Lutz Wegner

Datenbanken I

Universität
Kassel

SKRIPTEN DER
PRAKTISCHEN INFORMATIK

Datenbanken I

Lutz Wegner

Datenbanken I

Skriptum zur gleichnamigen Vorlesung an der Universität Kassel

8. Auflage April 2012

8. Auflage April 2012

Skriptum zur gleichnamigen Vorlesung an der Universität Kassel im Sommersemester 2012.

Prof. Dr. Lutz Wegner
Universität Kassel
FB 16 – Elektrotechnik/Informatik
Wilhelmshöher Allee 73
D-34121 Kassel
wegner@db.informatik.uni-kassel.de

Vorwort

Vorwort zur 5. Auflage

Im Sommersemester 2008 habe ich die Vorlesung „Datenbanken I“ nach längerer Pause wieder übernommen und dazu dieses Skript, dessen Ursprünge bis ins Jahr 1991 zurückgehen und das zuletzt 2001 grundsätzlich überarbeitet wurde, neu durchgesehen und in Teilen ersetzt und ergänzt.

Der Kern der Veranstaltung bildet die Behandlung relationaler Datenbankmanagementsysteme (RDBMS). Relationale Datenbanken gehen auf Codd (1970) zurück und sind damit so alt - und so aktuell - wie z. B. Unix/Linux. Die Beschäftigung mit den Grundlagen relationaler Datenbanken gehört zum kanonischen Wissen eines Informatikers. Genauso wichtig ist aber auch der praktische Umgang mit der Materie, etwa in Form von SQL-Übungen auf einem aktuellen RDBMS, z. B. Oracle, DB2, SQL-Server oder MySQL.

Die folgenden Vorgaben und Überlegungen, analog entnommen dem Vorwort zur 2. Auflage, beeinflussen auch heute noch die Stoffauswahl und Schwerpunktbildung:

- die Veranstaltung ist Pflicht im Bachelor-Studiengang Informatik, daneben aber auch im Studienplan für Elektrotechniker und Mathematiker. Sie ist auf 2 Semesterwochenstunden Vorlesung und 2 SWS Übungen festgelegt, womit sich das gesamte Gebiet der Datenbanken in Theorie und Praxis **nicht** abdecken läßt;

- die Schwerpunkte werden deshalb auf die Motivation für Datenbankmanagementsysteme, speziell das Transaktionsprinzip, eine kurze Übersicht der Datenmodelle, auf Grundlagen der Relationentheorie, eine Einführung in die Abfragesprache SQL (Structured Query Language) mit praktischen Übungen und die Normalisierungen gelegt;
- nicht - oder nur kurz - können technische Gesichtspunkte (z.B. Abfrageabarbeitung, Recovery und Parallelverarbeitung) und z.B. verteilte oder objektorientierte Datenbanken behandelt werden; eine Vertiefung dazu, etwa mit einer Veranstaltung „Datenbanken II“, die derzeit allerdings nicht angeboten wird, wäre wünschenswert;
- in den praktischen Übungen soll das Schwergewicht auf dem Umgang mit SQL liegen, wobei jedes System, das mindestens SQL99 unterstützt, ausreicht.

Relationale Datenbanken lassen sich dank der ER-Diagramme und der Tabellendarstellung der Relationen an Beispielen recht anschaulich erklären. An anderen Stellen, etwa bei den Normalformen, sind allerdings präzise Formulierungen gefragt. Ob in diesem Skript die Synthese aus knapper, saubere Darstellung und Anschaulichkeit gelungen ist, müssen die Leser beurteilen. Wer ergänzende Literatur sucht, ist sicher mit dem umfassenden Lehrbuch von Kemper und Eickler (derzeit in der 6. Auflage 2006) fachlich gut beraten [KE].

Dank gilt an dieser Stelle meinen früheren Mitarbeitern Manfred Paul, Sven Thelemann, Jens Thamm und Christian Zirkelbach sowie Kai Schweinsberg, der in diesem Jahr die Übungen betreut. Alle brachten wertvolle Ideen ein, trotzdem gehen alle Fehler zu meinen Lasten. Anregungen, Kritik und Korrekturhinweise sind deshalb immer willkommen.

Kassel, im März 2008

Lutz Wegner

Inhalt

1 Die Aufgaben einer Datenbank

1.1 Programme und Daten

Komplexe Anwendungssysteme werden heute kaum noch von Grund auf neu erstellt, sondern setzen auf vorgefertigten, standardisierten Komponenten auf. Für die Teilaufgabe der Datenhaltung wird man auf ein *Datenbanksystem* (oder kurz eine *Datenbank*) zurückgreifen.

Ein Datenbanksystem besteht aus der Verwaltungssoftware für die Haltung der Daten und den Zugriff darauf, dem sog. *Datenbankmanagementsystem* (DBMS) und den gehaltenen Daten selbst, der *Datenbasis* oder dem *Datenpool*.

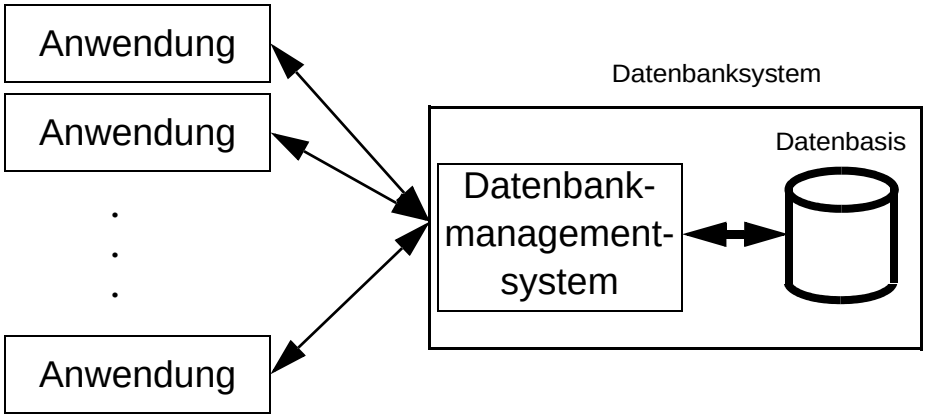


Abb. 1–1 *Schematischer Aufbau eines Datenbanksystems*

Wie man der Abb. 1–1 entnimmt, werden in der Regel mehrere Anwendungsprogramme, die in unterschiedlichen Programmiersprachen geschrieben sein können, auf die Datenbank zugreifen. In einer Mehrprogrammumgebung kann dies auch (quasi-)gleichzeitig geschehen.

Daraus folgt, dass die Daten auf nicht-flüchtigen (persistenten) Datenspeichern, meist Platten, in einem neutralen, programmunabhängigen Format abgelegt werden. Für den Zugriff auf die Daten stellen die Programme Abfragen an die Datenbank, die Teile des meist sehr großen Datenbestands (heute bis in den Bereich mehrerer Terabytes hinein, 1 Terabyte = 2^{40} Bytes) in die Programme einlesen. Dort werden sie in das interne Format gewandelt und verarbeitet.

Geänderte Daten und neu hinzukommende Datensätze werden dann wieder in die Datenbank zurückgeschrieben, andere ggf. dort gelöscht. Geschieht dies in der oben genannten Mehrprozessumgebung, muss zusätzlich auf Erhaltung der *Datenkonsistenz* geachtet werden, d.h. gegenseitig verschränkte Lese- und Schreiboperationen von zwei oder mehr Programmen (genauer *Transaktionen*) auf dem selben Datensatz müssen so ausgeführt werden, dass ihre Auswirkung von einer Hintereinanderausführung nicht zu unterscheiden ist.

Nicht vollständig ausführbare Transaktionen werden rückgängig gemacht, so dass sie keine Auswirkungen hinterlassen. Andererseits werden logisch abgeschlossene Transaktionen, deren Auswirkungen unter Umständen noch nicht auf den Plattenspeicher durchgeschlagen haben, wiederholt, bis sie wieder zu einem konsistenten Datenbankzustand geführt haben. Dies berücksichtigt auch den Fall eines Rechner- oder Datenspeicherausfalls. Wir werden darauf im Zusammenhang mit dem Transaktionsbegriff im Kapitel 6 genauer eingehen.

An dieser kurzen Beschreibung erkennt man bereits, dass die Nutzung eines Datenbanksystems weit über die Möglichkeiten hinausgeht, die eine selbstgeschriebene Dateiverwaltung bietet. Man wird deshalb eine solche enge Bindung von Anwendungsprogramm und eigener Dateiverwaltung nur noch für isolierte Einbenutzerprogramme oder Spezialfälle mit höchsten Leistungsanforderungen an die Zugriffsgeschwindigkeit, etwa im CAD-Bereich, dulden.

Offensichtliche Vorteile der Verwendung einer Datenbank sind demnach:

- geringerer Erstellungs- und Verwaltungsaufwand (jedenfalls auf längere Sicht),
- sichere Realisierung des gleichzeitigen Zugriffs auf Daten durch mehrere Anwender(-programme),
- Änderungen des Formats der Daten bedingen keine Änderungen in allen darauf bezugnehmenden Programmen (Programm-Datenunabhängigkeit),
- Vermeidung von Redundanz der Daten, d. h. zu jedem in der Datenbank gespeicherten Objekt der realen Welt existiert genau ein Satz von Daten,
- Damit verminderte mangelnde Übereinstimmung (Inkonsistenz),
- Datensicherung und Zugriffskontrolle sind nicht für jede Einzeldatei nötig,
- spontane Abfragen abweichend von existierenden Programmen werden ermöglicht.

Von allen oben genannten Gründen ist die *Datenunabhängigkeit* der schwerwiegendste Grund, auf die Verwendung einer Datenbank umzusteigen. Datenunabhängigkeit bezieht sich auf:

- die physische Realisierung (Blockung, dezimale, binäre, gepackte, entpackte Speicherung, Reihenfolge der Felder, Dateinamen, ...),
- Zugriffsmethoden (sequentiell, wahlfrei über binäres Suchen, Baumstruktur, Hash-Tabelle, Indexunterstützung, ...).

Datenbanken sind somit generische Programme zur sicheren, redundanzfreien, physisch unabhängigen Speicherung von Daten. Sie erlauben mehreren Anwendern den gleichzeitigen Zugriff auf die Datenbestände. Jedem Anwender wird dabei eine logische (von den physischen Realisierungen unabhängige) Sicht der Daten geboten.

1.2 ANSI/SPARC Schichtenmodell

Eine Formalisierung eines Konzepts zur Trennung der Sichten erfolgte durch die ANSI/X3/SPARC-Studiengruppe im Jahre 1975, die ein Architekturmodell in (mindestens) 3 Schichten vorschlägt. Der Bericht der Gruppe [AS75] führt den Begriff Schema in seiner heute gebräuchlichen Form ein. Die drei Ebenen lauten:

- externe Modelle
  (externe Schemata, Benutzersichten, external views),
- konzeptuelles Modell
  (konzeptuelles Schema, logische Gesamtsicht),
- internes Modell
  (internes Schema, physische Sicht).

Zwischen den Schichten existieren automatisierte Transformationsregeln. Die Gesamtheit der Schemata werden vom *Datenbankadministrator* (DBA) aufgebaut und überwacht.

Abweichend vom obigen Konzept, das von einer unterschiedlichen Darstellung der Daten auf Extern- und Hauptspeicher ausgeht, gibt es seit längerem auch Ansätze, die Unterscheidung aufzuheben und mit einem einstufigen, persistenten, virtuellen Adressraum zu arbeiten.

Anwendung und Datenbank verschmelzen mittels Programmierung in einer persistenten Programmiersprache, die meist auf objekt-orientierten Programmiersprachen wie C++ aufsetzen. Im Zusammenhang mit objekt-orientierten Datenbanksystemen im Kapitel 8 gehen wir darauf nochmals ein.

Terabytes? Terrorbytes!

Bereits für einen geringen Geldbetrag kann man sich eine Platte mit mehreren hundert Gigabyte Kapazität, also eine Platte mit mehr als $100 * 2^{30}$ Bytes, d. h. mit Platz für mehr als hundert Milliarden Zeichen, kaufen und in einen PC einbauen. Nach einer Einschätzung von Jim Gray [Gray95], werden parallele Systeme, aufgebaut aus PC-ähnlichen Boards (Cyberbacksteinen) mit Plattenfarmen die Zukunft beherrschen. Warum die Ansprüche unbegrenzt sind, kann man der folgenden Tabelle entnehmen.

Bereits jetzt sind Tabellen (einzelne Datenbankrelationen) in Terabytegröße in der Praxis vorhanden, z. B. die der Einzelhandelskette Wal-Mart mit 1,8 TB, d. h. 4 Milliarden Tupeln.

Ebenfalls im Gange ist die Digitalisierung der Buchbestände, z. B. für die Bayrische Staatsbibliothek in München, und zwar einmal als reiner ASCII-Text und zum anderen als Bitmap-Vollbild. Für die größte Bibliothek der Welt, die Library of Congress, schätzt man den Umfang in ASCII auf ungefähr 25 TB, was schon heute im Bereich des technisch Möglichen liegt.

Größe	Bezeich.	≈ Zehnerpot	Sprechw.	Beispiel
2^{10}	KB Kilo	10^3	Tausend	1/2 Schreibmaschinenseite ASCII
2^{20}	MB Mega	10^6	Million	1 Buch, 1 Floppy
2^{30}	GB Giga	10^9	Milliarde	Film in TV-Qualität, 10 m Bücher
2^{40}	TB Tera	10^{12}	Billion (dt.)	alle Röntgenaufn. eines Krankenhauses, kleinere Bibliothek

Größe	Bezeich.	≈ Zehnerpot	Sprechw.	Beispiel
2^{50}	PB Peta	10^{15}	Billiarde	3 Jahre EOS Landsat Daten was ein/e 60-jährige/r bisher gesehen hat bei 1 MB/s
2^{60}	EB Exa	10^{18}	Trillion	5 EB = alle jemals gesprochenen Worte
2^{70}	ZB Zetta	10^{21}	Trilli-arde	
2^{80}	YB Yotta	10^{24}	Quattril-lion	

Zugleich werden an diesen Größenordnungen auch die Grenzen der sog. *persistenten Adreßräume* (vgl. Diskussion zu Pointer Swizzling unten) sichtbar; die Seitentabelle für einen Adreßraum der Größe 2^{48} verlangt bei 4KB Seiten 64 Milliarden Einträge zu ca. 6 Bytes, d.h. die Seitentabelle selbst umfaßt bereits 360 GB !

Übung 1–1

Diskutieren Sie die Speicherung von flüchtigen Werten in einem laufenden Programm von der Speicherung persistenter Daten in einer Datei oder einer Datenbank. Welche Rolle spielt die Adressierungsart, warum spricht man von Orts- und Inhaltsadressierung?

2 Das Entity-Relationship Modell

2.1 Entitäten und Beziehungen

Ein *Datenmodell* stellt die sog. *Miniwelt*, d. h. den relevanten und vereinfachten Ausschnitt der zu behandelnden Realität, implementierungsneutral dar. Die so gewonnenen konzeptuellen Schemata sind übersichtlicher und führen zu besseren Gesamtsichten als Entwürfe, die sich bereits an den Möglichkeiten eines konkreten DBMS orientieren.

Ein solches abstraktes Datenmodell ist das *Entity-Relationship-Modell* (ER-Modell). In ihm werden Objekte der realen Welt als sog. Entitäten (entities) angesehen, etwa ein Mitarbeiter „Peter Müller“ oder das Projekt „Internet-Auftritt“, zwischen denen Beziehungen (relationships) existieren, z.B. die Beziehung „arbeitet-an“.

Entitäten mit gleichen Merkmalskategorien werden in einer Datenbank zusammengefasst. Sie bilden sog. *Entity-Klassen* (auch *Entity-Mengen* oder *Entity-Typen* genannt), z. B. Mitarbeiter eines Unternehmens (Typ MITARBEITER), Projekte (Typ PROJEKT) und Abteilungen (Typ ABTEILUNG).

Merkmale einer Entity-Klasse werden als *Attribute* bezeichnet, also z.B. für Mitarbeiter die Attribute Mitarbeiter-Identifikator (MID, numerisch), Vor- und Nachname (NAME, Zeichenkette) und Geburtsjahr (GJAHR, numerisch).

Demnach könnte ein spezieller Mitarbeiter (ein Entity aus der Entitymenge MITARBEITER) durch die Attribute [MID: 3932, NAME: Peter Müller, GJAHR: 1947] vollständig beschrieben sein.

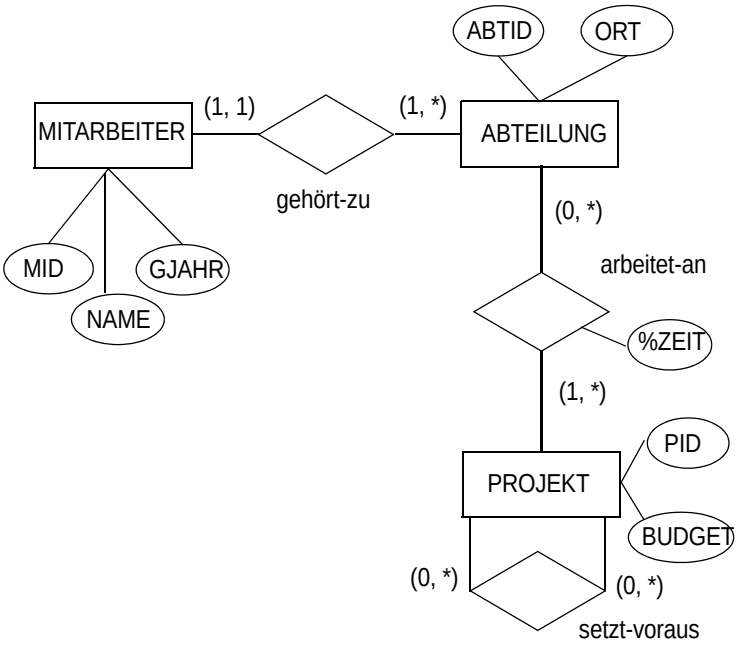


Abb. 2–1 *ER-Diagramm für Mitarbeiter-Abteilung-Projekt-Beispiel*

Attribute werden durch Festlegung des *Wertebereichs* (engl. domain), also die Nennung des Attribut-Typs (hier z. B. numerisch), und der Attributbezeichner (hier MID, NAME, ...) bestimmt. Existiert für ein Attribut gegebenenfalls kein Wert, kann stattdessen ein sog. Nullwert gespeichert werden (Details siehe unten).

Zwischen den Entitäten herrschen Beziehungen (engl. relationships), z. B. zwischen MITARBEITER und PROJEKT die Beziehung ARBEITET_AN, bzw. die Umkehrung WIRD_BEARBEITET_VON. Dabei können zwischen zwei Entity-Typen auch mehrere Beziehungstypen existieren, z. B. zwischen Mitarbeitern und Projekten die Beziehungen ARBEITET_AN und LEITET. Man spricht dann von mehreren *Rollen*, die ein Entity übernimmt.

Beziehungen können auch wieder Attribute zur Beschreibung allgemeiner Eigenschaften haben, z. B. könnte ARBEITET_AN zwischen MITARBEITER und PROJEKT mit PROZENT_ARBEITSZEIT, die Beziehung GEHÖRT_ZU zwischen MITARBEITER und ABTEILUNG mit dem Attribut SEIT (vom Typ Datum) versehen sein.

Das ER-Modell hat eine anschauliche graphische Darstellung: Im einfachsten Fall werden Entity-Mengen durch Rechtecke, Attribute durch Ellipsen, Beziehungen durch Rauten dargestellt, die durch Linien verbunden sind. Sind, wie in Abb. 2–1, zwei Entity-Mengen an einer Beziehung beteiligt, spricht man von binären Beziehungen (Grad 2, die weitaus gebräuchlichste Form). Daneben sind Beziehungen vom Grad 1, wie z. B. „setzt-voraus“ in Abb. 2–1, und vom Grad 3 üblich.

