mitarbeiter.abt__nr=abteilung.abt__nr
    AND abt__name = 'Freigabe';
```

```
SELECT m__nr
    FROM mitarbeiter
    WHERE abt__nr =
    (SELECT abt__nr
        FROM abteilung
        WHERE abt__name = 'Freigabe');
```

A.6.1
```
CREATE TABLE mit__a1__a2
 (m__nr INTEGER NOT NULL,
 m__name CHAR(20) NOT NULL,
 m__vorname CHAR(20) NOT NULL);

INSERT INTO mit__a1__a2(m__nr,m__name,m__vorname)
    SELECT m__nr,m__name,m__vorname
        FROM mitarbeiter
        WHERE abt__nr IN ('a1','a2');
```

A.6.2
```
CREATE TABLE mit__1989
    (m__nr INTEGER NOT NULL,
    m__name CHAR(20) NOT NULL,
    m__vorname CHAR(20) NOT NULL);

INSERT INTO mit__1989(m__nr,m__name,m__vorname)
    SELECT mitarbeiter.m__nr,m__name,m__vorname
    FROM mitarbeiter, arbeiten
    WHERE mitarbeiter.m__nr=arbeiten.m__nr
    AND einst__dat BETWEEN '01.01.1989'
                       AND '31.12.1989';
```

A.6.3
```
UPDATE arbeiten
    SET aufgabe = 'Sachbearbeiter'
    WHERE aufgabe = 'Gruppenleiter';
```

A.6.4
```
UPDATE projekt
    SET mittel = NULL;
```

A.6.5
```
UPDATE arbeiten
    SET aufgabe = 'Gruppenleiter'
    WHERE m__nr = 28559;
```

A.6.6
```
DELETE FROM abteilung
    WHERE stadt = 'Muenchen';
```

A.6.7
```
DELETE FROM mitarbeiter
    WHERE m__nr IN
    (SELECT m__nr
        FROM arbeiten
        WHERE pr__nr = 'p3');

DELETE FROM projekt
    WHERE pr__nr = 'p3';
```

A.7.1
```
CREATE VIEW v__mit__1988(nummer,name,vorname,abt)
  AS SELECT mitarbeiter.m__nr,m__name,m__vorname,abt__nr
        FROM mitarbeiter, arbeiten
        WHERE mitarbeiter.m__nr=arbeiten.m__nr
        AND einst__dat BETWEEN '01.01.1988'
                           AND '31.12.1988';
```

A.7.2
```
CREATE VIEW v__mit__a3
    AS SELECT m__nr, m__name, m__vorname
        FROM mitarbeiter
        WHERE abt__nr = 'a3';
```

A.7.3
```
CREATE VIEW v__mit__p3
    AS SELECT m__name,m__vorname
        FROM mitarbeiter
        WHERE m__nr =
        (SELECT m__nr
            FROM arbeiten
            WHERE pr__nr='p3');
```

A.7.5
```
CREATE VIEW v__arb__pr
    AS SELECT m__nr, aufgabe
        FROM arbeiten, projekt
        WHERE arbeiten.pr__nr=projekt.pr__nr
        AND pr__name = 'Merkur';
```

A.7.6
```
CREATE VIEW v__mit__nr
  AS SELECT m__nr, m__name, m__vorname
      FROM mitarbeiter
      WHERE m__nr < 10000
      WITH CHECK OPTION;
```

A.7.7
```
INSERT INTO v__mit__nr
    VALUES (8888, 'Holl', 'Martina');

DELETE FROM v__mit__nr
    WHERE m__nr = 28559;
```

A.8.1

a)
```
CREATE INDEX i__mit__mname
    ON mitarbeiter(m__name);
```

b)
```
CREATE INDEX i__mit__nam__vor
    ON mitarbeiter(m__name, m__vorname);
```

c)
```
CREATE INDEX i__mit__mnr
    ON mitarbeiter(m__nr);
```

```
CREATE INDEX i__arb__mnr
    ON arbeiten(m__nr);
```

d)
```
CREATE INDEX i__mit__abtnr
    ON mitarbeiter(abt__nr);

CREATE INDEX i__abt__abtnr
    ON abteilung(abt__nr);

CREATE INDEX i__abt__name
    ON abteilung(abt__name);
```

```
CREATE TABLE STUDENT
(S_MATRNR            INTEGER NOT NULL,      Primärschlüssel
 S_STUDNAME          CHAR(30),
 S_STUDPLZ           CHAR(4),
 S_STUDORT           CHAR(30));