An den Kanten können, wie in Abb. 2–1, Teilnahmebedingungen und Kardinalitäten eines Beziehungstyps angegeben werden. Unter Teilnahmebedingungen (auch Mitgliedschaften genannt) versteht man die Forderung, dass jedes Entity an der Beziehung teilnehmen muss, d.h. dass die Beziehung total sein muss, bzw. die Erlaubnis, dass sie partiell sein darf (optionale Mitgliedschaft).

So besagt Abb. 2–1 oben, dass nicht jede Abteilung ein Projekt, ggf. aber eine Abteilung viele Projekte bearbeitet und dass jedes Projekt von mindestens einer Abteilung, ggf. vielen, bearbeitet wird. Die Mitgliedschaft von ABTEILUNG an der Beziehung ist optional, PROJEKT dagegen nimmt zwangsweise an der Beziehung teil.

Man nennt dies dann auch eine existentielle Abhängigkeit, da ein Projekt nicht ohne eine „bearbeitet-Beziehung“ existieren kann (entfernt man die Abteilung, sterben auch alle von ihr bearbeiteten Projekte).

Der eigentliche Zahlenwert, der angibt, an wievielen Beziehungen $r \in R$ eine Entität $e \in E$ teilnehmen kann, heißt Kardinalität und sollte mit Minimal- und Maximalwert angegeben werden, wobei * für „beliebig viele“ steht (vgl. Abb. 2–1).

1:1-, 1:n-, n:m-Beziehungen

Am gebräuchlichsten ist bei Zweierbeziehungen jedoch die recht ungenaue Angabe des Verhältnisses der Kardinalitäten, d. h. man klassifiziert sie als 1:1-, 1:*n*-, *n*:1-, *n*:*m*-Beziehungen, und zwar als

1:1 wenn die Kardinalitäten links und rechts (0, 1) oder (1, 1)

1:*n* ~ rechts (0, 1) oder (1, 1) und links (0, *) oder (1, *)

n:1 ~ rechts (0, *) oder (1, *) und links (0, 1) oder (1, 1)

n:*m* ~ links und rechts (0, *) oder (1, *) sind.

GEHÖRT_ZU zwischen MITARBEITER und ABTEILUNG ist demnach eine *n*:1-Beziehung (viele Mitarbeiter in einer Abteilung). In der englischsprachigen Literatur ist dann von one-to-one, one-to-many, many-to-many relationships die Rede.

Mitgliedsklassen und Kardinalitäten haben großen Einfluss auf den Systementwurf. Dazu gehört die Befragung der „Miniwelt-Experten“ zur Ermittlung der Entity-Typen, Beziehungen und Restriktionen (z. B. Kardinalitäten), die als Systemanalyse bezeichnet wird. Auf der Basis des (erweiterten) ER-Modells, das auf Chen [Chen76] zurückgeht, wird der Entwurf des konzeptuellen Schemas heute von vielen DB-Werkzeugen (z. B. ERwin, S-Designer, SELECT-SE) mit graphischen Editoren und automatisierten Übersetzungen in das relationale Modell unterstützt. Er bestimmt, wo Nullwerte erlaubt und wie Tabellen aufzuteilen sind (die sog. Normalisierung).

2.2 Weitere Beispiele

Wie oben besprochen wären demnach Personen, Lieferanten, Kunden, Artikel, Angestellte, Projekte, Schulklassen, Lehrer, Schüler, usw. Beispiele für Entities. Sie bilden die *Entity-Klassen* (auch *Entity-Mengen* oder *Entity-Typen* genannt)

- Kunden eines Unternehmens (Typ KUNDE)
- Lieferanten eines Unternehmens (Typ LIEFERANT)

- Angestellte eines Unternehmens (Typ ANG)

Die eine Entity-Klasse auszeichnenden Merkmale werden als *Attribute* bezeichnet, also z. B.

- Kunden (KUNDE) durch die Attribute Kundennummer, Kundenname, Ort mit den Attributtypen Integer und zweimal Text
- Angestellter (ANG) durch die Attribute Personalnummer (numerisch), Name (Zeichenkette), Alter und Gehalt (numerisch)

Übung 2–1

Welche anderen Angaben könnten von Interesse sein?

Einzelne Entities (Entity-Ausprägungen, Entity-Instanzen) werden identifiziert durch ihre Attributwerte, z. B.

- ein spezieller Kunde durch das Wertetupel (4711, Müller, Altenhausen)
- ein Angestellter durch (101202, Müller, 52, 70000)

Existiert für ein Attribut gegebenenfalls kein Wert, z.B. wenn ein Angestellter kein festes Gehalt hat, sondern auf Provisionsbasis arbeitet, kann stattdessen ein sog. Nullwert gespeichert werden.

Übung 2–2

Die Identifikation von Entities orientiert sich an den gespeicherten Werten, d. h. zwei Entities lassen sich in der Datenbank unterscheiden, wenn sie mindestens einen unterschiedlichen Wert besitzen. Welche andere Identifikationsmethode könnte man sich vorstellen?

Übung 2–3

Man diskutiere die Modellbildung mit den Begriffen der Objektorientierung (z. B. Vererbung, Objektklassen, Objektinstanzen, Intension, Extension)!

Zwischen den Entities herrschen Beziehungen (engl. relationships), z. B. zwischen Lehrer und Klasse die Beziehung „unterrichtet-in“, bzw. die Umkehrung „wird-unterrichtet-von“. Abbildung 2–2 zeigt eine konkrete Ausprägung.

Andere Entity-Typen und ihre Beziehungen (genauer: Beziehungstypen) wären z. B.:

- Angestellter ↔ Projekt: „arbeitet-an“
- Kunde ↔ Artikel: „hat-bestellt“
- Schüler ↔ Klasse: „ist-in“

Übung 2–4

Diskutieren Sie Abbildung 2–2 ! Wie wäre der Fall eines Lehrers zu behandeln, der zwei Fächer in der selben Klasse unterrichtet?

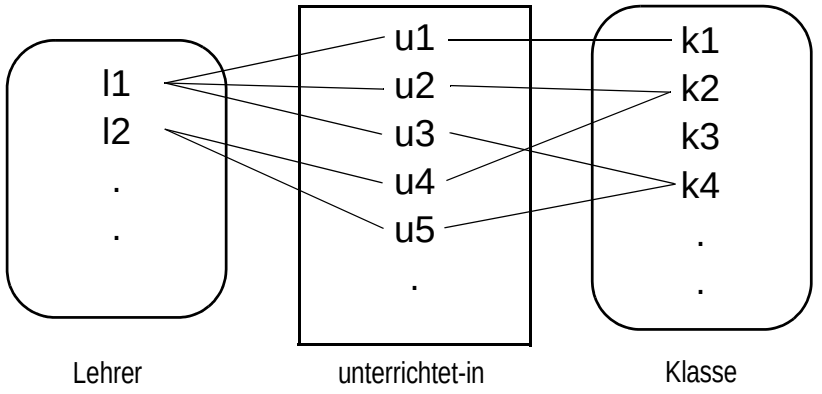


Abb. 2–2 *Ausprägungen einer Beziehung*

Im allgemeinen interessieren im konzeptuellen Schema nur die Entity- und Beziehungstypen. Ihre Darstellung erfolgt im sog. *ER-Diagramm* (nach Chen [Chen76], dem Hauptbegründer des Entity-Relationship-Modells) durch

- Vierecke für Entity-Typen
- Rauten für Beziehungs-Typen
- Elipsen für Eigenschaften der Entities oder Beziehungen, sofern überhaupt dargestellt
- Kanten für die Verbindung von Entities, Beziehungen und Eigenschaften

Die Anzahl der an einer Beziehung beteiligten Entity-Typen heißt *Grad* der Beziehung, üblich sind 1, 2 oder (manchmal problematisch) 3 – vgl. Abb. 2–3 unten.

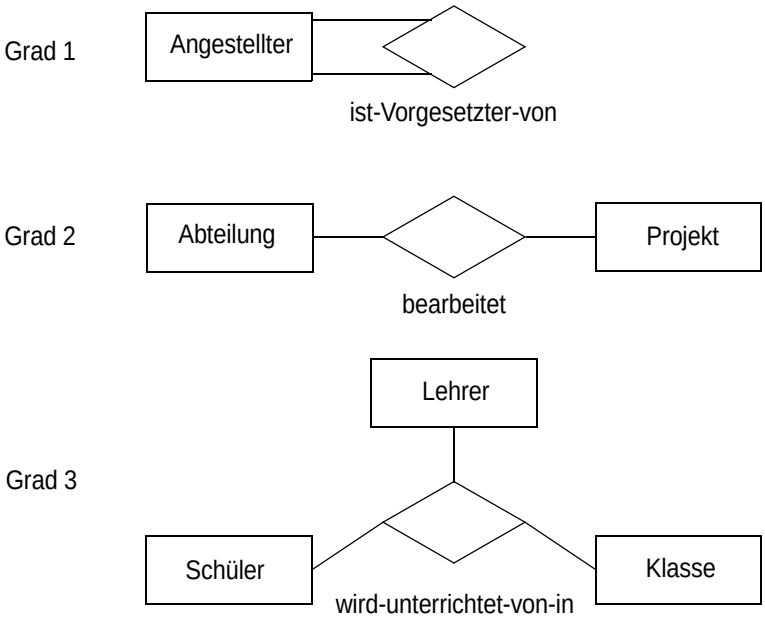


Abb. 2–3 *Grad eines Beziehungstyps*

Übung 2–5

Geben Sie eine konkrete Ausprägung - analog zu Abb. 2–2 - zu der ternären Beziehung „wird-unterrichtet-von-in“ zwischen Schüler, Lehrer und Klasse.

Übung 2–6

Zeigen Sie, daß die untenstehenden drei binären Beziehungen in Abb. 2–4 nicht notwendigerweise äquivalent sind zu der ternären Beziehung aus Abb. 2–3 oben!

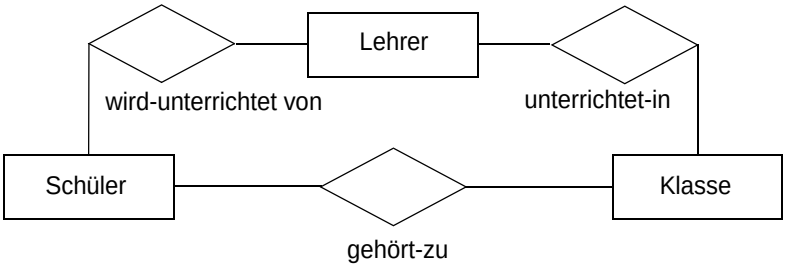


Abb. 2–4 *Drei binäre Beziehungen als Ersatz für eine ternäre Beziehung*

Zwischen zwei Entity-Typen können auch mehrere Beziehungstypen existieren, z. B. Angestellter und Projekt mit Beziehungen „arbeitet-an“ und „leitet“. Man spricht dann von mehreren Rollen, die ein Entity Angestellter übernimmt.

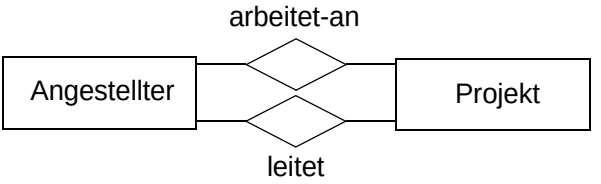


Abb. 2–5 *Beziehung mit mehreren Rollen*

Beziehungen können auch wieder Attribute (allgemeiner Eigenschaften) haben, z. B. „arbeitet-an“ zwischen Angestellter und Projekt könnte das Attribut „Prozent-Arbeitszeit“ haben, die Beziehung „unterrichtet-in“

zwischen Lehrer und Klasse die Attribute „Fach“ und „Zeit“ (vom Typ Datum). Attribute an den Beziehungen werden auch dargestellt durch Kanten mit einem Kreis um den Attributnamen.

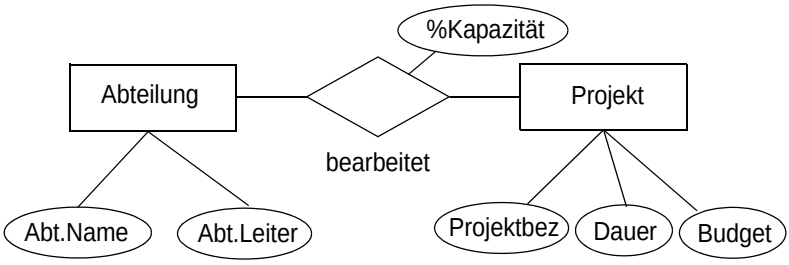


Abb. 2–6 *Attribute an Entities und Beziehungen*

Entwurfsproblem des ER-Modells: keine scharfe Abgrenzung zwischen Entities und Beziehungen und unterschiedliche Sprechweisen und Notationen.

Übung 2–7

Man ergänze das ER-Diagramm für das Schulbeispiel mit den Entities Lehrer, Klasse, Schüler um die Beziehungen „ist-Klassenlehrer-von“, „ist-Klassensprecher-von“.

Neben dem Grad betrachtet man noch *Teilnahmebedingungen* und *Kardinalitäten* eines Beziehungstyps. Unter Teilnahmebedingungen (auch *Mitgliedschaft* genannt) versteht man Auflagen, die fordern, daß eine Beziehung total sein muß, bzw. erlauben, daß sie partiell sein darf (*optionale Mitgliedschaft*).

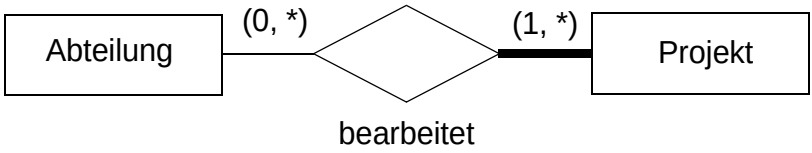


Abb. 2–7 *Mitgliedschaften zwischen Abteilung und Projekt*

So besagt die Abb. 2–7 oben, daß nicht jede Abteilung ein Projekt, ggf. aber eine Abteilung viele Projekte bearbeitet und daß jedes Projekt von mindestens einer Abteilung, ggf. vielen, bearbeitet werden muß. Die Mitgliedschaft von Abteilung ist damit optional, Projekt dagegen nimmt zwangsweise an der Beziehung teil.

Man nennt dies dann auch eine *existentielle Abhängigkeit*, da ein Projekt nicht ohne eine „bearbeitet-Beziehung“ existieren kann. Häufig wird in ER-Diagrammen die totale Beziehung auch durch eine Doppelkante betont, daneben sind auch Darstellungen mit einem „Kringel“ an der Raute oder dem Entity für die Andeutung der *optionalen Mitgliedschaft* gebräuchlich. Diese Darstellung wurde für Abb. 2–8 gewählt, um auszudrücken, daß nicht jeder Angestellte einen Rechner am Arbeitsplatz hat und manche Rechner keinem Besitzer zugeordnet sind.

Abb. 2–8 *Alternative Darstellung für optionale Teilnahme*

Der eigentliche Zahlenwert, der angibt, an wievielen Beziehungen $r \in R$ ein Entity $e \in E$ teilnehmen kann, heißt *Kardinalität* und sollte mit Minimal- und Maximalwert angegeben werden, wobei * für „beliebig viele“ steht (vgl. Abb. 2–7 und Abb. 2–9).

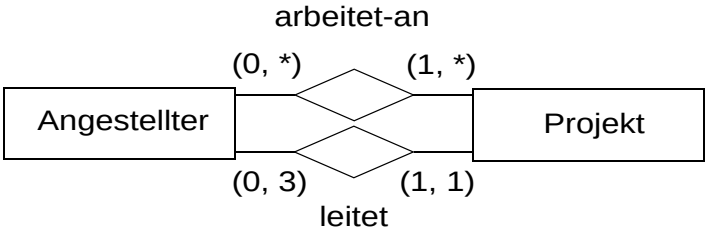


Abb. 2–9 *Beziehungen mit Kardinalitätsangaben*

In der Abb. 2–9 gilt also: ein Angestellter muß nicht an einem Projekt arbeiten und kann an vielen Projekten mitarbeiten, in jedem Projekt arbeitet mindestens ein Angestellter. Jedes Projekt wird von genau einem Angestellten geleitet, ein Angestellter leitet höchstens drei Projekte.

Übung 2–8

Welche der Kanten in Abb. 2–9 könnte man doppelt ziehen als zusätzliche Andeutung der existentiellen Abhängigkeit?

Am gebräuchlichsten ist bei Zweierbeziehungen jedoch die recht ungenaue Angabe des Verhältnisses der Kardinalitäten, d.h. man klassifiziert sie als 1:1-, 1:*n*-, *n*:1-, *n*:*m*-Beziehungen. Man beachte, daß (1,*)-Beziehung-(0,1) zu 1:*n* und nicht zu *n*:1 wird!

Beispiel 2–1 Rollen

Wenn man in Abb. 2–9 fordert, dass jeder Angestellte höchstens ein Projekt leiten darf, so gilt: die Beziehung „leitet“ zwischen Angestellter und Projekt ist 1:1. Die Beziehung „arbeitet-an“ zwischen Angestellten und Projekt ist dagegen *n*:1 (viele Angestellte arbeiten an einem Projekt).

Übung 2–9

Man ergänze das Beispiel mit Lehrern, Klassen, Schülern und den Beziehungen „unterrichtet-in“, „ist-Klassenlehrer“, „ist-in“, „ist-Klassensprecher“ um Kardinalitäten.

Beispiel 2–2 Kunden-Artikel-Lieferanten

Zwischen Lieferanten und Artikeln besteht eine (1:*n*)-Beziehung, d.h. ein Lieferant liefert viele Artikel, aber jeder Artikel wird von genau einem Lieferanten geliefert. Kunden können Bestellungen offen haben, müssen aber nicht. Jede Bestellung muß sich aber genau einem Kunden zuordnen lassen. In Bestellungen taucht mindestens ein Artikel auf mit einer

Bestellmenge. Die Beziehungen sind in Abb. 2–10 dargestellt und werden im folgenden wiederholt aufgegriffen!

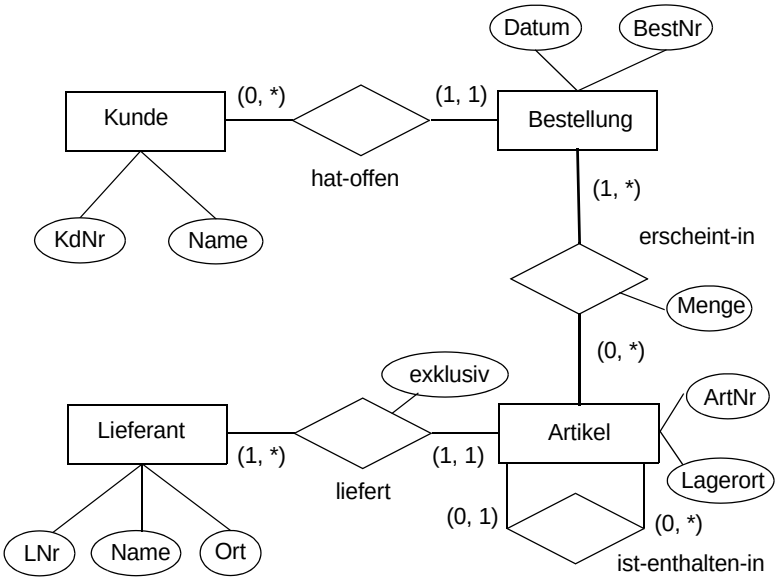


Abb. 2–10 *ER-Diagramm für das Bestellungs-Beispiel*

Übung 2–10

Welche Beziehungen in Abb. 2–10 sind existentiell?