CREATE TABLE BUCH
(B_SIGNATUR          CHAR(15),              Primärschlüssel
 B_FACHGEBIET        CHAR(02),
 B_AUTORNAME         CHAR(40),
 B_TITEL             CHAR(65),
 B_ERSCHEINUNGSORT   CHAR(30),
 B_ERSCHEINUNGSJAHR  INTEGER);

CREATE TABLE AUSLEIHVORGANG
(A_MATRNR            INTEGER NOT NULL,
 A_SIGNATUR          CHAR(15),              Primärschlüssel
 A_AUSLEIHDATUM      DATE,
 A_RÜCKGABEDATUM     DATE,
 A_MAHNZAHL          INTEGER);

CREATE TABLE STATISTIK
(STAT_SIGNATUR       CHAR(15),              Primärschlüssel
 STAT_AUSLEIHDATUM   DATE,                  Primärschlüssel
 STAT_STUDPLZ        CHAR(4));
```

Formulieren Sie - ausgehend von den vier oben definierten Basistabellen - SQL-Befehle für die folgenden Auswertungen, die zu einem beliebigen Stichtag angefertigt werden sollen.
Geben Sie an, ob Sie sich auf ANSI-SQL oder eine spezielle Implementation beziehen.

3.1 Anzahl der Ausleihvorgänge, die am Stichtag erfolgt sind.

3.2 Anzahl der Bücher der Bibliothek, die am Stichtag im Besitz von Studenten (= entliehen) sind.

3.3 Anzahl der Studenten, die am Stichtag von der Bibliothek mit mindestens einem Ausleihvorgang bedient worden sind.

3.4 Anzahl der Studenten, die am Stichtag mindestens ein Buch der Bibliothek im Besitz haben.

3.5 Anzahl der Studenten, die am Stichtag kein Buch der Bibliothek im Besitz haben.

3.6 Liste aller Studenten (A_MATRNR, S_STUDNAME, Anzahl), bei denen Mahnungen erfolgt sind und die die angemahnten Bücher noch nicht zurückgebracht haben, aufsteigend sortiert nach der Anzahl der Mahnschreiben je Student.
Hinweis: A_MAHNZAHL enthält die Anzahl der versandten Mahnschreiben je Ausleihvorgang.

3. In einem Sammelabrechnungssystem werden einzelne Lieferpositionen tagesbezogen für die spätere monatliche Fakturierung gespeichert. Die Löschung der Positionszeilen erfolgt nicht schon bei der Rechnungsschreibung, sondern erst nach einer statistischen Auswertung am Kalenderjahresende.

Ein Ausschnitt aus einem solchen System sei durch folgende drei Basistabellen gegeben:

```
CREATE TABLE ARTIKEL
(A_ARTNR       INTEGER NOT NULL,
 A_ARTBEZEICH  CHAR(25));

CREATE TABLE KUNDE
(K_KDNR        INTEGER NOT NULL,
 K_KDNAME      CHAR(30),
 K_KDPLZ       INTEGER,
 K_KDORT       CHAR(30),
 K_KDSTRASSE   CHAR(30));

CREATE TABLE LIEFERPOSITION
(L_KDNR        INTEGER NOT NULL,
 L_ARTNR       INTEGER NOT NULL,
 L_DATUM       DATE,
 L_MENGE       INTEGER);
```

Für die Statistik des Jahres 1990 werden u.a. folgende Übersichten benötigt:

1. Liste aller 1990 verkauften Artikel (Nummer, Bezeichnung) mit der jeweiligen Gesamtumsatzmenge im Jahre 1990 sortiert nach der Artikelnummer

2. alle Artikel (Nummer, Bezeichnung), die 1990 nicht verkauft worden sind, sortiert nach der Artikelnummer

3. alle Kunden (Nummer), die 1990 nichts gekauft haben, sortiert nach der Kundennummer; Anzahl dieser Kunden

Formulieren Sie - ausgehend von den drei oben definierten Basistabellen - SQL-Befehle für diese Auswertungen.
Geben Sie an, ob Sie sich auf ANSI-SQL oder INGRES-SQL beziehen.

4. Wie muß die Lösung von 1 geändert werden, um die gleiche Liste absteigend sortiert nach der jeweiligen Gesamtumsatzmenge zu erhalten? (1 Zeile)

5. Wie muß die Lösung von 1 geändert werden, um die gleiche Liste beschränkt auf den einzigen Großkunden (KundenNr 10000) zu erhalten? (1 Zeile)

6. Welches SQL-Konzept, das in verschiedenen Datenbanksystemen in unterschiedlicher Syntax ausgedrückt wird, ermöglicht es, die Listen 1 und 2 in einem Arbeitsgang zu erzeugen? (1 Zeile)

A.4.8

```
SELECT m_nr
        FROM mitarbeiter
        WHERE abt_nr
        IN      (
                SELECT abt_nr
                        FROM abteilung
                        WHERE stadt='Stuttgart'
                );
```

Im inneren Select werden aus der Tabelle der Abteilungen (abteilung) alle Abteilungsnummern (abt_nr) ausgewählt, deren Standort (stadt) Stuttgart ist.
Im äußeren Select werden alle Mitarbeiternummern (m_nr) aus der Mitarbeitertabelle (mitarbeiter) ausgewählt, deren Abteilungsnummern (abt_br) im inneren Select herausgefiltert wurden.

A.4.9

```
SELECT m_name, m_vorname
        FROM   mitarbeiter
        WHERE m_nr
        IN      (
                SELECT m_nr
                        FROM arbeiten
                        WHERE einst_dat>#03/31/89#
                );
```

Im inneren Select werden aus der Tabelle der Projekt-Mitarbeiter-Zuordnungen (arbeiten) alle Mitarbeiternummern (m_nr) der Mitarbeiter ausgewählt, die später als 31.03.1989 eingestellt wurden.
Im äußeren Select werden alle Vor- und Nachnamen (m_name, m_vorname) aus der Mitarbeitertabelle (mitarbeiter) ausgewählt, deren Mitarbeiternummern (m_nr) im inneren Select herausgefiltert wurden.

A.4.10

```
SELECT stadt
        FROM abteilung
        GROUP BY stadt;
```

Aus der Tabelle der Abteilungen (abteilung) werden alle Städte ausgewählt (stadt) und nach den Namen der Städten gruppiert.

A.4.11

```
SELECT MAX(m_nr)
        FROM abteilung;
```

Die größte Personalnummer (m_nr) aus der Tabelle der Abteilungen (abteilung) wird ausgegeben.

A.4.12

```
SELECT aufgabe
        FROM arbeiten
        GROUP BY aufgabe
        HAVING COUNT(*)>1;
```

Aus der Tabelle der Mitarbeiter-Projekt-Zuordnungen (arbeiten) werden die Aufgaben (aufgabe) ausgegeben, die mehr als ein Mitarbeiter ausübt.

A.4.13

```
SELECT DISTINCT m_nr
        FROM arbeiten
        WHERE aufgabe='Sachbearbeiter'
UNION
SELECT m_nr
        FROM mitarbeiter
        WHERE abt_nr='a3';
```

Anmerkung:
Fehler in der Musterlösung.

Im ersten Select werden alle Mitarbeiternummern (m_nr) Aus der Tabelle der Mitarbeiter-Projekt-Zuordnungen (arbeiten) ausgewählt, deren Aufgabe (aufgabe) Sachbearbeiter ist.
Im zweiten Select werden die Nummern der Mitarbeiter (m_nr) aus der Mitarbeitertabelle ausgewählt, die in der Abteilung mit der Nummer (abt_nr) „a3“ arbeiten.
Aus den Teilmengen der Select-Anweisung wird nun mit dem Union-Operator die Vereinigungsmenge gebildet.

5.15

A.4.14

```
SELECT pr_name
       FROM projekt
       WHERE pr_nr
       IN     (
              SELECT pr_nr
                     FROM arbeiten
                     WHERE aufgabe='Sachbearbeiter'
              );
```

Fehler in der Angabe: Da im inneren SELECT mehr als ein Wert aus der Tabelle "arbeiten" herausgefiltert wird (da es in jedem Projekt einen Sachbearbeiter gibt), ist im äußeren SELECT eine Verküpfung mit "=" nicht möglich. Hier ist der IN-Operator korrekt.

A.5.1 **a)**

```
SELECT *
       FROM arbeiten, projekt
       WHERE arbeiten.pr_nr=projekt.pr_nr;
```

Für jeden Mitarbeiter soll der Projektname und die Mittel des Projekts angegeben werden an dem er mitarbeitet (die doppelten Spalten werden mit angezeigt).

b)