Manche Beziehungen sind spezieller Art, z. B. unterscheidet man

- *Spezialisierungen* (Untermengen, *is-a Beziehungen,* Top-down-Sichtweise)
- *Generalisierungen* (Obermengenbildung, Verallgemeinerung, Bottom-up-Sichtweise)
- die bereits bekannten *existentiellen Beziehungen* (meist 1:*n*).

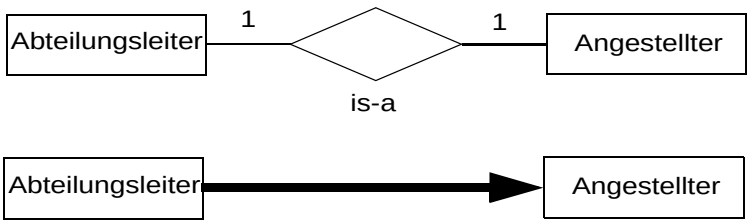


Abb. 2–11 *Teilmengenbeziehung „is-a“ (Untertyp, Spezialisierung)*

In Abb. 2–11 ist jeder Abteilungsleiter auch ein (is-a) Angestellter. Abteilungsleiter kann als Untertyp von Angestellter bezeichnet werden und ist ggf. durch besondere Attribute ausgezeichnet (z.B. „Prokura seit“ für Abteilungsleiter). In Abb. 2–11 ist dies nochmals hervorgehoben, wobei die Spezialisierung alternativ auch durch einen Pfeil angedeutet wird. Da Spezialisierungen auch in Zusammenhang mit der Typhierarchie der Objektorientierung auftaucht und dort gerne als Vererbungsbaum dargestellt wird, beachte man die Richtung des Pfeils - von der Spezialisierung (*Sub-Typ*) zum übergeordneten *Super-Typ*!

Die umgekehrte Sichtweise ist die *Generalisierung*, bei der Entitytypen zu einem Super-Typ verallgemeinert werden. In Abb. 2–12 kann man Bücher und Artikel zu Publikationen zusammenfassen.

Spezialisierungen und Generalisierungen können

- *total* sein, d. h. es existiert kein Entity im Super-Typ zu dem es nicht ein Entity in einem der Sub-Typen gibt; das Gegenteil wäre eine *partielle* Beziehung
- *disjunkt* sein, d. h. die Sub-Typen haben keine gemeinsamen Elemente; das Gegenteil wäre *nicht-disjunkt*.

Hierarchisch-existentielle Beziehungen (H-Beziehungen) müssen auf einer Seite eine Kardinalität 1 besitzen, d.h. *E* steht zu *E'* in einer 1:*n* Beziehung und zu jedem speziellen Entity *e'* in *E'* muß genau ein *e* in *E* existieren. Daher sagt man auch *E'* ist *E* hierarchisch untergeordnet *oder e'* hängt existentiell von *e* ab.

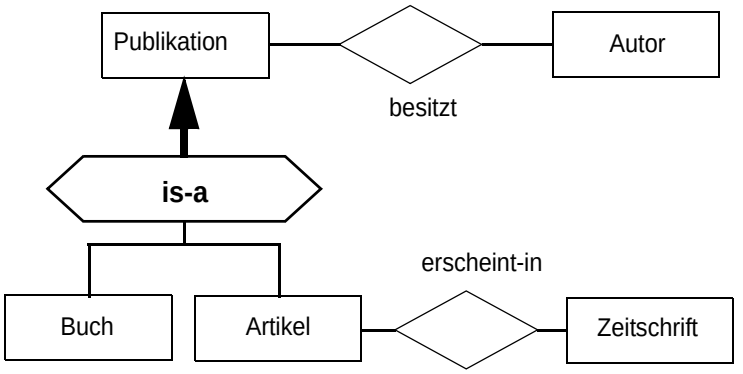


Abb. 2–12 *Darstellung der Generalisierung*

Beispiel 2–3 Abteilungen und Angestellte

Man betrachte in Abb. 2–13 die „arbeitet-in“-Beziehung zwischen Abteilung und Angestellten. Zu jedem Angestellten darf höchstens eine Abteilung existieren, in der er arbeitet. In der Darstellung als Relation weiter unten erscheint Attribut AN in Relation ANGESTELLTE als sog. *Fremdschlüssel*, d. h. der Wertebereich von AN ist die Menge genau der Werte, die in ABTEILUNG ein Abteilungstupel identifiziert (genaueres später).

Damit ist ein Angestellter abhängig von der angegebenen Abteilung:

- wird die Abteilung gestrichen, sind die abhängigen Angestelltentupel zu streichen oder ein neuer AN-Wert muß eingetragen werden;
- für AN in ANGESTELLTE sind Nullwerte nicht erlaubt.

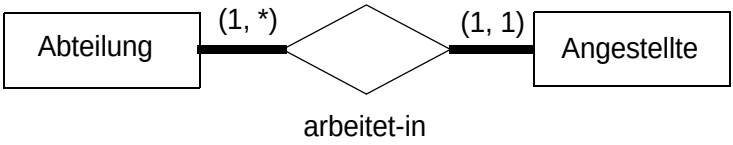


Abb. 2–13 *Eine hierarchisch-existentielle Beziehung*

ABTEILUNG

AbtNr	AbtName
1	FuE
2	Marketing

ANGESTELLTE

PersNr	Name	AN
101	Schmidt	1
102	Müller	2
103	Huber	1
201	Meier	2
202	Wagner	2

Übung 2–11

Welche hierarchischen Beziehungen gibt es im Lehrer-Schüler-Klasse Beispiel? Welche Untertyps-Beziehungen, welche Generalisierungen?

Übung 2–12

Abteilungen ohne Angestellte gebe es nicht, wohl aber Abteilungen ohne Projekte (z. B. die Personalabteilung). Projekte hängen existentiell-hierarchisch von Abteilungen ab. Geben Sie ER-Diagramm und Tabellen an! Taucht in der Projekttabelle jede Abteilung auf?

Übung 2–13

Läßt man auch mengenwertige Attribute zu, können die Angestellten einer Abteilung bei den Abteilungen, z. B. als *Menge von Personalnummern* (PersNr), gespeichert werden. Geben Sie die neu gestaltete Tabelle ABTEILUNG an! Welche Vor- und Nachteile hat diese Form der Darstellung einer hierarischen Beziehung?

3 Das relationale Modell

3.1 Relationen

Ausgangspunkt für das *Relationenmodell* ist die Arbeit von E. F. Codd [Codd70]. In diesem Modell werden

- Entity-Mengen und ihre Beziehungen als *Relationen* behandelt,
- Relationen durch *Tabellen* dargestellt.

Die formalen Grundlagen stammen aus der Mengentheorie:

> eine *n-stellige Relation* ist eine Teilmenge des *Kreuzprodukts* (*kartesischen Produkts*) von n beliebigen *Wertebereichen* (engl.: *domains*) $W_1, W_2, ..., W_n$,
> kurz $W_1 \times W_2 \times ... \times W_n$ Menge aller n-Tupel $(w_1, w_2, ..., w_n)$
> mit Komponente $w_i \in W_i$ für $i = 1, 2, ..., n$.

Der Wert n bezeichnet den *Grad* (engl.: *degree*, *arity*) der Relation.

Beispiel 3–1

Es gelte Grad $n = 2$, $W_1 = \{4711, 4712\}$, $W_2 = \{pi, pa, po\}$. Damit ist $W_1 \times W_2 = \{(4711, pi), (4711, pa), (4711, po), (4712, pi), (4712, pa), (4712, po)\}$.
Eine Relation BSP $\subseteq W_1 \times W_2$ wäre etwa {(4711, pi), (4711, po), (4712, pa)} mit 3 Tupeln (Kardinalität 3).

BSP

NA	TA
4711	pi
4711	po
4712	pa

In Tabellenform erscheinen die *Attribute* als Spalten mit Namen (hier NA und TA) und ggf. Typangabe. Der *Name der Relation* im Beispiel sei BSP, seine Angabe erfolgt meist zusammen mit dem *Relationenschema* (Menge der Attribute) in der Form

- BSP(NA, TA),

oder in der Form

- BSP(NA: Integer; TA: string).

Ein praxisnäheres Beispiel wäre die folgende Mitarbeitertabelle (Grad 4, 4-stellige Relation).

MITARBEITER

NAME	MID	GJAHR	ABTID
Peter	1022	1959	FE
Gabi	1017	1963	HR
Beate	1305	1978	VB
Rolf	1298	1983	HR
Uwe	1172	1978	FE

Ein spezielles 4-Tupel wäre [*Gabi, 1017, 1963, HR*].

Für die *Wertebereiche der Attribute* gilt (vgl. Typdefinitionen in Programmiersprachen):

- im „normalen“ Relationenmodell sind nur *atomare Werte* zugelassen (sog. 1. *Normalform*, 1NF),

- ggf. ist der *Wertebereic*h mit benutzerdefinierten oder generischen Typnamen versehen, z.B. BESTELLDATUM: Datum
- ggf. sind *Spezialisierungen* (Teilbereich a..b, Aufzählung) möglich, oder
- spezielle Attributtypen sind über mehrere Relationen hinweg verwendbar, z.B. Typ ARTIKELNUMMER für ARTNR in BESTELLUNG und für ANR in ANGEBOT als Untertyp von Integer mit 7 Stellen und Prüfziffer,
- folgt aus der Gleichheit eines Attributnamens über mehrere Tabellen hinweg die Gleichheit des Typs.

Wie bereits besprochen, ist bei der *Tabellendarstellung* zu beachten:

- die Zeilen sind paarweise verschieden,
- die Reihenfolge der Zeilen ist beliebig,
- die Spaltenanordnung ist beliebig.

Allerdings ist durch die Abspeicherung der Tupel (Zeilen) als Sätze einer Datei und der Attributwerte als Felder dieser Sätze de facto immer eine Reihung festgelegt, so daß wiederholte Sitzungen in der Regel die gleiche Darstellung liefern.

3.2 Nullwerte

Es kann vorkommen, daß man für einen Attributwert keine Angaben machen kann, oder daß eine Angabe in der Spalte keinen Sinn macht. Beispiele für Auftreten eines solchen Werts in der Praxis wären [Date82]:

- Geburtsdatum unbekannt
- Sprecher wird noch bekanntgegeben
- momentaner Aufenthaltsort unbekannt
- nicht zutreffend (Heiratsdatum bei Familienstand = ledig)
- errechnetes Durchschnittsgehalt der Angestellten einer Abteilung wenn Abteilung z. Zt. keine Angestellten hat (besser Wert = 0 ?).

Deshalb sieht man für jeden Wertebereich einen Wert „unbekannt“ (engl.: *null value*, hier auch DB-Null) vor, dessen Ausgabedarstellung ’?’ oder „leer“ ist und dessen Angabe durch die symbolische DB-Nullkonstante NULL erfolgt.

Die Operationen der relationalen Algebra werden dann erweitert um eine Semantik für Nullwerte. Darin kann man fordern (vgl. [Date82]), daß alle Vergleiche mit mindestens einem DB-Null Operanden DB-Null ergeben. Speziell ergibt der Vergleich NULL = NULL wieder NULL und nicht wahr, was wenig intuitiv und Quelle der meisten Irrtümer ist! In der Praxis liefern die meisten SQL-Implementierungen deshalb bei Vergleichen mit einem DB-Nullwert immer falsch ab, außer bei dem speziellen Test ISNULL, wenn tatsächlich ein Nullwert vorliegt.

Nullwerte können daneben auch für Attribute ausgeschlossen werden, z.B. könnte ARTIKELNUMMER = NULL für einen Artikel verboten, für BEZEICHNUNG aber zugelassen sein.

Läßt man sich auf eine Nullwertsemantik ein, kann man für die dreiwertige Logik mit wahr (T, true), falsch (F, false) und vielleicht wahr (M, maybe) die Wahrheitstafeln aus Abb. 3–1 aufstellen.

UND	T	M	F
T	T	M	F
M	M	M	F
F	F	F	F

ODER	T	M	F
T	T	T	T
M	T	M	M
F	T	M	F

NICHT	
T	F
M	M
F	T

Abb. 3–1 *Wahrheitstafeln der Nullwertsemantik*

Übung 3–1

Man betrachte das Anfügen einer neuen Spalte und das Einfügen einer neuen Zeile. Welche Rolle spielen Nullwerte?

Übung 3–2

Diskutieren Sie die Wahrheitstafeln mit den Aussagen

Garantiefall = (Alter < 6 Monate) und (Garantiekarte vorhanden)

Eignung = (Diplom in ...) oder (Berufserfahrung > 3 Jahre)

Codd schlägt in [Codd87] weitere Unterscheidung bei Nullwerten vor:

- eine A-Markierung für „anwendbare“ (missing and applicable),
- eine I-Markierung für „nichtanwendbare“ (zu ignorierende, inkonsistente, missing and inapplicable) Nullwerte.

Beispiel 3–2 Name des Ehegatten

- ?A wenn Familienstand = verheiratet (V), Name aber unbekannt
- ?I wenn Familienstand = nicht verheiratet/geschieden/verwitwet (S).

Wie die folgende Tabelle [Yue91] zeigt, ist dann aber auch eine U-Markierung (unbekannt ob anwendbar, unknown) möglich.

FAMILIEN

NAME	FAMILIENSTAND	EHEPARTNER
Anna	V	Bernd
Claus	S	?I
Doris	V	?A
Erich	?A	?U

Nullwerte im Zusammenhang mit Mengen ergeben weitere Probleme z.B.

- Sei die Menge der Ehepartner gegeben durch {Hans, Anna, ?}
  Kann man aus der Mengeneigenschaft schließen, daß ? ungleich ’Hans’ und ungleich ’Anna’ ist?

- Ist ? ein ?I dann wäre Martin ∈{Hans, Anna, ?} falsch, ist ? ein ?A, dann liefert das Prädikat ?A.
- Was ist die Kardinalität von {Hans, Anna, ?}?

Übung 3–3

Diskutieren Sie die drei Arten von Null im Zusammenhang mit einer ANGESTELLTEN-Tabelle. Geschäftsführer/innen sind in keiner Abteilung, Lehrlinge sind noch keiner zugeteilt .

Hardware und Implementierungssprachen bieten keine Unterstützung für Nullwerte, anders etwa als im IEEE Gleitkommastandard mit ± Unendlich und NAN (not a number) für ungültige Werte (signalisierend und nicht-signalisierend, d. h. mit und ohne Unterbrechungsauslösung).

Sofern die Nullwertproblematik überhaupt angegangen wird, lassen sich für heutige SQL-Systeme die folgenden Regeln angeben.

- Es existiert nur ein gemeinsamer Nullwert sowohl für „unbekannt“ als auch für „trifft nicht zu“; also sogenannte 3VL (*three valued logic*).
- Vergleiche mit DB-Nullwerten liefern falsch! Damit liefert in der Tabelle oben

  ```
  SELECT NAME, EHEPARTNER
  FROM FAMILIEN
  WHERE EHEPARTNER <> GABI
  ```

  nur die 1. Zeile mit (Anna, Bernd), obwohl klar ist, daß Claus mit Familienstand S nicht einen Ehepartner mit Namen „Gabi“ haben kann.
- Die Suchbedingung
  `EHEPARTNER = NULL`
  ist in SQL unzulässig. Stattdessen gibt es ein Prädikat `IS NULL` das gdw. wahr liefert, wenn der Wert DB-Null ist, falsch sonst.

- Bei Duplikatseliminierung gelten zwei Tupel $v = (v_1, v_2, ..., v_n)$ und $w = (w_1, w_2, ..., w_n)$ als gleich gdw. für alle i $(1 \leq i \leq n)$ v_i und w_i beide DB-Null sind oder v_i und w_i beide nicht DB-Null und $v_i = w_i$.
- Die Funktionen SUM (Summe) und AVG (average, Durchschnitt) ignorieren DB-Nullwerte, auch COUNT(X) (Zählen, Abzählen) für Attribut(e) X, nicht dagegen COUNT(*) (Zählen aller Tupel).

A	B
?	10
20	?
?	?

  Damit liefert etwa AVG(B) den Wert $x = 10$, SUM(B) den Wert $y = 10$, COUNT(B) den Wert 1, COUNT(*) den Wert z = 3, wodurch x ≠ y/z.

- Alle elementaren arithmetischen Operationen ergeben DB-Null, wenn ein Operand DB-Null ist.
- Eine explizite Umwandlung von DB-Null in einen Ersatzwert mittels einer Funktion NVL (*null value function*) ist möglich und wird empfohlen, z.B. 0 für Integer oder für ein Attribut RÜCKGABE vom Typ Datum mittels `NVL(RÜCKGABE, '01-01-2090')` bei einer Dauerleihe in einer Bibliothek.
- Die übliche Implementierung von Nullwerten, z.B. in DB2, ist mittels eines extra Bytes als verstecktes Attribut für die Signalisierung des Auftretens eines Nullwerts. Eine andere Alternative ist die Reservierung eines Werts (bei 3VL) aus dem Wertebereich des Attributs, z. B. $80 00 00 00 bei einem 4-Byte Integer in ESCHER.

Übung 3–4

Ist 0 ein guter interner Repräsentant für DB-Null? Ist 0 eine gute externe Darstellung von DB-Null?

Übung 3–5

Geben Sie ein Beispiel für ein numerisches Attribut A und NVL(A, x) mit $x \neq 0$ an!

Übung 3–6

SQL* Plus von Oracle läßt die leere Zeichenkette als Repräsentant von Null zu, warnt aber vor diesem „Trick“, da er u. U. zukünftig nicht mehr unterstützt werde. Warum? Welche andere interne Darstellung könnte man sich für DB-Null bei Zeichenketten vorstellen? Was ist eine gute externe Darstellung?

Übung 3–7

Wie tritt DB-Null in der Sortierordnung auf? Wie sieht ein Null-Tupel einer n-stelligen Relation aus? Wieviele kann es davon geben?

Übung 3–8

Läßt man in geschachtelten Relationen mengenwertige Attribute zu, ergeben sich neue Probleme für Nullwerte. Unterscheiden Sie zwischen der leeren Menge, der Menge mit einem Null-Tupel und DB-Null als Wert für die Menge!

Übung 3–9

Was ergibt die Vereinigung von {{A, 4711}, {B, 4711}, {?, 4712}, {?, ?}} mit {{A, 4712}, {?, 4712}, {A, ?}, {?, ?}} und wäre man vielleicht mit Multimengen besser bedient?

Die Probleme mit Multimengen, Duplikaten, und Nullwerten sind eng miteinander verwandt [DGK82]. Die Vereinigung von Multimengen kann definiert werden über das maximale Auftreten eines Elements oder über die Summe des Auftretens, also z.B. $A = \{2, 3, 5\}$, $B = \{2, 3, 3\}$, $A \cup_{\max} B = \{2, 3, 3, 5\}$, oder $A \cup_{\text{Summe}} B = \{2, 2, 3, 3, 3, 5\}$, wobei insbesondere $A \cup A \neq A$.

S_1

NAME	ALTER
Schmidt	30
Schmidt	30
Meier	40

S_2

NAME	ALTER
Müller	40

Abb. 3–2 *Vereinigung bei Multimengen*

Andererseits betrachte man $S = S_1 \cup_{\max} S_2$ aus Abbildung 3–2 oben. Bildet man zuerst die Vereinigung S und projiziert dann auf die Spalte ALTER, erhält man {30, 30, 40, 40}; projiziert man dagegen zuerst und vereinigt dann, erhält man $\{30, 30, 40\} \cup \{40\} = \{30, 30, 40\}$ – für *Vereinigung* und *Projektion* gilt in der erweiterten Relationenalgebra nicht mehr das Distributivgesetz.