```
SELECT projekt.*, m_nr
       FROM arbeiten, projekt
       WHERE arbeiten.pr_nr=projekt.pr_nr;
```

Für jeden Mitarbeiter sollen die gesamten Daten des Projekts angegeben werden an dem er mitarbeitet (die doppelten Spalten werden nicht mit angegezeigt).

c)

```
SELECT mitarbeiter.*, abteilung.*
       FROM mitarbeiter, abteilung
       WHERE mitarbeiter.wohnort<>abteilung.stadt;
```

Kartesisches Produkt aus der Tabelle der Mitarbeiter und der Abteilungen über das Joinfeld Wohnort bzw. Stadt. (Der Sinn bleibt mir verborgen ?!?)

A.5.2

```
SELECT m_nr, aufgabe
       FROM arbeit
       WHERE pr_nr
       =      (
              SELECT pr_nr
                     FROM projekt
                     WHERE pr_name='Gemini'
              );
```

Im inneren Select werden alle Projektnummern (pr_nr) aus der Projekt-Tabelle (projekt) herausgefiltert, deren Name (pr_name) „Gemini" ist.
Im äußeren Select werden alle Nummern (m_nr) und Aufgaben (aufgabe) der Mitarbeiter ausgegeben deren Nummer im inneren Select ermittelt wurde.

A.5.3

```
SELECT m_name, m_vorname
       FROM mitarbeiter, abteilung
       WHERE mitarbeiter.abt_nr=abteilung.abt_nr
              AND    (
                     abteilung.name='Beratung'
                     OR
                     abteilung.name='Diagnose'
                     );
```

Anmerkung:
Fehler in der Musterlösung

Aus der Tabelle der Mitarbeiter (mitarbeiter) werden die Namen (m_name) und Vornamen (m_vorname) ausgegeben, die den Kriterien des folgenden Joins entsprechen:
Die Tabelle der Mitarbeiter (mitarbeiter) und die Tabelle der Abteilungen (abteilung) werden über das Join-Feld Abteilungsnummer (abt_nr) verbunden und zusätzlich werden die Bedingungen gestellt, daß der Name der Abteilung (name) in der Tabelle der Abteilungen (abteilung) entweder „Beratung" oder „Diagnose" ist.

5.16

A.5.4

```
SELECT einst_dat
       FROM mitarbeiter, arbeit
       WHERE mitarbeiter.m_nr=arbeit.m_nr
              AND arbeit.aufgabe='Sachbearbeiter'
              AND mitarbeiter.abt_nr='a2';
```

Aus der Tabelle der Mitarbeiter-Projekt-Zuordnungen (arbeit) werden die Einstellungsdaten (einst_dat) ausgegeben, die folgender Join ermittelt:
Die Mitarbeitertabelle (mitarbeiter) und die Tabelle der Mitarbeiter-Projekt-Zuordnungen (arbeit) werden über das Join-Feld Mitarbeiternummer (m_nr) verbunden und zusätzlich werden die Bedingungen gestellt, daß in der Tabelle „arbeit" die Aufgabe (aufgabe) des Mitarbeiters „Sachbearbeiter" ist und der Mitarbeiter in der Abteilung mit der Nummer (abt_nr) „a2" ist.

A.5.5

```
SELECT pr_name
       FROM projekt, arbeiten
       WHERE projekt.pr_nr=arbeiten.pr_nr
              AND arbeiten.aufgabe='Sachbearbeiter'
       GROUP BY arbeiten.pr_nr, pr_name
       HAVING COUNT(*)>=2;
```

Die Projekttabelle (projekt) und Projekt-Mitarbeiter-Zuordnungstabelle (arbeiten) werden über das Join-Feld „pr_nr" miteinander verknüpft und zusätzlich wird die Bedingung gestellt, daß die Aufgabe (aufgabe) des Mitarbeiters „Sachbearbeiter" ist.
Der durch diesen Join gebildete Datenbestand wird nun nach der Projektnummer gruppiert (pr_nr) und alle Projektnamen (pr_name) der Projekte, die mehr als einen Datensatz in diesem aufgebauten Datenbestand haben werden dann ausgegeben

A.5.6

```
SELECT mitarbeiter.m_name, mitarbeiter.m_vorname
       FROM mitarbeiter, arbeiten, projekt
       WHERE mitarbeiter.m_nr=arbeiten.m_nr
              AND arbeiten.aufgabe='Gruppenleiter'
              AND arbeiten.pr_nr=projekt.pr_nr
              AND projekt.pr_name='Merkur';
```

Aus der Tabelle der Mitarbeiter (mitarbeiter) werden die Namen und Vornamen (m_name, m_vorname) ausgegeben, die folgender Join ermittelt:
Die Tabellen der Mitarbeiter (mitarbeiter) und der Mitarbeiter-Projekt-Zuordnungen (arbeit) werden über das Feld Mitarbeiternummer (m_nr) verknüpft, die Tabellen der Projekte (projekte) und der Mitarbeiter-Projekt-Zuordnungen (arbeit) werden über das Feld Projektnummer (pr_nr) verknüpft. Darüberhinaus werden noch folgende Bedingungen gestellt:
Die Aufgabe des Mitarbeiter muß „Gruppenleiter" sein und der Name des Projekts muß „Merkur" sein.

A.5.7

```
SELECT a.m_nr
       FROM mitarbeiter AS a, mitarbeiter AS b
       WHERE a.m_nr<>b.m_nr
              AND a.wohnort=b.wohnort
              AND a.abt_nr=b.abt_nr;
```

Anmerkung:
Fehler in der Musterlösung

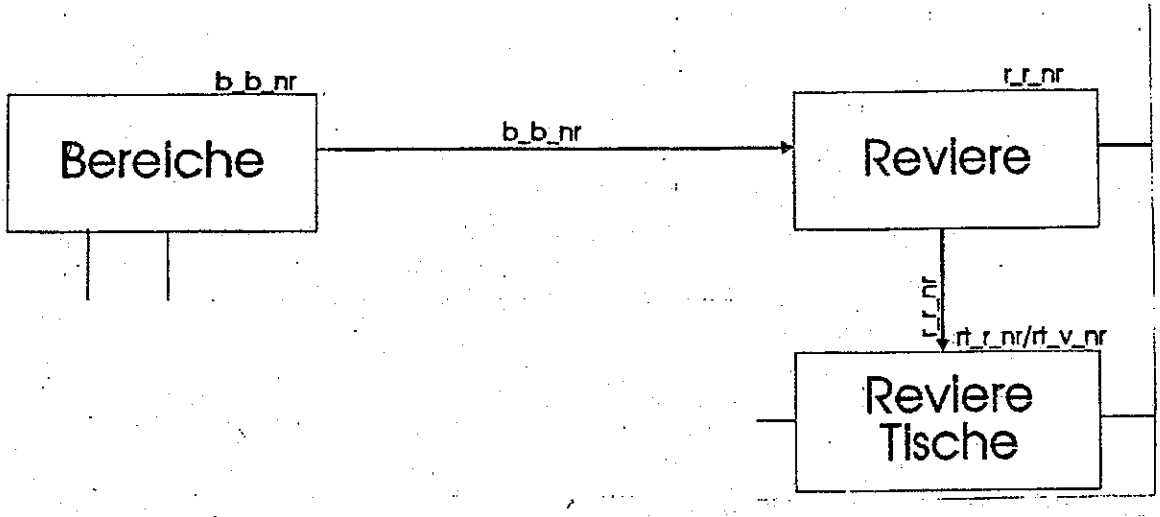
Die Tabelle Mitarbeiter (mitarbeiter) wird mit sich selbst verknüpft mit den Bedingungen:
Alle Nummern (m_nr) der verschiedenen Mitarbeiter, die in den gleichen Abteilungen (abt_nr) arbeiten und deren Wohnort (wohnort) auch gleich ist werden ausgegeben.

A.5.8 a)

```
SELECT m_nr
       FROM mitarbeiter, abteilung
       WHERE mitarbeiter.abt_nr=abteilung.abt_nr
              AND abteilung.name='Freigabe';
```

Die Mitarbeitertabelle (mitarbeiter) und die Abteilungstabelle (abteilung) werden über das Feld Abteilungsnummer (abt_nr) miteinander verknüpft und alle Nummern (m_nr) der Mitarbeiter die in der Abteilung (name) „Freigabe" arbeiten ausgegeben.

Ausschnitt aus dem Datenmodell eines Gastronomieverwaltungssystems



5.1.1.5.Existiert ein Bereich ohne Reviere?

Es soll angezeigt werden:

- Bereichsnummer b_b_nr
- Bereichsbezeichnung b_b_bez