3.3 Schlüssel

Das Relationenmodell ist werteorientiert, deshalb erfolgt die Unterscheidung zweier Entities (Tupel) durch Merkmalsunterschiede, d. h. durch Attributwerte. Die *minimale Menge der Attribute*, die eine eindeutige Unterscheidung der Tupel zuläßt, heißt *Schlüssel*. Existieren mehrere minimale Mengen, spricht man von Schlüsselkandidaten (engl. *candidate keys*). Einen darunter bestimmt man zum *Primärschlüssel*, für den keine Nullwerte zugelassen sind.

Die Eigenschaft, ein Schlüssel zu sein, sollte allerdings vom Schema abhängen, nicht von der zufälligen, momentanen Ausprägung der Menge der Tupel. Im *Schema* deuten wir Primärschlüssel durch Unterstreichung der Attribute an.

Beispiel 3–3

KDNR	KNAME	STADT
4711	Meier	Frankfurt
4712	Müller	Frankfurt
4701	Schmidt	Hamburg
4715	Schmidt	Hamburg

Tab. 3–1

- {KDNR} ist Schlüsselkandidat.
- KDNR ist Primärschlüssel (Kundennummer nicht zweimal vergeben und kein Kunde ohne Kundennummer!).
- KNAME und auch {KNAME, STADT} sind keine Schlüssel, da zwei Kunden gleichen Namens in der selben Stadt existieren können.
- {KDNR, KNAME} identifiziert zwar auch eindeutig jedes Tupel, wegen des überflüssigen Attributs KNAME ist die Kombination jedoch kein Schlüssel.

Der schnelle Zugriff auf Tupel über die Angabe des Primärschlüsselwerts, hier KDNR, sollte vom System immer unterstützt werden. Der Zugriff auf die Tabelle über die Kombination Kundenname und Stadt sollte allerdings auch unterstützt werden, da häufig Kunden ihre Kundenummer nicht parat haben. Man spricht dann von *Sekundärschlüsseln* (engl. *alternate keys*).

Hinweis: Häufig wird dieser Begriff auch für Attribute benutzt, für die ein Index existiert, auch wenn die *Schlüsseleigenschaft selbst nicht gegeben* ist, z. B. „Sekundärschlüssel“ KNAME.

Übung 3–10

Entwerfen Sie für die Entity-Typen LEHRER, SCHÜLER, KLASSEN geeignete Relationenschemata und bestimmen Sie die Schlüssel. Die Auf-

gabe ist insofern schwierig, als „künstliche“ Primärschlüssel für Personen nicht beliebt sind.

3.4 Fremdschlüssel und referentielle Integrität

Der Begriff *Fremdschlüssel* (engl. *foreign key,* vgl. Beispiel 2–3 auf Seite 20) bezeichnet Attribut(e) in einer Relation R_1, die Primärschlüssel in Relation R_2 sind, wobei R_1 hierarchisch von R_2 abhängt.

Beispiel 3–4

Man betrachte die Relationen aus Kapitel 2. Es sei

R_1 = BESTELLUNGEN(BESTNR, DATUM, ..., KDNR).

R_2 = KUNDEN(KDNR, KNAME, ...)

Bestellungen werden eindeutig identifiziert durch die Bestellnummer BESTNR. Die Kundennummer KDNR ist Fremdschlüssel in R_1 und Primärschlüssel in R_2, Bestellungen hängen existentiell von Kunden ab.

Über die Fremdschlüssel läßt sich *referentielle Integrität* erzwingen, z. B. wird man fordern, daß sich hinter KDNR in BESTELLUNGEN höchstens ein Kunde verbirgt, etwa wenn die Rechnung geschrieben wird, ggf. wird man einen Nullwert akzeptieren, wenn der Besteller temporär unbekannt ist.

Es muß also gelten (vgl. [Sauer], S. 28ff):

- jeder Fremdschlüsselwert in R_1 ist gleich einem Primärschlüsselwert in R_2 oder
- der Fremdschlüsselwert ist NULL.

Diese *Regel der referenziellen Integrität* vermeidet sog. *hängende Tupel* (*dangling tuples*), also im Beispiel Bestellungen mit nicht-existenten Kundennummern. Verletzungen der referentiellen Integrität sind möglich durch

- Tupel einfügen (INSERT)

- Tupel ändern (UPDATE)
- Tupel löschen (DELETE)

Übung 3–11

Vorausgesetzt seien die Tabellen BESTELLUNGEN und KUNDEN aus Kapitel 2. KDNR in BESTELLUNGEN ist Fremdschlüssel aus KUNDEN. Diskutieren Sie die folgenden Fälle!

- Einfügen einer Bestellung mit unbekannter Kundennummer 7412:

```
INSERT INTO BESTELLUNGEN VALUES
(20998, 92-11-13, ..., 7412)
```

- Ändern Kundennummer in BESTELLUNGEN auf 7412:

```
UPDATE BESTELLUNGEN SET KDNR = 7412
WHERE BESTNR = 20007
```

- Ändern Kundennummer von 4712 auf 47121:

```
UPDATE KUNDEN SET KDNR = 47121 WHERE KDNR = 4712
```

- Löschen Bestellungen von Kunden 4710:

```
DELETE FROM BESTELLUNGEN WHERE KDNR = 4710
```

- Löschen eines Kunden:

```
DELETE FROM KUNDEN WHERE KDNR = 4710
```

Übung 3–12

Beim letzten Beispiel könnte, sofern überhaupt zugelassen und der neue Eintrag in BESTELLUNGEN nicht auf NULL gesetzt wird, jede abhängige Bestellung gelöscht werden. Da BESTNR Fremdschlüssel in der Relationen BEST_POSTEN ist, kann dies wiederum zu Löschoperationen in BEST_POSTEN führen (*cascading delete*).

Diskutieren Sie diesen Fall auch an Beispielen mit ARTIKEL-LIEFERANT, bzw. SCHÜLER-KLASSE-KLASSENLEHRER mit Attribut KLASSENID in SCHÜLER als Fremdschlüssel in die Relation KLASSE hinein.

Verknüpfung über Primär- und Fremdschlüssel

Über die Primärschlüssel und Fremdschlüssel lassen sich jetzt Tabellen verknüpfen (vgl. *Join-Operation* unten), d.h. Beziehungen auswerten:

Beispiel 3–5

Die Schemata ähnlich der Abb. 2–10 lauten:

```
LIEFERANTEN(LNR, LNAME, ...)
ARTIKEL(ARTNR, BEZEICHNUNG,..., LNR)
```

LNR in ARTIKEL ist Fremdschlüssel für die Lieferanten der angebotenen Artikel (je Artikel nur ein Lieferant). Gesucht seien alle Lieferanten und Bezeichnungen der von ihnen gelieferten Artikel.

```
SELECT LNAME, BEZEICHNUNG
FROM LIEFERANTEN, ARTIKEL
WHERE LIEFERANTEN.LNR = ARTIKEL.LNR;
```

Übung 3–13

Geben Sie das Ergebnis der obigen Operation an, wenn es zwei Lieferanten (100, Müller, ...), (200, Schmidt, ...) und drei Artikel (4710, Bolzen, ..., 200), (4711, Mutter, ..., 100), (4712, Unterlegscheibe, ..., 100) gibt!

Übung 3–14

Betrachten Sie die Tabellen MITARBEITER und ABTEILUNG.

MITARBEITER

NAME	MID	GJAHR	ABTID
Peter	1022	1959	FE
Gabi	1017	1963	HR
Beate	1305	1978	VB
Rolf	1298	1983	HR
Uwe	1172	1978	FE

ABTEILUNG

ABTID	ABTBEZ	ORT
FE	Forschung & Entwicklung	Dresden
HR	Personalabteilung	Kassel
EC	e-Commerce	?
VB	Vertrieb & Beratung	Dresden

Diskutieren Sie Schlüssel, Fremdschlüssel, referentielle Integrität.

3.5 Operationen der relationalen Algebra

Generell bezeichne $R(A_1: W_1, A_2: W_2, ..., A_n: W_n)$ eine n-stellige Relation mit Attributnamen $A_1, A_2, ..., A_n$ und Wertebereichen $W_1, W_2, ..., W_n$. Falls die Wertebereiche nicht interessieren schreiben wir $R(A_1, A_2, ..., A_n)$ oder nur R.

R ist dabei der Name einer *Relationenvariablen*. Daneben können namenlose Relationen aus *Literalen* und *Relationenausdrücken* entstehen.

Ein Tupel aus $R(A_1, A_2, ..., A_n)$ werde mit $(A_1: a_1, A_2: a_2, ..., A_n: a_n)$ oder nur kurz mit $(a_1, a_2, ..., a_n)$ bezeichnet.

Beispiel 3–6

Aus der Relation(envariablen) ARTIKEL(ANR, BEZ, PREIS, LAGER-ORT) bilden wir durch Vereinigung mit dem Tupelliteral (255, ’Schraube’, 0.20, A101) und anschließender Projektion auf {ANR, BEZ} einen Relationenausdruck, dessen Ergebnis eine zweispaltige, namenlose Relation ist.

Restriktion oder Selektion

Restriktion und Selektion sind gleichwertige Begriffe für die Auswahl von Zeilen gemäß einer Bedingung.

Syntax $\sigma_B(R)$

Eine Selektion liefert alle Tupel der Relation *R*, die Bedingung *B* erfüllen. Bedingungen werden formuliert mittels Vergleichsoperatoren <, >, = , ≥, ≤, ≠, die als Operanden Konstante (dargestellt durch Literale) und Attributwerte (angegeben durch Attributnamen A_i) enthalten. Bedingungen können über die Boolschen Operatoren UND (∧), ODER (∨), NICHT (¬) verknüpft werden.

Die Selektion, nicht zu verwechseln mit dem mächtigeren SQL-SELECT, ist damit wie die folgende Projektion eine unäre Operation.

TLISTE

NAME	FB	TEL
Hans	17	4477
Emil	19	3443
Anna	17	4476
Gabi	?	2441

$\sigma_{FB = 17}$ **(TLISTE)**

NAME	FB	TEL
Hans	17	4477
Anna	17	4476

Projektion

Unter Projektion versteht man die Auswahl von Spalten einer Relation. Dadurch entstehende doppelte Zeilen (Duplikate) müssen entfernt werden (Mengeneigenschaft!).

Syntax $\pi_L (R)$ wobei Liste $L = A_{i1}, A_{i2}, ..., A_{im}$
Attribute aus Relation *R* enthält.

TLIST

NAME	FB	TEL
Hans	17	4477
Emil	19	3443
Anna	17	4476
Gabi	?	2441

$\pi_{\text{NAME, FB}}$ (TLIST)

NAME	FB
Hans	17
Emil	19
Anna	17
Gabi	?

Produkt

Die Produktbildung ist eine binäre Operation. Sie nimmt zwei Tabellen und erzeugt eine neue Tabelle aus der Kombination aller Zeilen (Karthesisches Produkt).

Syntax $R \times S$ mit R, S Relationen

Die Resultatsrelation hat die Attribute ($R.A_1$, $R.A_2$, ..., $R.A_n$, $S.A_1$, $S.A_2$, ...,$S.A_m$) mit „Punktnotation“ zur Unterscheidung gleichnamiger Attribute.

TLIST

NAME	FB	TEL
Hans	17	4477
Emil	19	3443

FBNAMEN

BEZ	FB
Math-Inf	17
E_Tech	19
WiWi	7

TLIST $\times$ FBNAMEN

TLIST. NAME	TLIST. FB	TLIST. TEL	FBNA-MEN.BEZ	FBNA-MEN.FB
Hans	17	4477	Math-Inf	17
Hans	17	4477	E-Tech	19
Hans	17	4477	WiWi	7
Emil	19	3443	Math-Inf	17
Emil	19	3443	E-Tech	19
Emil	19	3443	WiWi	7

Vereinigung

Zwei verträgliche Relationen R und S lassen sich vereinigen. Dies setzt gleichen Grad der Relationen voraus und jedes Attribut der ersten Relation muß verträglich sein mit dem korrespondierenden Attribut der zweiten Relation, d.h. $R.W_i = S.W_i$ für $i = 1, ..., n$.

Syntax $R \cup S$

Das Ergebnis ist eine namenlose Relation mit neuen Attributnamen U_i, z.B. der lexikographisch kleinere von $R.A_i$ und $S.A_i$. Wegen der beliebigen Anordnung der Attribute ist ggf. eine Umordnung der Spalten vor der Vereinigung nötig. Üblicherweise wird man auch eine semantische Verträglichkeit der Attribute fordern, d.h. FB und TEL wären semantisch nicht verträglich, obwohl für beide Wertebereiche u.U. Integer als Typ vereinbart war.

TLIST

NAME	TEL
Hans	4477
Emil	3443

TELNUM

TEILN	DURCHWAHL
Hans	4477
Egon	2441

TLIST ∪ TELNUM

NAME	DURCHWAHL
Hans	4477
Emil	3443
Egon	2441

Differenz

Finde alle Tupel, die in Relation *R*, aber nicht in Relation *S*, sind. Es wird wieder verlangt, daß die Relationen verträglich (engl. *union-compatible*) sind.

Syntax $R - S$

Als Beispiel ergibt etwa TLIST - TELNUM die Relation mit dem einzelnen Tupel (Emil, 3443).

Die fünf genannten Operationen sind notwendig und hinreichend für eine relationale Algebra. Zusammenfassend [KS86] gilt also:

relationale Ausdrücke *E* lassen sich aufbauen aus den Grundbausteinen

- Relationenvariablen
- Relationenkonstanten

und, wenn E_1 und E_2 relationale Ausdrücke sind, entsteht mit

- $E_1 \cup E_2$
- $E_1 - E_2$

- $E_1 \times E_2$
- $\sigma_P(E_1)$ mit P Prädikat über den Attributen von E_1
- $\pi_S(E_1)$ mit S Liste von Attributen aus E_1

wieder ein relationaler Ausdruck.

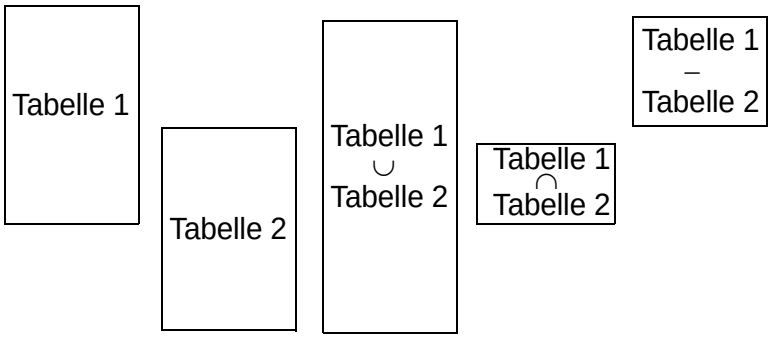


Abb. 3–3 *Bildliche Darstellung der Operationen nach [OR87]*

Manche Autoren rechnen auch die Umbenennung von Attributen β zu den essentiellen Operationen. Wir haben sie implizit vorausgesetzt.

Beispiel 3–7 Relationaler Ausdruck

Gesucht seien aus der Telefonliste TLIST die Namen und Fachbereiche aller Einträge, ergänzt um die Fachbereichsbezeichnung BEZ aus FBNAMEN.

$$\pi_{\texttt{TLIST.NAME, TLIST.FB, FBNAMEN.BEZ}}\ (\sigma_{\texttt{TLIST.FB = FBNAMEN.FB}}(\texttt{TLIST} \times \texttt{FBNAMEN}))$$

Übung 3–15

Wie sieht das Ergebnis aus mit den Tabellen von oben?

Zusätzliche Operationen

Üblicherweise arbeitet man aber mit den folgenden zusätzlichen Operatoren.

Durchschnitt

Syntax $R \cap S$ mit R, S verträgliche Relationen.

Beispiel 3–8 Durchschnitt

TLIST $\cap$ TELNUM ergibt die Relation mit dem einzelnen Tupel (Hans, 4477).

Join, Natural Join, Theta Join

Eine der häufigsten Operation im relationalen Datenmodell ist die Produktbildung mit anschließender Selektion (vgl. Beispiel oben). Speziell der sog. *Natural Join*, d. h. die Selektion auf Gleichheit der beiden *Joinattribute* unter Aufnahme nur eines der beiden Attribute, wird zur Auswertung von Beziehungen benötigt. Wird keine weitere Einschränkung gemacht, versteht man unter *Join* immer den *Natural Join*.

Im allgemeinen Fall spricht man vom *Theta-Join*, wobei Theta ein Prädikat ist.

Syntax $R \bowtie_{\Theta} S$ (ausgesprochen: Theta-Join von R über Attribut A_i mit S über Attribut A_j)

Damit also $R \bowtie_{\Theta} S = \sigma_{\Theta} (R \times S)$;

$A_i \Theta A_j$ ergibt wahr auf den selektierten Attributen.

Im Fall des Natural Join über das Joinattribut FB oben schreiben wir einfach TLIST $\bowtie$ FBNAMEN. Generell gilt [Sauer]:

- Die Wertebereiche der Joinattribute sind gleich.
- Joinattribute brauchen keine Schlüsselattribute zu sein.
- Die Joinattribute brauchen nicht die selben Namen zu haben.
- Jede Relation kann mit jeder anderen durch einen Join verbunden werden, auch mit sich selbst.

Beispiel 3–9 Join

Gesucht sind Händler mit Kunden von außerhalb.

$$\text{HÄNDLER} \bowtie_{\text{KUNDEN.ORT} \neq \text{HÄNDLER.STADT}} \text{KUNDEN}$$

HÄNDLER

HNAME	STADT
Schmidt KG	Hamburg
Müller GmbH	München

KUNDEN

KNAME	ORT
Meier	Hamburg
Wagner	Hamburg
Kunze	Kassel
Bauer	München

HÄNDLER $\bowtie_{\text{ORT} \neq \text{STADT}}$ KUNDEN

HNAME	STADT	KNAME	ORT
Schmidt KG	Hamburg	Kunze	Kassel
Schmidt KG	Hamburg	Bauer	München
Müller GmbH	München	Meier	Hamburg
Müller GmbH	München	Wagner	Hamburg
Müller GmbH	München	Kunze	Kassel

Equi-Join

Basiert das Vergleichsprädikat Θ auf der Gleichheit, spricht man vom *Equi-Join*. Im Gegensatz zum Natural Join werden selbst bei Namensgleichheit beide Attribute aufgeführt.

Übung 3–16

Wie sieht das Ergebnis des Equi-Joins von HÄNDLER über STADT mit KUNDEN über ORT aus?

Übung 3–17

Wie lautet das Ergebnis des zweifachen Joins

$$\text{PROJEKT} \bowtie \text{ARBEITET\_AN} \bowtie \text{MITARBEITER}$$

mit den unten angegebenen Tabellen? Das Attribut ZEIT in ARBEITET_AN bezeichne den prozentualen Zeitanteil. Spielt die Reihenfolge eine Rolle?

PROJEKT

PID	NAME
4711	RZ-Ausbau
4712	PC-Versorgung

ARBEITET_AN

PID	MNR	ZEIT
4711	100	50
4711	250	100
4712	100	50
4712	150	100
4712	200	100

MITARBEITER

MNR	MNAME
100	Hans
150	Gabi
200	Inge
250	Rolf

Übung 3–18

Wie oben, aber mit den Tabellen BESTELLUNGEN, BEST_POSTEN und ARTIKEL aus Kapitel 2!

Bei komplexeren relationalen Ausdrücken gibt es gewöhnlich mehrere Abarbeitungsstrategien mit großen Unterschieden im Aufwand. Speziell der Join mit der Produktbildung als Zwischenschritt ist wegen der Größe der entstehenden Tabellen teuer.

Übung 3–19

Gesucht seien die Mitarbeiter von Projekt 4711. Wie geht man vor?

Übung 3–20

Zwei offensichtliche Implementierungsarten für den Join sind

- äußere und innere Schleife
- *Sort-Merge-Join* (Join mittels Sortieren und Mischen).

Beschreiben Sie die beiden Methoden und schätzen Sie den Aufwand für Tabellen der Größe *n*, bzw. *m* ab!

Semi-Join

Häufig interessieren nach einem Join nur die Attribute der linken Relation. Diese implizite Projektion nennt man *Semi-Join*, kurz

> $R \ltimes S = \pi_R (R \bowtie S)$, wobei das tiefgestellte *R* die Abkürzung für „Attribute von *R*“ ist.

Beispiel 3–10 Semi-Join

siehe Tabellen *R* und *S* unten (aus [Ullm], S. 60f).

Übung 3–21

Zeigen Sie, daß $R \ltimes S = R \bowtie \pi_{R \cap S} (S)$, d. h. der Semi-Join von *R* mit *S* ist gleichwertig mit dem Join von *R* mit *S* nachdem *S* auf die gemeinsamen Attribute projiziert wurde.

Outer Join

In einem „normalen“ Join nehmen nur die Tupel teil, die Bedingung Θ erfüllen. Man nennt ihn auch „*innerer Join*“. Möchte man generell auch alle Tupel „ohne Treffer“ auflisten, bietet es sich an, diese mit Nullwerten auf der trefferlosen Seite aufzufüllen. Man spricht dann vom „*äußeren Join*“ (*outer join*).

R

A	B	C
a	b	c
d	b	c
b	b	f
c	a	d

S

B	C	D
b	c	d
b	c	e
a	d	b

R ⋈ S

A	B	C	D
a	b	c	d
a	b	c	e
d	b	c	d
d	b	c	e
c	a	d	b

R ⋉ S

A	B	C
a	b	c
d	b	c
c	a	d

Nimmt man nur die Tupel der linken oder rechten Seite auf, spricht man vom *left outer join*, bzw. *right outer join* (linker und rechter äußerer Join).

Übung 3–22

In TLIST(NAME, FB, TEL) seien durch Eingabefehler manche Fachbereiche falsch aufgeführt, z.B. FB = 71 statt FB = 17. Wie sähe der natürliche, äußere Join mit der Tabelle FBNAMEN(BEZ, FB) aus? Wieso wäre er nützlich?

Division oder Quotient

Man erläutert die *Division* (manchmal *Quotient* genannt) am leichtesten mit einem Beispiel. Gegeben seien

- Relation ARBEITET_AN(PID, MNR) mit Projekten und Mitarbeitern,

R

A	B
a	a
a	b
b	d
c	a

S

B	C
a	b
b	a
b	c
c	a

Outer Join R * ⟗ * S

A	B	C
a	a	b
a	b	a
a	b	c
c	a	b
b	d	?
?	c	a

- Relation TEAM(MNR) mit Mitarbeiternummern.

Gesucht sind Projekte (PID), die **alle** im TEAM aufgeführten Mitarbeiter beschäftigen (vgl. Tabellen unten).

Syntax $R \div S$

Für Relationen *R* und *S* seien die Attribute von *S* eine Teilmenge der Attribute von *R*. Die Attribute von *S* heißen *Divisionsattribute*. Ergebnis der Division von *R* durch *S* ist eine Projektion auf die Attribute von *R*, die **nicht** Divisionsattribute sind. Das Resultat entsteht durch Bildung von *Äquivalenzklassen* über den Nicht-Divisionsattributen (hier: {(4711, 100), (4711, 250)} und {(4712, 100), (4712, 150), (4712, 200)}. Ein Vertreter aus jeder Klasse – projiziert auf die Nicht-Divisionsattribute – wird aufgenommen, wenn **jedes Tupel aus *S*** sich in der Klasse bezüglich der Divisionsattribute wiederfindet.

Die Division ist nützlich im Zusammenhang mit „für alle-Abfragen“. Im Beispiel etwa: „Liste alle Projekte *p* auf, so daß **für alle** Mitarbeiter *m* in TEAM gilt, *m* arbeitet an *p* mit“.

ARBEITET_AN

PID	MNR
4711	100
4711	250
4712	100
4712	150
4712	200

TEAM

MNR
100
150

ARBEITET_AN ÷ TEAM

PID
4712

Da die Division nicht zu den fünf essentiellen Operationen gehört, kann sie umgangen werden. Allgemein gilt, wenn *R*/*S* die Nicht-Divisionsattribute aus *R* bezeichnet:

$$R \div S = \pi_{R/S}(R) - \pi_{R/S}((\pi_{R/S}(R) \times S) - R)$$

speziell

$$\mathrm{ARBEITET\_AN} \div \mathrm{TEAM} = \pi_{\mathrm{PID}}(\mathrm{ARBEITET\_AN}) - \pi_{\mathrm{PID}}((\pi_{\mathrm{PID}}(\mathrm{ARBEITET\_AN}) \times \mathrm{TEAM}) - \mathrm{ARBEITET\_AN})$$

Übung 3–23

Überprüfen sie die spezielle Umformung am Beispiel.

Übung 3–24

Wäre ZEIT Attribut von ARBEITET_AN wie in den Tabellen zu Übung 3–17 oben, wie würde die Division ausgehen?

Übung 3–25

Bei den einstelligen Operationen Restriktion und Projektion bleiben Grad und Kardinalität des Ergebnisses unverändert, bzw. werden kleiner. Wie sieht es bei Vereinigung, Produkt, Theta-Join und Differenz aus?

NF^2-Algebra

Erweitert man die relationale Algebra auf geschachtelte Relationen kann man die folgenden Vorschläge für eine passende Algebra machen.

- $\cup$, -, π, $\rtimes$ werden wie bei der Relationenalgebra eingeführt.
- Die Selektion σ wird erweitert um Relationen als Operanden und Mengenvergleiche $\Theta := \subseteq, \subset, \supset, \supseteq, =$.
- Es gibt jetzt rekursiv aufgebaute Operationsparameter; so können π und σ auch innerhalb von Projektionslisten und Selektionsbedingungen dort angewendet werden, wo relationenwertige Attribute auftauchen.
- Zwei zusätzliche Operatoren ν und μ werden eingeführt für Nestung (engl.: *nest*) und Entnestung (*unnest*, flattening).

Beispiel 3–11 Nest und Unnest

Leider sind nest und unnest im Normalfall nicht inverse Operationen, spez. wenn die geschachtelte Relation nicht in der sog. *Partitioned Normal Form* (PNF) ist, d.h. wenn die Relation kein atomares Attribut (oder mehrere atomare Attribute) mit Schlüsseleigenschaft besitzt (siehe Abbildung unten).

3.6 Das tupel-relationale Kalkül

Die obigen Operationen der relationalen Algebra ergeben in geeigneter Schreibweise eine prozedurale Datenabfrage- und Manipulationssprache. Im allgemeinen wird man aber ein (Prädikaten-)Kalkül vorziehen, in dem

- Abarbeitungsschritte nicht explizit anzugeben sind,

A	B	C
1	2	7
1	3	6
1	4	5
2	1	1

$\rightarrow$
$\nu_{B,C;D}(r)$
$\leftarrow$
$\mu_{D}(r')$

A	D	
	B	C
1	2	7
	3	6
	4	5
2	1	1

A	B	C
1	2	7
1	3	6
1	4	5
2	1	1

$-/\rightarrow$
$\nu_{B,C;D}(r)$
$\leftarrow$
$\mu_{D}(r')$

A	D	
	B	C
1	2	7
	3	6
1	4	5
2	1	1

- die Beschreibung der gewünschten Eigenschaften mittels einer Formel geschieht, und

in der Variablen, die Tupel repräsentieren, auftreten dürfen (sog. *tupel-relationales Kalkül*).

Formal beschreiben wir das gewünschte Ergebnis mit

```
{t | P(t)}
```

und sprechen von der gesuchten Menge aller Tupel *t*, so daß Prädikat *P* für *t* wahr ergibt (*P* für *t* gilt).

Bezeichne im folgenden $t \in R$ das Auftreten von Tupel *t* in Relation *R* und bezeichne *t*[*A*] den Wert des Attributs *A* in Tupel *t*.

Beispiel 3–12

Gesucht seien alle Teilnehmertupel aus TLIST, die im FB 17 sind, aber keine 4000-er Telefonnummern haben.

$$\{t \mid t \in \text{TLIST} \wedge t[\text{FB}] = 17 \wedge (t[\text{TEL}] < 4000 \vee t[\text{TEL}] > 4999)\}$$

Sind wir nur am Namen des Teilnehmers interessiert, würden wir in der Algebra eine Projektion auf Attribut NAME machen. Im Kalkül müssen wir eine neue Tupelvariable *s* einführen und sie mit dem *Existenzquantor* ∃ binden:

$$\exists\, s\ (Q(s))$$

gesprochen: es existiert ein *s*, so daß *Q*(*s*) gilt. Damit lautet die Formel dann

$$\{t \mid \exists\, s\ (s \in \text{TLIST} \wedge s[\text{FB}] = 17 \wedge (s[\text{TEL}] < 4000 \vee s[\text{TEL}] > 4999) \wedge s[\text{NAME}] = t[\text{NAME}])\ \}$$

wobei *t freie* und *s gebundene Variable* heißt.

Beispiel 3–13

Gibt es Fachbereiche, außer 17, die 4000-er Nummern haben?

$$\{\ t \mid \exists\, s\ (s \in \text{TLIST} \wedge s[\text{FB}] \neq 17 \wedge s[\text{TEL}] < 5000 \wedge s[\text{TEL}] > 3999 \wedge \exists\, u\ (u \in \text{FBNAMEN} \wedge u[\text{FB}] = s[\text{FB}] \wedge u[\text{BEZ}] = t[\text{BEZ}]))\ \}$$

Der zusätzliche Existenzquantor entspricht offensichtlich dem Join in der Algebra.

Bei der Darstellung der Division tritt neben dem Existenzquantor noch der *Allquantor* (∀) auf.

Beispiel 3–14

Gesucht sind die Projektidentifier PID der Projekte *t*, an denen alle Mitarbeiter *u* aus TEAM mitwirken, d. h. für alle *u* muß ein Tupel *s* in

ARBEITET_AN mit der gewünschten Kombination Projektidentifier PID, Mitarbeiternummer MNR existieren.

$$\{t \mid \forall\ u\ (u \in \mathrm{TEAM} \wedge \exists\ s\ (s \in \mathrm{ARBEITET\_AN} \wedge s[\mathrm{PID}] = t[\mathrm{PID}] \wedge u[\mathrm{MNR}] = s[\mathrm{MNR}]))\ \}$$

Statt $\forall\ t\ (P(t))$ kann man äquivalent schreiben $\neg\exists\ t\ (\neg P(t))$, z.B. „.. existiert kein Mitarbeiter *u*, der nicht an Projekt *t* mitwirkt."

Übung 3–26

Man gebe zwei alternative Formulierungen mit $\forall$ und $\neg\exists$ für die folgende Aufgabe an! Gibt es in der Relation FBNAMEN ein Tupel für einen Fachbereich, der keinen Teilnehmer in TLIST hat?

Übung 3–27

Gesucht sind die Namen von Mitarbeitern, die mit weniger als 100% ihrer Arbeitszeit an der PC-Versorgung mitarbeiten
(Tabellen aus Übung 3–17).

Übung 3–28

Mit den Schemata aus Kapitel 2 suche man nach Lieferanten, deren Artikel momentan in keiner Bestellung auftauchen!

Ein Problem des Kalküls ist die Möglichkeit, eine unendliche Relation als Ergebnis einer Prädikatsauswertung zu erhalten. Werden z. B. alle Namen gesucht, die nicht in TLIST unter FB 17 eingetragen sind

$$\{\ t \mid \neg\exists\ s\ (s \in \mathrm{TLIST} \wedge s[\mathrm{FB}] = 17 \wedge t[\mathrm{NAME}] = s[\mathrm{NAME}])\}$$

so erfüllen auch *t*[NAME] = 'Rumpelstilzchen', *t*[NAME] = 'Heinrich VIII', usw. dieses Prädikat! Offensichtlich gefragt waren Namen aus der Relation TLIST oder ggf. sonstige, in der Datenbasis vorkommende Namen.

Daher fordert man die Beschränkung auf sog. *sichere Formeln* und definiert den Wertebereich DOM(*F*) einer Formel *F* als die Menge aller in

F selbst als Konstanten definierten Werte vereinigt mit den Wertebereichen der angesprochenen Relationen.

Damit wäre die Abfrage

$$\{ t \mid F(t)\} = \{ t \mid t[NAME] \neq \text{'Rumpelstilzchen'} \wedge \neg\exists\, s\, (s \in TLIST \wedge s[FB] = 17 \wedge s[NAME] = t[NAME]) \} = \{ t \mid t[NAME] \neq \text{'Rumpelstilzchen'} \wedge \forall\, s\, (s \in TLIST \wedge (s[FB] \neq 17 \vee s[NAME] \neq t[NAME])) \}$$

zulässig, die Antworten aber auf Namen aus TLIST beschränkt, wegen DOM(*F*) = {'Rumpelstilzchen'} ∪ {'Hans', 'Emil', 'Anna', 'Gabi', 17, 19, 4477, 3443, 4476, 2441, NULL}, d. h. es ist sichergestellt, daß für die Suche nach Namen, die beide logischen Ausdrücke (... ≠ 'Rumpelstilzchen' ∧ ∀ *s* (...)) erfüllen, keine endlose Schleife zur Generierung von Werten erzeugt wird.

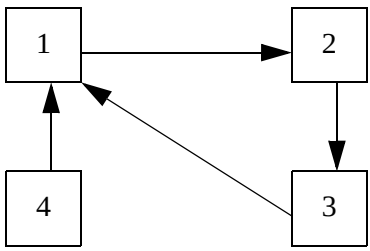
Die Forderung, daß in sicheren Formeln nur Werte aus DOM(*F*) betrachtet werden, schränkt die Mächtigkeit des Kalküls (engl. *safe tuple relational calculus*) im Vergleich zur relationalen Algebra nicht ein.

Übung 3–29

Betrachten Sie die folgende binäre Relation *R*(X, Y) mit nebenstehender graphischen Interpretation.

R

X	Y
1	2
2	3
3	1
4	1



Gesucht ist der transitive Abschluß *R** von *R*, wobei ein Tupel $t \in R^*$ angibt, daß *t*[Y] von *t*[X] aus erreichbar ist. Kann man die Tabelle im tupel-relationalen Kalkül formulieren?

Übung 3–30

Kann man die folgende Abfrage im tupel-relationalen Kalkül stellen?

> „Wie groß ist der durchschnittliche Arbeitszeitanteil der Mitarbeiter an jedem der Projekte an denen sie arbeiten?“

Die beiden Übungsbeispiele zeigen die Grenzen des Kalküls, wobei letztere Einschränkung durch die Hereinnahme von Gruppierungs- und Berechnungsfunktionen – wie in SQL auch vorhanden – zu beseitigen ist. Allgemein gibt es für die Mächtigkeit der Abfragesprachen die sog. *Chandra-Hierarchie* [Cha88]. Wir zeigen hier die geringfügig andere Klassifikation aus [JV85].

Kategorie	**Bemerkung**
Aussagenlogik	nur einzelne Tabelle
Tableau Abfragen	Grundlage der „universal relation Abfragen“
konjunktive Abfragen	kein ODER erlaubt
existenzielle Abfragen	disjunktive Abfragen erlaubt
relational vollständig	Prädikatenlogik 1. Stufe, Allquantor erlaubt
Fixpunkt Abfragen	Horn-Klauseln (Prolog), transitiver Abschluß möglich
volle Logik 1. Stufe	Disjunktion/Negation, unvollst. Information
Logik 2. Stufe	Welche Tabelle enthält Information über ...
volle Programmierfähigkeit	Type 0 Mächtigkeit

Tab. 3–2 Hierarchie der Sprachmächtigkeit von Abfragesprachen (aus [JV85])

4 Die relationale Abfragesprache SQL

4.1 Übersicht

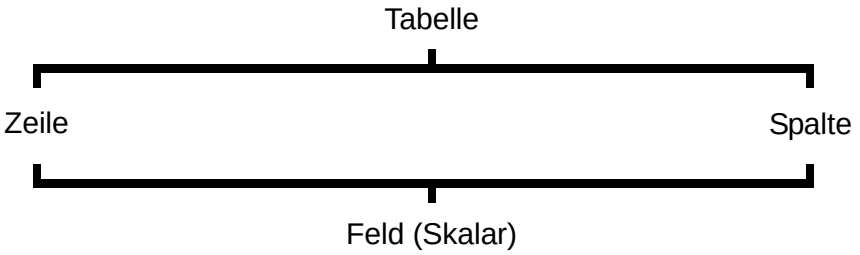
Relationale Algebra und Tupelkalkül sind für praktische Anwendungen wenig geeignet. Die universelle Datenbanksprache ist heute SQL (Structured Query Language). Daneben sind vielleicht noch QBE (Query-by-example) [Zloof77] und OQL, die Object Query Language der ODMG [Catt94], von Interesse.

Wir behandeln hier nur SQL, das Anfang der siebziger Jahre für das System/R, einer Forschungsentwicklung der IBM und Vorläufer von DB2 [Astr76], entwickelt wurde. SQL ist inzwischen als ISO-Standard in verschiedenen Entwicklungsstufen genormt (SQL-86, SQL-89, SQL-92, SQL-99, SQL:2003). SQL-89 wird von allen relationalen Datenbanksystemen, SQL-92 (auch SQL2 genannt) von den großen unterstützt.

Die neuere Version SQL-99 (nach neuerer Schreibweise SQL:1999, genauer ISO/IEC 9075:1999, auch SQL3 genannt) ist noch nicht in allen Datenbanksystemen implementiert. SQL:2003 ist noch weitgehend unimplementiert (lt. Wikipedia).

Ferner existieren Einbettungen in alle höheren Programmiersprachen wie COBOL, FORTRAN, Pascal, PL/I, C, Ada und Java (ab SQL-99).

Nach [Date86, S. 272] sind die folgenden Klassen von Objekten in SQL zu unterscheiden.



Übung 4–1

Wie lauten die Sprechweisen in der Relationenalgebra?

Beispiel 4–1 Anlegen einer Tabelle

Definition und Anlegen einer leeren Artikeltabelle.

```
CREATE TABLE ARTIKEL(ARTNR SMALLINT,
  BESTAND INTEGER, LAGERORT CHAR(7))
```

Einfügen von drei Artikeln

```
INSERT INTO ARTIKEL VALUES(100, 50, 'A301')
INSERT INTO ARTIKEL VALUES(200, 0, 'A302')
INSERT INTO ARTIKEL VALUES(150, 10, 'C111')
```

Gesucht seien Artikelnummer und Lagerort aller Artikel mit Bestand > 0:

```
SELECT ARTNR, LAGERORT
FROM ARTIKEL
WHERE BESTAND > 0
```

Das Ergebnis ist eine Tabelle der Art

ARTNR	LAGERORT
100	A301
150	C111

Beispiel 4–2

Erzeuge eine leere Mitarbeiter-Tabelle analog zu Übung 3.14.

```
CREATE TABLE MITARBEITER
     (NAME VARCHAR (40), MID INTEGER,
     GJAHR SMALLINT, ABTID CHAR(3))
```

In diese Tabelle fügen wir die fünf Mitarbeiter aus dem Beispiel ein.

```
INSERT INTO MITARBEITER VALUES
     ('Peter', 1022, 1959, 'FE')

...

INSERT INTO MITARBEITER VALUES
     ('Uwe' , 1172, 1978, 'FE')
```

Existiert auch schon die ABTEILUNGS-Tabelle, können wir nach Namen, Geburtsjahr und Abteilungsort von Mitarbeitern mit Geburtsjahr nach 1975 suchen.

```
SELECT NAME, GJAHR, ORT
  FROM MITARBEITER M, ABTEILUNG A
  WHERE M.ABTID = A.ABTID AND GJAHR > 1975
```

Dies ist die Abfrage im Ausdruck

$$\Pi_{\text{M.NAME, M.GJAHR, A.ORT}} \left(\sigma_{\text{M.ABTID = A.ABTID} \wedge \text{M.GJAHR > 1975}} (M \times A)\right)$$

und in der Formel

$$\{s \mid \exists t\ (t \in M \wedge t[\text{GJAHR}] > 1975 \wedge s[\text{NAME}] = t[\text{NAME}] \wedge s[\text{GJAHR}] = t[\text{GJAHR}] \wedge \exists u\ (u \in A \wedge s[\text{ORT}] = u[\text{ORT}] \wedge t[\text{ABTID}] = u[\text{ABTID}]))\ \}$$

für den tupel-relationalen Kalkül. Das Ergebnis ist die folgende Tabelle.