```
SELECT b_b_nr, b_b_bez
FROM bereiche
WHERE b_b_nr NOT IN
(SELECT DISTINCT r_b_nr FROM reviere);
```

Erklärung:

SELECT ...

Nicht´s besonderes.

FROM ...

Nicht´s besonderes.

WHERE b_b_nr NOT IN

Der nachfolgende Subselect liefert eine Liste aller r_b_nr, denen ein Revier zugeordnet ist.

Logisch, daß die gesuchten b_b_nr in dieser Liste nicht vorkommen dürfen. Daher b_b_nr NOT IN (...).

(SELECT DISTINCT r_b_nr FROM reviere);

Hier sucht man zuerst alle r_b_nr aus der Tabelle reviere heraus, denen ein Revier zugeordnet ist.

DISTINCT gibt an, daß bei mehrmaligem Vorkommen der r_b_nr diese nur einmal zu nehmen ist. Funktioniert natürlich auch ohne DISTINCT.

Dieser Subselect wird zuerst ausgeführt. Ist er ausgeführt, steht zwischen den Klammern () eine Liste aller r_b_nr. Ohne DISTINCT wäre die Liste länger.

Und so sieht dann die Ausgabe aus:

B_B_NR	B_B_BEZ
4	Hausverkauf

Es soll angezeigt werden:

- Reviernummer r_r_nr
- Revierbezeichnung r_r_bez
- Tischanzahl exp1

```
SELECT r_r_nr, r_r_bez, (COUNT(*)-1)
FROM reviere, rev_ti
WHERE r_r_nr = rt_r_nr
GROUP BY r_r_nr, r_r_bez;
```

Erklärung:

SELECT ..., (COUNT(*)-1)

OUPS!? (COUNT(*)-1) ist schon korrekt! (4.3. zeigt eine Alternative!)

Zunächst sei zur Erklärung von COUNT(*) ein trivialer Fall geschildert:

Es sollen aus der Tabelle rev_ti
- die rt_r_nr
- und die Anzahl der Tische angegeben werden.

Lösung:

```
select rt_r_nr, count(*)
from rev_ti
group by rt_r_nr;
```

Lesen wir nun die Abfrage. Dann heißt das:

– Nimm die Tabelle rev_ti!	from rev_ti
– Gruppiere diese nach der rt_r_nr!	group by rt_r_nr
– Zähle dann in jeder Gruppe die Zeilen!	count(*)
– Gib die rt_r_nr	select rt_r_nr,
– und die Anzahl der Zeilen aus!	count(*)

So– jetzt darf man sich fragen warum (COUNT(*) – 1)! (4.3. zeigt eine Alternative!) Man erinnert sich, daß die "Stehplätze" die Tischnummer 0 bekommen haben. Und da die Tischnummer 0 einem imaginären Tisch zugeordnet wurde, d.h. einem Tisch der eigentlich gar nicht existiert, darf man den natürlich nicht mitzählen. Aus diesem Grund ist dieser Tisch dann wieder von der ermittelten Tischanzahl (COUNT(*)) wieder abzuziehen (COUNT(*) – 1). Alles klar?

FROM ...

Nicht´s besonderes.

WHERE ...

Ein einfacher Join.

GROUP BY r_r_nr, r_r_bez;

Da in der SELECT – Anweisung die Spaltennamen nicht Parameter der Aggregatfunktion COUNT() sind, müssen diese Spaltennamen zur Gruppierung verwendet werden.

Anders formuliert: Alle Spaltennamen, die nicht Parameter einer Aggregatsfunktion sind, dürfen nur dann in der SELECT – Anweisung vorkommen, wenn sie zur Gruppierung verwendet werden.

Und so sieht dann die Ausgabe aus:

G_R_R_NR	G_R_R_BEZ	EXP1
B1	Bier	0
B2	Cocktail	0
G1	großer	4
G2	kleiner	3
R1	Buffett	1
R2	kleines	4
R3	großes	3

(die fett (G_, EXP1) markierten Buchstaben wurden von dBase hinzugefügt!)

5.1.1.8. Gibt es Reviere, denen keine Tische zugewiesen wurden?

Es soll angezeigt werden:

- Bereichsnummer b_b_nr
- Bereichsbezeichnung b_b_bez
- Revierbezeichnung r_r_bez
- Reviernummer r_r_nr