NAME	GJAHR	ORT
Beate	1978	Dresden
Rolf	1983	Kassel
Uwe	1978	Dresden

Anzumerken ist, dass Abfragen in SQL genau wie die Operationen der relationalen Algebra oder die Ausdrücke im Relationenkalkül als Ergebnis wieder Tabellen (Relationen) liefern. Programmiersprachen können aber nur einzelne atomare Werte oder ein Tupel nach dem anderen verarbeiten, man spricht von einem sog. *Impedance Mismatch* zwischen den Sprachen. Um über Tabellen iterieren zu können, führt SQL deshalb zusätzlich das Konzept des Datenzeigers (CURSOR) ein. Eine genauere Behandlung findet sich z. B. in [KE].

Im folgenden werden wir SQL an Beispielen etwas genauer erläutern. Dabei werden wir auch die Tabellendefinition oben um die Angabe von Primär- und Fremdschlüssel erweitern.

SQL-86

SQL-86 hat vier Teile:

- *Schemadefinitionssprache*: beschreibt Tabellen, Sichten und Privilegien
- *Datenmanipulationssprache*: für Zugriff und Änderung der Daten
- *Modul-Sprache*: prozedurales Interface für Anwendersprache, heute API (Application Programm Interface) genannt
- *Eingebettete Syntax*: Einstreuen von SQL-Anweisungen in Anwenderprogramme als Alternative (abkürzenden Schreibweise) zur Modul-Sprache

Begriff des *SQL-Schemas* umfaßt Tabellen, Views und Privilegien, erlaubt Namenserweiterung, definiert aber keinen Gültigkeitsbereich

```
CREATE SCHEMA AUTHORIZATION PAYROLL
CREATE TABLE EMPLOYEE
 (NUMBER CHAR(5) UNIQUE,
 NAME CHAR(10),
 DEPT_ID CHAR(5),
 DRIVERS_LICENSE CHAR(10) UNIQUE,
 SALARY FIXED(9,2),
 COMMISSION FIXED (9,2),
 MARITAL_STATUS CHAR(1))
...
```

```
CREATE VIEW DEPTEMP
 ...
GRANT SELECT, UPDATE(NAME)
 ON EMPLOYEE
 TO PERSONEL
```

- kein ALTER oder DROP im Standard
- keine INDEX-Definition, UNIQUE-Klausel im CREATE TABLE statt wie bei kommerziellen Produkten oft im CREATE INDEX
- keine DB-Spaces, CLUSTER-Klauseln, keine SEGMENTS
- keine on-line Interpretationsmechanismen
- keine Angaben zur Katalog-Tabelle

Datenmanipulationssprache mit den Elementen

```
INSERT
DELETE
UPDATE
SELECT
OPEN
FETCH
CLOSE
COMMIT
ROLLBACK
```

SELECT als Schlüssel zu SQL, z.B.

```
SELECT DEPT_ID, AVG(SALARY), MAX(COMMISSION)
FROM EMPLOYEE
WHERE MARITAL_STATUS = 'M'
GROUP BY DEPT_ID
```

```
SELECT DEPT_ID, AVG(SALARY), MAX(COMMISSION)
FROM EMPLOYEE
WHERE MARITAL_STATUS = 'M'
GROUP BY DEPT_ID
HAVING AVG(SALARY) < 1000 AND DEPT_ID LIKE 'A%B'
```

Abarbeitungsreihenfolge üblicherweise: Start mit FROM zur Erzielung einer Tabelle oder Kombination von Tabellen, dann Auswertung von WHERE, GROUP BY, HAVING in dieser Reihenfolge, dann Ausdrücke der SELECT-Liste.

SQL selbst ist mengenorientiert, aber die Schnittstellen zu Programmiersprachen verlangen „eine Zeile nach der anderen“ (Schlagwort: one tuple at a time).

Begriff des Cursors mit DECLARE, OPEN, FETCH, CLOSE. Cursor verhält sich wie ein benannte SELECT-Anweisung.

```
DECLARE CURSOR DEPTSCAN
  FOR SELECT
  (ID, DEPARTMENT.NAME, EMPLOYEE.NAME, AGE, SALARY)
  FROM DEPARTMENT, EMPLOYEE
  WHERE DEPARTMENT.ID = :X
  AND EMPLOYEE.JOBCODE = :Y
  AND DEPARTMENT.ID = EMPLOYEE.DEPT_ID

OPEN DEPTSCAN

loop:
  FETCH DEPTSCAN
  INTO (:IDVAR, :DEPTNAME, :EMPNAME, :AGE, :SALARY)

end loop;

CLOSE DEPTSCAN
```

„:X“ und „:Y“ Eingabewerte (aus Programmvariablen) zur Laufzeit, damit gesteuerter Aufsetzpunkt.

SQL-89

Ergänzung von SQL-86 im wesentlichen zur Stärkung der Integrität

- DEFAULT-Klausel
- CHECK-KLausel
- Grundlagen der referentiellen Integrität

DEFAULT als Alternative zu fehleranfälligem DB-Null, z. B. in EMPLOYEE

```
...
SALARY FIXED(9,2) DEFAULT (-1),
COMMISSION FIXED(9,2),
MARITAL-STATUS CHAR(1) DEFAULT('U'))
```

analog CHECK-Klausel als rudimentäre Wertebereichsprüfung

```
...
SALARY FIXED(9,2) DEFAULT (-1)
CHECK (SALARY = -1 OR SALARY > 0),
COMMISSION FIXED(9,2)
CHECK(SALARY > COMMISSION),
MARITAL_STATUS CHAR(1) DEFAULT('U')
CHECK MARITAL_STATUS IN ('S', 'M', 'D', 'U'))
```

Referentielle Integrität über Klauseln

> **PRIMARY KEY** (max. einmal je Tabelle)
> **REFERENCES** *Tabellenname*, wenn Tabelle Primärschlüssel spezifiziert hat, bzw.
> **REFERENCES** *Tabellenname*(*Spaltenname*)

```
CREATE TABLE EMPLOYEE
(NUMBER CHAR(5) PRIMARY KEY,
NAME CHAR(10),
DEPT_ID CHAR(5) REFERENCES DEPARTMENT,
DRIVERS_LICENSE CHAR(10) UNIQUE,
...
```

alternativ z. B. auch

```
...
DEPT_ID CHAR(5),
FOREIGN KEY(DEPT_ID) REFERENCES DEPARTMENT(ID)
```

Verletzungen der Integrität werden abgefangen, bzw. Änderungen werden automatisch abgewiesen.

4.2 Datendefinitionsanweisungen

4.2.1 CREATE TABLE

Anlegen einer sog. Basistabelle (im Gegensatz zu einem View).

Syntax

> Tabellendef ::= **CREATE TABLE** Tabellenname
> (Spaltendef [, ...])

Spaltendef ::= Spaltenname Datentyp [**NOT NULL**]
Datentyp ::= *siehe folgende Tabelle*

Basisdatentypen von SQL:1999 bzw. SQL:2003 (aus [Türker]):

Typ	Instanz	Beispielwert
BOOLEAN	Wahrheitswert	TRUE
SMALLINT INTEGER BIGINT	ganze Zahl	1704
DECIMAL(p,q) NUMERIC(p,q)	Festkommazahl	1003.44
FLOAT(p) REAL DOUBLE PRECISION	Fließkommazahl	1.5E-4
CHAR(q) VARCHAR(q) CLOB	alphanumerische Zeichenkette	'SQL:1999 + Delta = SQL:2003'
BIT(q) BIT VARYING(q) BLOB	binäre Zeichenkette	B'11011011'
DATE TIME TIMESTAMP INTERVAL	Datum Zeit Zeitstempel Zeitintervall	DATE'1997-06-19' TIME'11:30:49' TIMESTAMP'2002-08-23 14:15:00', INTERVAL '48' HOUR
XML	XML-Wert	<Titel>SQL:2003</Titel>

Hinweise:

- Datenbanksysteme wie MySQL, Oracle, DB2, SQLServer verwenden z. T. andere Bezeichner, manche Datentypen fehlen, andere kommen dazu.
- BOOLEAN fehlt z. B. in Oracle und DB2. MySQL hat eine vom Standard abweichende Interpretation; meist wird BOOLEAN durch CHAR(1) oder bei Oracle NUMBER(1) emuliert.

- Bei den Integer-Varianten gibt der Standard keine Wertebereiche vor, üblich sind aber SMALLINT mit 2, INTEGER mit 4 und BIGINT mit 8 Bytes, Oracle bildet Integer aber auf NUMBER(38) ab, was auf eine BCD-Codierung schließen läßt.
- Bei DECIMAL und NUMERIC spezifiziert *p* die Genauigkeit (Stellen insgesamt), *q* die feste Anzahl der Nachkommastellen; *q* darf weggelassen werden und bedeutet dann $q = 0$, d. h. der Wertebereich umfaßt die ganzen Zahlen. Die zuläßigen Stellenangaben sind implementationsspezifisch. Oracle bildet DECIMAL(*p*,*q*), bzw. NUMERIC(*p*,*q*) auf NUMBER(*p*,*q*) ab und hat einige Besonderheiten, etwa negative Werte für *q* und Werte $q > p$.
- FLOAT, REAL, DOUBLE bilden auf 32, 64 oder 80 Bit Gleitkommazahlen ab. Oracle bildet FLOAT und FLOAT(*p*) auf die eigenen Datentypen BINARY_FLOAT (32 bit) und BINARY_DOUBLE (64 bit) ab. NUMBER ohne Längenangabe repräsentiert in Oracle ebenfalls eine Gleitkommazahl.
- CHAR und CHAR(*q*) sind Zeichenketten fester Länge (kleinere Werte werden mit Leerzeichen gefüllt), die meist direkt im Tupel stehen. CHAR ist gleichwertig zu CHAR(1), *q* ist bei Oracle beschränkt auf 2000, in anderen Systemen oft $q < 255$.
- VARCHAR(*q*) sind Zeichenketten variabler, durch *q* beschränkter Länge. Oracle verwendet den Typbezeichner VARCHAR2(*q*).
- CLOB steht für **C**haracter **L**arge **OB**ject, BLOB für Binary ..., meist 64 KB bis 2 oder 4 GB Größe, oft auch Bezeichnung RAW.
- Jeder Datentyp unterstützt die Datenbank-Null (Nullwert).
- **CREATE TABLE** als Kommando mit Strichpunkt abgeschlossen.
- Namen, in ORACLE z. B. bis 30 Zeichen, müssen mit Buchstaben anfangen, Unterscheidung nach Groß- und Kleinbuchstaben nur wenn in doppelten Anführungszeichen.
- DBMS legt Tabellenbeschreibung im sog. DB-Katalog (Data Dictionary) ab.

Ein recht gründlicher Vergleich der Datenbankimplementierungen (leider ohne Verweis auf numerische Datentypen) findet sich unter
`http://troels.arvin.dk/db/rdbms/`

Beispiel 4–3 Definieren einer Kundentabelle

```
CREATE TABLE Kunden(KDNR SMALLINT,
 KNAME CHAR(30),
 KDADRESSE CHAR(30),
 RABATT NUMERIC(5,1))
```

Hinweis: Rabatt hat 5 Dezimalstellen, davon 1 Nachkommastelle. Abgespeichert und angezeigt (wenn negativ) wird das Vorzeichen, die Position des Dezimalpunkt steckt in der Typdefinition und muß nicht für jeden Wert mitgespeichert werden. Intern kann eine gepackte oder ungepackte BCD-Speicherung verwendet werden.

Übung 4–2

Legen Sie geeignete Tabellen für die Beispiele TLIST, FBNAMEN und für das Lehrer-Schüler-Klassen Szenario an! Verwenden Sie **NOT NULL** für die vorgesehenen Primärschlüssel.

4.2.2 DROP TABLE

Löschen einer Tabelle (Daten und Eintrag im Katalog) mit

Syntax

Tabelle-Löschen ::= **DROP TABLE** Tabellenname

Löscht gleichzeitig alle „abhängigen“ Sichten, Indexe und Synonyme.

4.2.3 CREATE VIEW

Definition einer Sicht (virtuellen Tabelle) aus einer oder mehreren Basistabellen oder anderen Sichten. Erzeugt keine physische Tabelle, Sicht wird zur Abfragezeit erzeugt.

Ziele:

- vereinfachter Zugriff
- Datenunabhängigkeit
- Datenschutz

Syntax

> Sichtdef ::= **CREATE VIEW** Sichtname [(Spaltenname [, ...])]
> **AS** Abfrage [**WITH CHECK OPTION**]

Hinweise:

- unter dem Sichtnamen wird die virtuelle Tabelle angesprochen.
- Spaltennamen können entfallen, dann identisch mit Orginalnamen.
- müssen als *Synonyme* neu vergeben werden bei Namensgleichheit in 2 oder mehr Quelltabellen oder
- wenn Spalten über Ausdrücke gebildet werden (vgl. Spalte `AVG(Gehalt)` unten).
- wenn CHECK-Option gesetzt, wird bei *aktualisierbaren Views* (siehe unten) geprüft, ob geänderte oder neue Zeile überhaupt nach WHERE-Klausel in den View eingefügt oder verändert werden darf.

Beispiel 4–4 Beispiel für CHECK-Klausel

Abweisung der Änderung in View TEL17 unten

```
UPDATE TEL17
  SET FB = 19 WHERE TEL = 4477
```

- Sichten sind aktualisierbar, wenn sie
  - auf genau einer Tabelle (vgl. Hinweis unten) oder
  - genau einer aktualisierbaren Sicht beruhen und
  - nicht mit sich selbst verknüpft sind

  - der Benutzer das Änderungsrecht für die Basistabellen hat
  - die Unterabfrage nicht die DISTINCT-Klausel enthält und auch nicht eine SQL-Formel (vgl. AVG(Gehalt) oben) und keine GROUP BY-Klausel enthält.

- GROUP BY in View-Definition erlaubt, wenn Sicht nur aus einer Tabelle oder Sicht abgeleitet ist
- ORDER BY nicht erlaubt, aber Abfragen mittels View können ORDER BY-Klausel enthalten
- View erleichtert Auswertung komplexer Join-Abfragen, speziell für naive Benutzer.

Hinweis: Oracle erlaubt zwar aktualisierbare Sichten, die auf einem Join beruhen, allerdings dürfen die DML-Anweisungen nur genau eine darunterliegende Tabelle berühren und der Schlüssel der betroffenen Tabelle muß auch im View Schlüsseleigenschaft haben (Oracle: Tabelle ist *key-preserved*).

Bei einem View über den Join von TLIST mit FBNAMEN wäre ein Update oder Insert, obwohl es nur FBNAMEN betrifft, nicht zulässig, weil FBNAMEN nicht „key-preserved“ ist; FB ist Schlüssel in FBNAMEN (jeder Fachbereich hat nur einen Namen), aber im Join ist das Join-Attribut FB nicht Schlüssel, denn ein Fachbereich hat viele Telefonteilnehmer.

Beispiel 4–5 Lokale Telefonnummern des FB 17

```
CREATE VIEW TEL17 AS
 SELECT TEL, FB
 FROM TLIST
 WHERE FB = 17;
```

Bei Abfragen in einem View, oder bei erneutem View wird View-Definition zur Laufzeit erweitert. Kann zu Zurückweisungen der Abfrage führen.

Beispiel 4–6 Zurückweisung einer Abfrage in einem View

(aus dBase Handbuch, S. 6.29)

```
CREATE VIEW Pers1(C1, C2) AS
  SELECT AVG(Gehalt), Einsatz
  FROM Personal
  GROUP BY Einsatz;
```

Abfrage

```
SELECT C1 FROM Pers1;
```

ungültig, weil nach der Erweiterung

```
SELECT AVG(Gehalt) FROM PERSONAL
GROUP BY Einsatz;
```

die Spalte EINSATZ für den Befehl GROUP BY nicht in der SELECT-Klausel enthalten ist. Gültig wären aber

```
SELECT * FROM Pers1;
```

```
SELECT C2 FROM Pers1;
```

Einschränkungen für die Abfrage in der Viewdefinition:

- keine Vereinigung (UNION)
- wenn Abfrage lautet SELECT * FROM ... , dann kann bei Spaltenlöschung oder Erweiterung der Basistabelle der View nicht mehr funktionieren. Dann View löschen und neu definieren.
- ORDER BY, GROUP BY siehe oben
- kein FOR UPDATE OF, INTO, SAVE TO TEMP

Übung 4–3

Mit „+“ als Verkettungsoperator für CHAR-Werte definiere man einen zweispaltigen View über KUNDEN, der Kundennummer und Name (verkettet aus KNAME und KDADRESSE) enthält.

Übung 4–4

Gegeben sei eine Tabelle ARTIKEL mit den Spalten ARTNR, ARTBEZ, LNR, NETTOPREIS. Man definiere einen View mit den Spalten BRUTTOPREIS und FUEPREIS (Preis mit 40% FuE-Rabatt zuzügl. MWSt).

Die Einschränkungen sind meist einleuchtend, wenn die Substitution im View explizit hingeschrieben wird. Z. B. ist ein View nicht aktualisierbar, wenn eine Formel für eine Spalte auftritt.

Man betrachte etwa die Tabelle ANG(NAME, ABT, ALTER, GEHALT, KOMMISSION) und die folgende Viewdefinition:

```
CREATE VIEW EINKOMMEN(ENAME, EABT, GPLUSK) AS
  SELECT NAME, ABT, GEHALT + KOMMISSION
  FROM ANG
  WHERE ALTER > 40
```

Jetzt wollen wir ein kombiniertes Einkommen ändern:

```
UPDATE EINKOMMEN
  SET GPLUSK = 200000
  WHERE ENAME = 'Mueller'
```

Die Abfrage aus der Sichtdefinition eingesetzt in das UPDATE ergibt:

```
UPDATE
     SELECT NAME, ABT, GEHALT + KOMMISSION
     FROM ANG
     WHERE ALTER > 40
  SET GEHALT + KOMMISSION = 200000
  WHERE NAME = 'Mueller'
```

Damit steht aber ein arithmetischer Ausdruck auf der linker Seite der Zuweisung!

Übung 4–5

Wenn in der Sicht die 3. Spalte ohne die Addition nur als GEHALT definiert, würde dann ein UPDATE für Mueller auch das Gehalt eines Angestellten ’Mueller’ mit Alter ≤ 40 Jahren ändern?

4.2.4 DROP VIEW

Löschen der Sicht mit

Syntax

Sichtlöschen ::= **DROP VIEW** Sichtname

wodurch auch alle abhängigen Sichten gelöscht werden.

4.2.5 GRANT

Gewähren von Rechten an Tabellen (entziehen mit **REVOKE**). Wenn implementiert, dann Anmelden bei DBMS mit Paßwort und User-Id (Benutzername).

Syntax

Rechte-Setzen ::= **GRANT** Rechte **ON** Tabelle **TO**
Benutzer [**WITH GRANT OPTION**]

Benutzer ::= Benutzername [, ...] | **PUBLIC**

Rechte ::= Recht [, ...] | **ALL**

Recht ::= **ALTER | DELETE** | **INDEX** | **INSERT** | **SELECT** |
UPDATE [(Spaltenname [, ...])]