```
SELECT b_b_nr, b_b_bez, r_r_bez, r_r_nr
FROM bereiche, reviere, rev_ti
WHERE b_b_nr = r_r_nr
AND r_r_nr = rt_r.rev_nr
GROUP BY b_b_nr, b_b_bez, r_r_nr, r_r_bez
HAVING COUNT(*) = 1;
```

Erklärung:

SELECT ...

Nicht´s Neues.

FROM ...

Nicht´s besonderes.

WHERE ...

Zwei einfache Joins.

GROUP BY b_b_nr, b_b_bez, r_r_nr, r_r_bez
HAVING COUNT(*) = 1;

Zunächst sei wieder ein trivialer Fall geschildert:

Es sollen aus der Tabelle rev_ti die rt_t_nr angezeigt werden, denen keine Tische zugewiesen wurden.

Lösung:

```
select rt_r_nr
from rev_ti
group by rt_r_nr
having count(*) = 1;
```

Lesen wir nun die Abfrage. Dann heißt das:

– Nimm die Tabelle rev_ti!	from _ti
– Gruppiere diese nach der rt_r_nr!	group by rt_r_nr
– Zähle dann in jeder Gruppe die Zeilen!	... count(*) ...
– Für die Ausgabe merke nur die rt_r_nr, dessen Gruppe nur aus einer Zeile besteht	having ... = 1
– Gib die rt_r_nr	select rt_r_nr,

HAVING COUNT(*) = 1, "1" weil Reviere, denen kein Tisch zugeordnet wurde, in der Tabelle tische mit der imaginären Tischnummer 0 enthaten sind.

Für die gewünschten Ausgaben müssen nun aber mehrere Tabellen herangezogen werden. Schließlich soll ja auch die b_b_nr, die b_b_bez und die r_r_bez ausgegeben werden.
Hierzu wird aus den drei Tabellen bereiche, reviere und rev_ti eine Tabelle erstellt.
Das erreicht man in der WHERE – Anweisung über die zwei einfachen Joins.
Dann wird diese Tabelle nach der b_b_nr, der b_b_bez, der r_r_nr und der r_r_bez gruppiert.
Jetzt werden die Zeilen in einer Gruppe gezählt.
Und am Schluß wird nur die Gruppe ausgegeben, die die HAVING – Bedingung erfüllt.
D.h. es wird eine Gruppe ausgegeben, die nur aus einer Zeile besteht.

Und so sieht dann die Ausgabe aus:

G_B_B_NR	G_B_B_BEZ	G_R_R_BEZ	G_R_R_NR
1	Bar	Bier	B1
1	Bar	Cocktail	B2

Codd'sche Regeln

0. Grundregel

ein relationales DBMS darf eine Datenbank ausschließlich über seine eigenen relationalen Leistungsmerkmale verwalten (→ Systemtabellen "Metadatenbank")

1. Informationsregel

auf der logischen Ebene sind alle Informationen explizit ausschließlich durch Tabellen dargestellt.

2. Regel für den garantierten Zugriff

jeder Elementarwert muß durch Tabellenname, Primärschlüssel und Spaltenname eindeutig angesprochen werden können.

3. Systematische Behandlung von "Null"-Werten

Nullwerte (= Engl. Blanks, dtsch. Nullen) werden unabhängig vom Datentyp unterstützt, da sie fehlende Informationen in systematischer Weise darstellen. ("unbekannt").

4. dynamischer Online-Katalog, basierend auf dem relationalen Datenmodell

die Beschreibung einer Datenbank erfolgt logisch genauso wie die gewöhnlicher Daten. Daher kann für die Abfrage von Metadaten dieselbe relationale Sprache verwendet werden wie für die Abfrage der regulären Daten.

5. umfassende Daten-Sprache

Es muß 2 1 Sprache mit vorbelegt Syntax für folgende Funktionen geben:

- Datendefinition, Datenstrukturdef. (SQL – DDL / SDL)
- Datenmanipulation (SQL – DML)
- Integritätsbedingungen (DBZ – SQL)
- Transaktionsgrenzen

6. Regel für Änderung der Beschreibung

Alle theoretisch veränderbaren Datenstrukturen müssen auch durch den DBMS geändert werden können.

(z.B. Hinzufügen / Weglassen von Feldern, Verändern von Feldlängen)

7. Kommandos INSERT, UPDATE, DELETE auf höherer Sprachebene

Beim Lesen, Einfügen, Ändern und Löschen von Daten kann eine Basisrelation oder eine abgeleitete Relation als ein einziger Operand behandelt werden.