Bei WITH GRANT OPTION können eigene Rechte wiederum an andere Benutzer weitergegeben werden.

Beispiel 4–7 Rechte an TLIST und an ARTIKEL

Alle dürfen TLIST abfragen.

```
GRANT SELECT ON TLIST TO PUBLIC
```

Nur Telefondienst darf ändern.

```
GRANT ALL ON TLIST TO TELMASTER
```

Mitarbeiter der Inventur dürfen Menge und Lagerort ändern. Löschen des Artikels oder der Artikelnummer sind nicht erlaubt.

```
GRANT UPDATE(MENGE, LAGERORT) ON ARTIKEL TO INVENTUR
```

Meist weitere Befehle für Paßwörter, Erlaubnis eine Tabelle einzurichten, etc.; normalerweise beschränkt auf den sog. DBA (Datenbankadministrator).

Übung 4–6

Warum genügt es nicht, das DELETE-Recht zu sperren, um eine Zeile zu schützen?

4.3 Datenmanipulationsanweisungen

Zentrales Hilfsmittel ist die SELECT-Klausel, die entweder als eigenständiges Kommando oder Teil anderer Kommandos Tabellen liest und daraus neue Tabellen erzeugt (vgl. View-Definition oben).

4.3.1 SELECT

Syntax

Abfrage ::= (Abfrage) |
Abfrage **UNION** Abfrage |
Abfrage **MINUS** Abfrage |
Abfrage **INTERSECT** Abfrage |
SELECT [**ALL** | **DISTINCT**] Spaltenangabe Tabellenangabe

Spaltenangabe ::= [Tabelle**.**]***** | Spaltenausdruck [**,** ...]
Spaltenausdruck ::= Ausdruck [Alias-Name]

ALL (Standardwert) liefert alle ausgewählten Zeilen, **DISTINCT** unterdrückt Duplikate. Spaltenausdruck wählt anzuzeigende Spalten aus, entweder

- Menge der Spalten aus angegebener Tabelle bei “Tabelle.*“
- Menge der Spalten aus Tabellen der FROM-Klausel bei nur “*“

- eine Spalte je Ausdruck, ansprechbar über danachstehenden (Spalten-)Alias-Namen.
- Ausdruck aufgebaut wie arithmetischer Ausdruck mit
- +, -, * und / als Operatoren und den Operanden
- Spaltennamen, ggf. in der Form Tabellenname.Spaltenname oder Alias-Name.Spaltenname
- geklammerte Ausdrücke
- Funktionsaufrufe mit **DISTINCT** oder **ALL** Spaltenreferenz
- Konstanten
- Tabellen können sein Basistabellen oder Views

Funktionen im Standard SQL sind

- **AVG** (average, Mittelwert)
- **COUNT** (Anzahl der Spaltenwerte), speziell COUNT(*) für Anzahl der Zeilen
- **MAX** (größter Wert der Spalte)
- **MIN** (kleinster Wert der Spalte)
- **SUM** (Summe der Spaltenwerte)

Beispiel 4–8

TLISTE

NAME	FB	TEL
Hans	17	4477
Emil	19	3443
Anna	17	4476
Gabi	?	2441
Klaus	19	3443

FBNAMEN

FB	FBBEZ
17	Math-Inf
19	E-Tech
99	Esoterik

FB	FBBEZ	Anz.:	COUNT(T.TEL)
17	Math-Inf	Anz.:	2
19	E-Tech	Anz.:	2

Gesucht sind FB-Nummer und Bezeichnung aus Tabelle FBNAMEN mit Anzahl der Teilnehmer, die für diesen Fachbereich in TLIST eingetragen sind. Die dritte Spalte enthält eine Textkonstante.

```
SELECT t.FB, n.FBBEZ, 'Anz.: ', COUNT(t.TEL)
 FROM TLIST t, FBNAMEN n
 WHERE t.FB = n.FB
 GROUP BY t.FB, n.FBBEZ
```

Übung 4–7

Teilen sich mehrere Personen ein Telefon wie oben, kann auch die folgende Abfrage sinnvoll sein. Warum?

```
SELECT t.FB, n.FBBEZ, 'Anz.: ',
  COUNT(DISTINCT t.TEL)
  FROM TLIST t, FBNAMEN n
  WHERE t.FB = n.FB
  GROUP BY t.FB, n.FBBEZ
```

Beispiel 4–9 Der klassische Join

Gesucht sind alle Lieferanten, die mindestens einen Artikel der momentanen Artikeltabelle liefern.

```
SELECT DISTINCT LIEFERANTEN.*
  FROM LIEFERANTEN, ARTIKEL
  WHERE LIEFERANTEN.LNR = ARTIKEL.LNR
```

Übung 4–8

Was liefern die folgenden Abfragen?

```
SELECT t.FB, n.FBBEZ
  FROM TLIST t, FBNAMEN n
  WHERE t.FB = n.FB
```

und

```
SELECT DISTINCT t.FB, n.FBBEZ
  FROM TLIST t, FBNAMEN n
  WHERE t.FB = n.FB
```

Der fehlende Teil der Abfrage ist die *Tabellenangabe* mit

- FROM-Teil zur Definition der Eingangstabellen,
- WHERE-Teil zur Selektion der Zeilen der Eingangstabellen
- GROUP-BY-Teil zur Zusammenfassung von Zeilen auf der Basis gleicher Spaltenwerte
- HAVING-Teil zur Selektion nur der Gruppen im GROUP-BY-Teil, die der Bedingung genügen.

Syntax

Tabellenangabe ::= from-Klausel [where-Klausel]
[group-by-Klausel]
[having-Klausel]

from-Klausel ::= **FROM** Tabellenspezifikation [**,** ...]

Tabellenspezifikation ::= Tabellenname [Alias-Name]

where-Klausel ::= **WHERE** Suchbedingung

group-by-Klausel ::= **GROUP BY** Spaltenreferenz [**,** ...]

having-Klausel ::= **HAVING** Suchbedingung

Wie bei den Spaltenausdrücken, sind bei Tabellennamen wieder Synonyme zugelassen (alias, Range-Variablen [Sauer]).

Dabei dient ein Alias meist als Abkürzung, er ist unumgänglich bei Join mit eigener Tabelle:

Beispiel 4–10 Join einer Tabelle mit sich selbst

Finde alle Mitarbeiter, die mehr als ihr Chef verdienen [Sauer, S. 60]]

```
SELECT m1.name, m1.chef, m1.gehalt, m2.gehalt CHEFGEH
  FROM mitarbeiter m1, mitarbeiter m2
  WHERE m1.gehalt > m2.gehalt AND m1.chef = m2.name
```

MITARBEITER

NAME	CHEF	GEHALT
Hans	**Anna**	**3000**
Emil	**Anna**	**5000**
Anna		**4500**
Gabi	**Klaus**	**4000**
Doris	**Klaus**	**2500**
Klaus	**Anna**	**3000**

NAME	CHEF	GEHALT	CHEFGEH
Emil	**Anna**	**5000**	**4500**
Gabi	**Klaus**	**4000**	**3000**

Suchbedingung ist ein Prädikat, d. h. logischer Ausdruck aufgebaut aus Vergleichen, Konstanten, geklammerten Ausdrücken und verbunden mit den Operatoren **NOT**, **AND**, **OR**, und ergibt wahr, falsch oder DB-Null.

Sauer unterscheidet sieben Prädikate ([Sauer, S. 61f]):

direkter Vergleich A.LNR = L.LNR

Intervall	TEL BETWEEN 4000 AND 4999
Ähnlichkeit	NAME LIKE 'H%'
NULL-Test	FB IS NULL
IN	FBBEZ IN ('E-Tech', 'Math-Inf')
ALL/ANY	FBNAMEN.FB > ALL (SELECT FB FROM TLIST)
EXISTS	EXISTS (SELECT * FROM FBNAMEN WHERE FBBEZ = 'INF');

Beispiel für Gebrauch des ALL-Prädikats aus [Sauer], S.62f.

Beispiel 4–11 All-Prädikat

Wer verdient mehr als die Mitarbeiter von Klaus?

```
SELECT * FROM MITARBEITER
  WHERE GEHALT > ALL (SELECT GEHALT
     FROM MITARBEITER
     WHERE CHEF = 'Klaus')
```

Die Unterabfrage ergibt die beiden Gehälter {2500, 4000} für Gabi und Doris. Resultat ist die folgende Tabelle:

NAME	CHEF	GEHALT
Emil	**Anna**	**5000**
Anna		**4500**

Beispiel 4–12

Was liefert die obige Abfrage mit ANY statt ALL?

Das EXISTS-Prädikat liefert wahr gdw. die Unterabfrage mindestens eine Zeile auswählt.

```
SELECT FB, FBBEZ FROM FBNAMEN
  WHERE EXISTS (SELECT * FROM TLIST
    WHERE TLIST.FB = FBNAMEN.FB)
  ORDER BY FB
```

Hinter die äußerste Abfrage darf noch eine order-by-Klausel gestellt werden zur Angabe des Resultats in auf- oder absteigender Sortierordnung.

Syntax

Sortierabfrage ::= Abfrage **ORDER BY** Sortierausdruck [, ...]

Sortierausdruck ::= Ausdruck [**ASC** | **DESC**]

Beispiel 4–13 Alle Telefonteilnehmer nach FB und Namen sortiert.

```
SELECT * FROM TLIST ORDER BY FB, NAME
```

4.3.2 DELETE

Löschen von Zeilen

Syntax

Löschanweisung ::= **DELETE FROM** Tabelle [where-Klausel]

Beispiel 4–14 Löschen

Lösche alle Telefoneinträge deren FB-Angabe keine Entsprechung in der Liste der Fachbereichsnamen hat.

```
DELETE FROM TLIST
  WHERE NOT EXISTS
    (SELECT *
      FROM FBNAMEN
      WHERE TLIST.FB = FBNAMEN.FB)
```

4.3.3 UPDATE

Ändern von Zeilen

Syntax

Änderungsanweisung ::= **UPDATE** Tabelle
SET Wertzuweisung [**,** ...] [where-Klausel]

Wertzuweisung ::= Spaltenreferenz **=** Skalarausdruck |
Spaltenreferenz **= NULL**

Beispiel 4–15 Update

Hans zieht in Fachbereich ’Informatik’ (FB 25) um und bekommt eine neue Nummer.

```
UPDATE TLIST SET FB = 25, TEL = 5025
  WHERE NAME = 'Hans'
```

Übung 4–9

Fügen Sie den neuen Fachbereich 25 in FBNAMEN ein!

4.3.4 INSERT

Anfügen von Zeilen. Nichtbenannte Spalten werden mit DB-Null gefüllt. Quelle der Daten sind entweder Konstanten oder Daten aus anderen Tabellen, die mittels SELECT ausgewählt werden.

Syntax

Einfügeanweisung ::= **INSERT INTO** Tabelle
[**(** Spaltenangabe [**,** ...] **)**] Quellangabe

Quellangabe ::= **VALUES(** Wert [**,** ...] **)** | Abfrage

Wert ::= Konstante | **NULL**

Beispiel 4–16 Insert ... Values

Rumpelstilzchen tritt dem Fachbereich 17 bei, hat aber noch kein Telefon.

```
INSERT INTO TLIST (NAME, FB)
 VALUES('Rumpelstilzchen', 17)
```

Beispiel 4–17 Insert ... Select

Übernehmen aller FB-Nummern aus TLIST.

```
INSERT INTO FBNAMEN (FB)
  SELECT TLIST.FB
    FROM TLIST
```

Will man nur die unbekannten Nummern übernehmen, müßte man vorher eine entsprechende Sicht

```
... WHERE TLIST.FB <> ALL(SELECT FB FROM FBNAMEN)
```

anlegen. Innerhalb des INSERT ist, wie auch bei UPDATE und DELETE, die Selektion über die gerade zu ändernde Tabelle eigentlich nicht erlaubt, d.h.

```
INSERT INTO FBNAMEN (FB)
  SELECT t.FB FROM TLIST t
    WHERE t.FB <> ALL(SELECT n.FB FROM FBNAMEN n).
```

ist nicht zugelassen, wird aber z. B. von ORACLE akzeptiert.

Insert, Delete und Update im interaktiven Editieren viel zu umständlich. Fast alle Systeme bieten dafür einen visuellen Editor.

4.4 Weitere SQL-Entwicklungen

aus [Shaw90]

- SQL-86: ungefähr 100 Seiten
- SQL-89 ungefähr 115 Seiten
- SQL2 Entwurf ungefähr 450 Seiten
- SQL3 Entwurf ungefähr 700 Seiten

SQL2 aufgeteilt in drei Teilmengen:

- Entry SQL2 – geringe Änderungen
- Intermediate SQL2 – rund 50 % der neuen Merkmale
- Full SQL2 – alles! Jahre der Implementationsarbeit ...

4.4.1 Entry SQL2

- bisher Resultatsvariable SQLCODE mit „Erfolg“, „nicht gefunden“ und „Fehler“, implementationsabhängiger Fehlercode, besser genormter SQLSTATE
- Ersatz der Numerierung von Spalten durch AS-Klausel, also statt

```
SELECT NAME, SALARY + COMMISSION
FROM EMPLOYEE
ORDER BY NAME, 2
```

  besser

```
SELECT NAME AS EMP_NAME, SALARY + COMMISSION AS PAY
FROM EMPLOYEE
ORDER BY EMP_NAME, PAY
```

- Fluchtsymbole für SQL-Schlüsselwörter als Benutzernamen
- Sprachinterface für Ada und C
- einige Korrekturen zu SQL-89

4.4.2 Intermediate SQL2

- Neue Datentypen DATE, TIME, TIMESTAMP und Operationen darauf; Intervalle YEAR, MONTH, ...
- Klausel ON DELETE CASCADE (vgl. Abschnitt 3.4: Fremdschlüssel und Integrität)
- Domains als Macro für eine Art Benutzerdatentyp, z. B.

```
CREATE DOMAIN MONEY IS DECIMAL(5, 2)
   DEFAULT (-1)
   CHECK( VALUE = -1 OR VALUE > 0 )
   NOT NULL

CREATE TABLE EMP (NAME CHAR(5),
   SALARY MONEY)
```

- Erweiterter Zeichensatz, inkl. 2-Byte-Zeichensätze
- bessere Sortierordnungen (Schlüsselwort COLLATE)
- Outer Join (Schlüsselworte `LEFT JOIN`, `RIGHT JOIN`, `FULL JOIN`), z. B.

```
TLIST LEFT JOIN FBNAMEN ON (TLIST.FB = FBNAMEN.FB)
```

- Syntax für natürlichen Join

```
TLIST NATURAL LEFT JOIN FBNAMEN
```

- Cursor „Scrolling“, z. B. für gegebenen Cursor EMP_CURSOR

```
FETCH RELATIVE +3 FROM EMP_CURSOR
FETCH ABSOLUTE 5 FROM EMP_CURSOR {5. Zeile von oben}
FETCH LAST FROM EMP_CURSOR
```

- Zeilenausdrücke, d. h. geklammerte Wertetupel, z. B.

```
... WHERE (E1.ALTER, E1.GEHALT) >>
(E2.ALTER, E2.GEHALT)
```

statt

```
... WHERE E1.ALTER > E2.ALTER OR
(E1.ALTER = E2.ALTER AND E1.GEHALT > E2.GEHALT)
```

- Operationen auf Zeichenketten, CASE-Ausdrücke, COALESCE (erster von NULL verschiedener Wert in einer Liste, vgl. NVL-Funktion in Abschnitt 3.2), NULLIF

```
COALESCE (EMPLOYEE.ALTER, 'ALTER UNBEKANNT')
```

statt

```
CASE WHEN EMPLOYEE.ALTER IS NULL
THEN 'ALTER UNBEKANNT' ELSE EMPLOYEE.ALTER END
```

und

```
NULLIF (EMPLOYEE.SALARY, -1)
```

statt

```
CASE WHEN EMPLOYEE.SALARY = -1
THEN NULL ELSE EMPLOYEE.SALARY END
```

- Type Casting (explizite Typwandlungen)
- CHARACTER VARYING mit Angabe max. Länge und Length-Funktion (Abkürzung VARCHAR, Oracle VARCHAR2)
- Einschränkungen in Views (z. B. keine UNION) entfallen fast alle

- Ersatz eines Skalarwerts durch Abfrage mit skalarem Resultat fast überall möglich
- `GET DIAGNOSTICS` (vgl. SQLSTATE)
- `SET AUTHORIZATION` für laufende Anwendungen
- `PREPARE` und `EXECUTE` für online, interaktive SQL-Abfragen (Standard in praktisch allen kommerziellen Versionen)
- Schemamanipulationen (ALTER, DROP)
- CATALOG-Tabellen standardisiert für Tabellen, Spalten, Views
- sog. CONSTRAINT-Namen (Bezeichner für Attribute mit Einschränkungen), z. B.

```
CREATE TABLE EMPLOYEE
  (NUMBER CHARACTER(5) CONSTRAINT EMP_KEY
    PRIMARY KEY,
  NAME CHARACTER(10),
  DEPT_ID CHARACTER(5) CONSTRAINT DEPT_REF
    REFERENCES DEPARTMENT,
  DRIVERS_LICENSE CHARACTER(10) CONSTRAINT
    EMP_UKEY UNIQUE,
  SALARY FIXED(5, 2) CONSTRAINT SAL_CHECK
    CHECK(SALARY > 0))
```

  dann mit ALTER Einschränkungen aufheben, bzw. temporär aussetzen, usw.

- Read-only Transaktionen
- separate Compilierung

4.4.3 Full SQL2

- mehr Funktionen für TIME-Datentyp
- mehr zu CATALOG-Tabelle, inkl. Integrität
- temporäre Tabellen mit Option, sie über ein COMMIT zu halten
- BIT-Datentyp für binäre Daten wie z. B. Pixel-Bilder
- Ausrichtung von Spalten für Verträglichkeit in UNION, EXCEPT, INTERSECT (sog. CORRESPONDING-Klausel)

- UNION JOIN um folgende Situation zu lösen: zwei Tabellen mit Gehalts- und Ausbildungsinformation mit mehreren Zeilen je Angestellten

```
SALARY_HISTORY(EMP_ID, DATE, SALARY)
EDU_HISTORY(EMP_ID, COURSE, GRADE, YEAR, MONTH)
```

Gesucht ist Auflistung der Gehalts- und dahinter der Ausbildungshistorie jedes/r Angestellten; geht nicht mit UNION, da unterschiedliche Attribute, JOIN paßt nicht, da Gehalt nichts mit Ausbildungstupel zu tun hat. Idee ist, fehlende Attribute mit Nullen zu füllen, so daß

```
SALARY_HISTORY UNION JOIN EDU_HISTORY
```

die folgende Art von Tabelle erzeugt

SALARY_HISTORY	Nullspalten
Nullspalten	EDU_HISTORY

wobei eine Sortierung mit ORDER BY sich üblicherweise anschließt.

```
SELECT COALESCE(SALARY_HISTORY.EMP_ID,
   EDU_HISTORY.EMP_ID) AS EMPID,
   SALARY_HISTORY.*, EDU_HISTORY.*
FROM SALARY_HISTORY UNION JOIN EDU_HISTORY
ORDER BY EMPID
```

COALESCE sorgt dafür, daß der Nicht-Null Wert aus den zwei gleichnamigen Spalten EMP_ID genommen wird, mit Alias EMPID bezeichnet.