8. physische Datenunabhängigkeit (Trennung von log. Ebene)

Anwendungsprogramme und Terminaloperationen dürfen auf logischer Ebene nicht beeinträchtigt werden, wenn ~~systembedingte Informationserhaltenden~~ Änderungen an den in den physischen Darstellungen der Daten oder in den Zugriffsmethoden durchgeführt werden.

9. Logische Datenunabhängigkeit

(logische vs externe Ebene)

Anwendungsprogramme und Terminalaktivitäten dürfen auf logischer Ebene nicht beeinträchtigt werden, wenn irgendwelche informationserhaltenden Änderungen an den Basistabellen durchgeführt werden.

10. Integritätsunabhängigkeit

Integritätseinschränkungen müssen in der Sprache der relationalen Daten definierbar sein und in einem Schema (aber nicht in den Applikationen) gespeichert werden. → Frame

11. Verteilungsunabhängigkeit

Die Unabhängigkeit der Applikationen von der physischen Verteilung (Lokalisation) der Daten muß gewährleistet sein.

12. keine Möglichkeit zur Umgehung der höheren Sprachebene

Die Verwendung einer Sprache auf niedrigerer Ebene darf es nicht ermöglichen, auf der höheren Sprachebene definierte Integritätsregeln und -beschränkungen zu umgehen.

3 Integritäts-Leistungsmerkmale (Codd)

1. Entitäts-Integrität

Festlegung eines Primärschlüssels für Relationen; Primärschlüssel-Werte dürfen keinen NAK-Wert enthalten.

Ziel: Wiederauffindbarkeit der in der DB entsprechend dem logischen Schlüssel gespeicherten Daten

2. referentielle Integrität

Für jeden Fremdschlüssel einer Relation muß ein Primärschlüssel definiert sein. Jeder Wert des Fremdschlüssels muß einem Wert des Primärschlüssels entsprechen oder ein NAK-Wert sein.

Ziel: Konsistenz der Relationsbeziehungen untereinander (Verbindung von Tabellen)

3. anwenderdefinierte Integrität

Der Anwender kann zusätzliche Integritätsbedingungen definieren, die sich auf Tupel, Tupelmengen oder DB-Zustandswerte beziehen.

9 strukturelle Leistungsmerkmale

1. Daten nur in Form von Relationen

Relationen entsprechen Tabellen mit nicht numerierten Zeilen (Tupeln), mit benannten Spalten (Attributen), ohne Positionskonzepte und ohne Wiederholungsgruppen.

2. Basis-Relationen

Die Basisrelationen repräsentieren ausschließlich die physisch gespeicherten Daten.

3. Ergebnis-Tabellen bei Abfragen

Das Ergebnis einer DB-Abfrage (Query) ist eine weitere Tabelle, die gespeichert und später weiter bearbeitet werden kann.

4. Views-Relationen für virtuelle Benutzer-Sichten

Die externen Sichten entsprechen virtuellen Tabellen, die sichtbar nur durch relationale Kommandos, nicht aber durch phys. Daten der DB dargestellt werden.

5. Schnappschuß-Relationen

... werden in der DB mit einem Schemaeintrag gespeichert, wobei Datum, Ergänzungszeitpunkt sowie eine Beschreibung enthalten sind.

6. Attribute einer Relation

Jede Spalte einer Tabelle entspricht einem Attribut und besitzt einen Attributnamen.

7. Wertebereich (domain)

Ein Wertebereich eines Attributs ist diejenige Menge, aus der es seine Werte bezieht.

8. Primärschlüssel (primary keys)

Jede Basisrelation hat ein oder mehrere Attribute, deren Werte jeden Tupel dieser Relation eindeutig identifizieren.

9. Fremdschlüssel (foreign keys)

Ein Fremdschlüssel ist jede Spalte (Kombination) in einer Datenbank, die denselben Wertebereich wie der Primärschlüssel einer Basisrelation hat.

Grill

Relationale Datenbanken

Ziele, Methoden, Lösungen

2. verbesserte und erweiterte Auflage

CW-Edition