- ON UPDATE CASCADE (vgl. ON DELETE CASCADE)
- MATCH-Klausel und MATCH-Prädikat für Beziehungen auf mehreren Spalten, z. B. Schlüssel über 2 Spalten

```
CREATE TABLE DEPARTMENT
   (LOC CHARACTER(20),
   ID CHARACTER(5),
   NAME CHARACTER(10),
   PRIMARY KEY (LOC, ID))
```

weder `LOC` noch `ID` haben UNIQUE-Klausel! Jetzt referentielle Integrität über mehrere Spalten

```
CREATE TABLE EMPLOYEE
   (NUMBER CHAR(5),
   ELOC CHAR(5),
   DEPT_ID CHAR(5),
   FOREIGN KEY (ELOC, DEPT_ID) REFERENCES
      DEPARTMENT(LOC, ID))
```

Einfügen oder Ändern einer Angestelltenzeile nur, wenn entweder ELOC oder DEPT_ID ungleich DB-Null und der Wert ungleich DB-Null existiert in DEPARTMENT.

- geschachteltes SELECT in FROM-Klausel (in-line View)
- Sichtenmaterialisierung (View einer View mit berechneten und umsortierten Spalten und Zeilen)
- Tabellenkonstanten
- SET SCHEMA Anweisung erzeugt Gültigkeitsbereich (Scope) für Namen
- Unterabfragen in CHECK-Klausel, z. B.

```
CREATE TABLE LKW
   (GROESSE INTEGER,
   KFZNR CHAR(5) PRIMARY KEY)

CREATE TABLE PKW
   (MODEL CHAR(10),
   KFZNR CHAR(5) PRIMARY KEY,
   CHECK(NOT EXISTS SELECT * FROM LKW
      WHERE PKW.KFZNR = LKW.KFZNR))
```

verlangt, daß nach jeder Änderung geprüft wird, daher kann Einschränkung (engl. *constraint*) an Tabelle LKW oder PKW hängen.

- daher auch „freistehende“ Zusicherungen (engl. *assertions*), z. B.

```
CREATE ASSERTION KFZNR_REGEL
   CHECK(NOT EXISTS SELECT * FROM PKW, LKW
      WHERE PKW.KFZNR = LKW.KFZNR)
```

- An- und Abschalten von Zusicherungen

```
SET CONSTRAINTS OFF
... andere SQL-Anweisungen
SET CONSTRAINTS ON
```

bei COMMIT immer Überprüfung der Zusicherungen

- in SQL-89 nur eine DISTINCT-Klausel je Abfrage erlaubt, jetzt

```
SELECT AVG(DISTINCT SALARY),
   SUM(DISTINCT COMMISSION)
FROM EMPLOYEE
```

- SQL-89 definiert INSERT-Privileg nur für alle Spalten zusammen, jetzt spaltenweise

```
GRANT INSERT (NAME, NUMBER) ON EMPLOYEE TO 'SMITH'
```

- Selbstreferenzierendes UPDATE und DELETE, in SQL-89 WHERE-Klausel kann sich nicht auf zu entfernende oder zu ändernde Tabelle beziehen, jetzt

```
UPDATE EMPLOYEE SET SALARY = SALARY * 1.1
WHERE EMPLOYEE.SALARY < SELECT AVG(SALARY)
 FROM EMPLOYEE E2 WHERE EMPLOYEE.NUMBER <> E2.NUMBER
 AND EMPLOYEE.DEPT_ID = E2.DEPT_ID
```

erhöhe Gehalt aller Angestellten, deren Gehalt kleiner dem Durchschnittsgehalt aller anderen Angestellten der selben Abteilung ist.

```
DELETE FROM EMPLOYEE
WHERE EMPLOYEE.SALARY >= ALL SELECT E2.SALARY
 FROM EMPLOYEE E2
 WHERE EMPLOYEE.NUMBER <> E2.NUMBER
 AND EMPLOYEE.JOBCODE = E2.JOBCODE
```

Entferne alle Angestellten, deren Gehalt gleich oder höher als das Gehalt aller **anderen** Angestellten mit dem selben Jobcode ist. Hat zur Folge, daß „Vorher-“ und „Nachher-Kopien“ der Tabelle gehalten werden müssen!

Übung 4–10

Prüfen Sie die oben angegebenen Erweiterungen an Ihrem Lieblingsdatenbanksystem! Was gibt es, was geht anders?

4.4.4 SQL3 (SQL-99)

Die folgende Übersicht wurde dem Vortrag von Peter Pistor am 3.2.2000 in Jena entnommen.

Die Antwort: SQL-99

- SQL-99: de-jure-Standard (wie SQL-92)
- 5-teiliger Standard; 07/99 verabschiedet
- Weitere Teile:
  - OLAP (Verdichtung/Analyse von Daten)
  - MED (DBMS als Integrations-Vehikel)
  - OLB (Einbettung in Java)
- SQL/MM:
  - Flankierende Anwendungspakete für Bild-Banken IR, GIS
  - Portabilität
  - Laufzeitverhalten

Typsystem von SQL-99

- Vordefinierte Datentypen; neu:
  - BOOLEAN
  - CLOB, BLOB
- „constructed types“
  - Arrays
  - Referenz-Typen
  - Lokatoren
- Benutzerdefinierte Typen
  - „distinct types“: basieren auf vordefinierten Datentypen, Verkapselung
  - „structured types“ (SDTs)

StrukturierteTypen

- 00-Paradigma
- Attribute (alle Typen)
- Methoden („message“ -Prinzip; Redefinition)
- Verkapselung („observer, mutator methods“)
- Definierbare Ordnungsrelationen
- Definierbare „Transformationen“ (Serialisierung)
- Einfache Vererbung: Typ-Hierarchien
- Substituierbarkeit
- Abstrakte und instanziierbare Typen
- SDTs wie „normale“ Typen verwendbar
- Tabellen-Hierarchien („Tupel-Typen“)
- Referenz-Typen
- Privilegien: UNDER, USAGE

StrukturierteTypen: Beispiel 1

- Abstrakter Typ
  Erlaubt Modellierung gemeinsamer Konzepte diverser Subtypen ohne künstliche Supertyp-Instanzen

```
CREATE TYPE ST_Geometry AS
   (ST_PrivateGeometry SMALLINT,
   ST_PrivateGeometryDimension SMALLINT)
   NOT INSTANTIABLE NOT FINAL
```

- Instanziierbarer Typ
  Erbt Attribute und Methoden vom Supertyp (abstrakt oder instanziierbar)

```
CREATE TYPE ST_Point UNDER ST_Geometry AS
   (ST_PrivateX DOUBLE PRECISION,
   ST_PrivateY DOUBLE PRECISION)
   INSTANTIABLE NOT FINAL
```

Strukturierte Typen: Beispiel 2

- Instanziierbarer Wurzel-Typ

```
CREATE TYPE person AS
  (SSID INTEGER,
   Name VARCHAR(20))
  INSTANTIABLE NOT FINAL
```

- Instanziierbare Sub-Typen

```
CREATE TYPE employee UNDER person AS
  (PID INTEGER,
   Salary INTEGER,
   Manager REF employee)
  INSTANIABLE NOT FINAL

CREATE TYPE manager UNDER employee AS
  (Bonus INTEGER)
  INSTANTIABLE NOT FINAL
```

Substituierbarkeit; typspezifisches Verhalten

- Beispiel:

```
CREATE TABLE Projects
  (PrID INTEGER,
   works-for employee,
   PrLeader employee)

SELECT p.works-for.JahresGehalt() FROM Projects p
```

- Statischer Typ von *works-for: employee*
- Laufzeit-Typ: *employee* oder *manager*
- Aufruf von Methoden (siehe weiter unten) des jeweils aktuellen Typs („dynamic dispatch“)
- Navigation im Typbaum möglich (aber nur in Ausnahmefällen erforderlich)

```
.. (TREAT(p.works-for AS manager)).Salary
.. (p.works-for AS person).SSID
```

Routinen („SQL-invoked routines“)

- Prozeduren und reguläre Funktionen
  - Auswahl über statische Parametertypen
  - „static dispatch“
  - Überladbar

  - Aufrufbeispiele: `CALL myProc; myFunc()`
- Instanz-Methoden
  - Funktionen; spezielle Aufrufkonventionen
  - Überladbar; redefinierbar („override“)
  - „dynamlc dispatch“; Laufzeit-Funktion von „SELF“ bestimmt
  - Aufrufbeispiel: `p.JahresGehalt()`
- Statische Methoden (Zugeständnis an Java)
  - Aufrufbeispiel: `person::TypeInfo()`
- Privilegien: EXECUTE

Realisierung von Routinen

- Über Wirtssprachen
  - Funktions/Prozedur-Kopf in SQL
    - Name, Parameter, ggf. Ergebnis-Typ
    - Protokoll-Definitionen
  - Rumpf in einer 3GL-Sprache
  - Ausnutzung bestehender Bibliotheken
  - Ggf. Laufzeit-Vorteile
- In SQL
  - Neue imperative Sprachkonstrukte
    - Kontrollfluß-Strukturen; Blöcke
    - Zuweisungen
    - Behandlung von Ausnahmebedingungen
  - Beispiel: siehe unten

Tabellen mit referenzierbaren Tupeln

- „Getypte Tabellen“; Basis: strukturierte Typen
- Objekt-Identität à la SQL-99
- Jedes Attribut entspricht einer Spalte
- Zeilen über Referenz-Werte ansprechbar, falls Tabelle entsprechend deklariert (siehe weiter unten)

- Lesen, UPDATE, INSERT: Impliziter Aufruf von Beobachter-, Mutator-, Konstruktor-Methoden
- Spalten vom Typ REF: Zieltabelle über SCOPE-Konstrukt festlegbar
- Beispiel: siehe unten

Routinen-Deklarationen

- Beispiel 1: Methode für *employee*

```
CREATE JahresGehalt()
RETURNS INTEGER
FOR employee
RETURN (13*Salary)
```

- Beispiel 2: Methode für *manager*

```
CREATE JahresGehalt()
RETURNS INTEGER
FOR manager
BEGIN
  DECLARE result INTEGER;
  SET result = (SELF AS employee).JahresGehalt();
  RETURN(result + SELF.Bonus);
END
```

Getypte Tabellen: Beispiel

- Tabellen-Definition

```
CREATE TABLE Employees OF employee
REF IS RefCol SYSTEM GENERATED
Manager WITH OPTIONS SCOPE Employees
```

- Anfrage: Pfadausdrücke statt Joins

```
SELECT e.Name, e.Manager->Name AS Chef
FROM Employees e
WHERE e.Salary < 1000
```

Tabellen-Hierarchien

- Basis: getypte Tabellen
- Maximal eine Supertabelle je Tabelle
- Zeile einer Subtabelle auch Zeile aller Supertabellen

- Eine Typ-Hierarchie: n Tabellen-Hierarchien
- Zugriff über Supertabelle: Sicht dieser Tabelle
- Zeilen von Subtabellen bei Bedarf ausblendbar

```
CREATE TABLE People OF person
CREATE TABLE Employees OF employee UNDER People
CREATE TABLE Managers OF manager UNDER Employees

SELECT * FROM People -- alle Zeilen
                     -- nur person-Spalten

SELECT * FROM ONLY(Employees)
                     -- nur employee-Zeilen
                     -- und Spalten
```

Genestete (d. h. „NF2“)Tabellen

- Sehr dürftig
- Möglich über Spalten vom Typ ARRAY
- Anfrage-Möglichkeiten:
  - Implizite Umwandlung ARRAY --> Tabelle
  - Optionale Index-Spalte
  - (eingeschränkte) Nestbarkeit von SFW
- Beispiel:

```
CREATE TABLE Projekte
   (PrId INTEGER,
    Team REF employee ARRAY[10]
    PrLeiter REF employee)

SELECT p.PrId, CARDINALITY(p.Team) AS T_Anz
FROM Projekte p
```

Schema-Evolution

- Schon immer:
  - Zufügen/Löschen von Tabellen und von Spalten
  - Zufügen/Löschen von Domänen
  - Zufügen/Löschen von Konsistenz-Bedingungen
- Neu:

  - Zufügen/Löschen von Routinen
  - Zufügen/Löschen von Typen
  - Zur Anpassung von Typ- und Routinen-Paketen möglich:
    - Ändern von (externen) Routinen
      -- Implementierung; Charakteristiken
    - Ändern von Typen
      -- Zufügen/Löschen von Attributen
      -- Zufügen/Löschen von Methoden

Rekursive Anfragen

- Beispiel (lineare direkte Rekursion):

```
WITH RECURSIVE Ahnen (Kind, Ahne) AS
  (SELECT Kind, Elter FROM Eltern WHERE ...
  UNION ALL
  SELECT a.Kind, e.Elter
  FROM Ahnen a, Eltern e
  WHERE a.Elter = e.Kind)
SELECT * FROM Ahnen
```

- Lineare Rekursion; verschränkte Rekursion
- Optionale „breadth/depth first“ Markierung
- Optionaler Abbruch bei Erkennung von Zyklen
- Vorbild: Datalog; SQL-Anpassung nicht-trivial
  - Unterstützung von Duplikaten
  - Beschränkungen zur Sicherung der Stratifikation:
    - Gruppierung; „outer join“
    - Mengen-Differenz; Mengen-Durchschnitt

Trigger

- Auslöser: UPDATE, INSERT. DELETE (nicht: SELECT)
- Zeitpunkt: BEFORE, AFTER (nicht: INSTEAD)
- Granularität: FOR EACH STATEMENT / ROW
- Zugriff auf Vor- und Nachzustand
- Bedingungsgesteuerte Aktionen; Kaskadierung möglich
- Nur für Basis-Tabellen (nicht: Sichten)

- Trigger-Ordnung: Zeit der Definition
- Ergänzung des Integritäts-Subsystems
  - AFTER Trigger:
    - Ausführung der Aktionen nach vollständiger Ausführung der auslösenden Anweisung (einschl. RI-Aktionen); Aktionen sehen konsistenten DB-Zustand
    - Einsatz zur Sicherung von Geschäftsregeln
  - BEFORE Trigger:
    - Starke Beschränkungen; verhindern Erzeugung inkonsistenter Zustände durch Trigger-Aktionen
    - Vorzugsweise zur Validierung von Eingabedaten und Generierung bestimmter Feldwerte

Anbindung an Wirts-Sprachen

- Eingebettetes statisches SQL
  - Gutes Laufzeitverhalten
  - Passable Schnittstelle
- Dynamisches SQL
  - Klassischer Ansatz
  - CLI: CALL-Schnittstelle („ODBC++“)
  - Binäre Portabilität auf Kosten des Laufzeitverhaltens
- Problem: Abbildung von Datentypen
  - Vordefiniert für Basis-Typen, „constructed types“
  - SDTs: definierbare „Transformationen“ (Abbildungen zwischen SDTs und String-Typen)
- Java-Anbindung
- SQLJ in ISO-Fassung (siehe unten)

SQL/OLB: SQL-Einbettung in Java

- Hintergrund
  - DB-Zugang prinzipiell gelöst: JDBC
    - Schnittstellen: ODBC; DBMS-C/S-Protokolle

    - Aber: umständlicher als eingebettetes SQL
    - Aber: i. A. schlechtes Laufzeitverhalten
  – Antwort: de facto Standard SQLJ
    - Teil 0: Einbettung von SQL-Anweisungen in Java
    - Teil 1: Nutzung von Java-Methoden in SQL
    - Teil 2: Manipulation von Java-Objekten in SQL
- SQL/OLB entspricht SQLJ Teil 0
  – Einfache Programmierung
  – Potentiell gutes Laufzeitverhalten
  – Trotzdem: binäre Portabilität

SQL/MED: Verwaltung externer Daten

- Einbindung externer Dateien über *DATALINK*s
  – Angereicherte URLs
  – Spalten-Optionen für:
    - Recovery
    - Zugangs-Kontrolle
    - Integritäts-Überwachung
  – Anwendung (z. B.): Verwaltung von Bild-Archiven
- Maskierung externer Daten als SQL-Tabellen:
  – „wrappers“: Satz von 3GL-Routinen
  – „foreign server“: definiert über einen „wrapper“
  – „foreign table“: definiert über einen „foreign server“
  – Fremde Daten: Dateien, Mess-Fühler, Datenbanken
  – Fremde Datenbanken: Föderation

SQL-99: Was erreicht wurde

- Kompatible Erweiterung von SQL-92
- Neue Basis-Typen, speziell für MM-Anwendungen
- Erweiterbares Typ-System
- Kollektionen (leider nur Ansätze)

- Benutzerdefinierbare Routinen
- Berechnungs-Vollständigkeit
- Objekt-Orientierung
- Grundlagen für aktive Datenbanken
- Rekursive Anfragen
- Verbesserte Integritätsüberwachung
- Verbesserte Sprach-Anbindung (teilweise in Arbeit)
- Integration externer Daten (in Arbeit)

SQL-99: Ausblick und Kritik

- Umfang des „Kerns“; es fehlen z. B.:
  - Trigger; strukturierte Typen; Namens-Überladung
- Mangelhafte Orthogonalität
  - Kollektionen (auch: XML); Tabellen: Exoten im Typsystem
  - Objekt-Identität (nur bei Tabellen; Probleme bei Nestung)
  - Autorisierung und Routinen: „invokers’ rights“
- Typ-System:
  - Typ-Migration (z. B.: *person ==> employee*)
  - Bessere Trennung lnterface-lmplementierung
  - PUBLIC/PRIVATE/PROTECTED
    (z. Z. nur EXECUTE-Privileg)
- Temporale DBMS
- Ambivalent: Dominanz der Anbieter
  - Umfang des Kerns
  - Test-Suiten; Wartung des Standards
  - Neue Funktionalität

4.4.5 SQL:2003

Der SQL:2003 Standard liegt in 9 Teilen vor und enthält über 600 neue Features, die fast alles in SQL:1999 in Kleinigkeiten berühren. Daneben

bringt er aber wenige große Neuerungen (vgl. [KW07]). Zu letzteren zählen:

- Identitätsspalten (identity columns) mit automatisch hochzählenden Werten (vgl. MS Access „AutoWert“)
- Auto-generierte Spalten, deren Werte sich aus anderen Spalten automatisch berechnen
- Generierte Sequenzen (Sequenzgeneratoren, die Werte bei jedem Aufruf mit definiertem Increment hochzählen); verwendbar über Tabellen hinweg
- Eine MERGE-Anweisung, die das Zusammenführen zweier Tabellen mit Sonderbehandlung schon vorhandener Datensätze erlaubt
- Erweiterungen der CREATE TABLE Anweisung, jetzt möglich "CREATE TABLE AS" und "CREATE TABLE LIKE"
- Entfernen der schlecht implementierten Datentypen BIT und BIT VARYING
- Einführung von Multimengen mit einer Reihe interessanter Funktionen, u. a. UNNEST (vgl. die Anmerkungen am Ende von Abschnitt 3.2 zu Multimengen und Ende von Abschnitt 3.5 zu UNNEST)
- Tabellenwertige Funktionen:
  SELECT a, b FROM TabellenFunktion(Parameter)
- SQL/XML als Verbindung von XML mit SQL.

Von den ursprünglich 14 geplanten Teilen sind einige weggefallen oder in andere aufgenommen worden. Übriggeblieben sind (auf insgesamt 3606 Seiten):

- Part 1: Framework (SQL/Framework)
- Part 2: Foundation (SQL/Foundation)
- Part 3: Call-Level Interface (SQL/CLI)
- Part 4: Persistent Stored Modules (SQL/PSM)
- Part 9: Management of External Data (SQL/MED)
- Part 10: Object Language Bindings (SQL/OLB)
- Part 11: Information and Definition Schemas (SQL/Schemata)

- Part 13: SQL Routines and Types Using the Java Programming Language (SQL/JRT)
- Part 14: XML-Related Specifications (SQL/XML)

Aus dem zukünftigen Standard 200*n* hat man einen Teil, der nur die Zusammenarbeit von SQL und XML (speziell XQuery) betrifft, als Standard SQL:2006 ausgegliedert. Wir gehen hierauf nicht ein.

4.5 Query-by-Example

Abfragen über direkte Manipulation, IBM Entwicklung [Zloof77], recht nett in dBase realisiert.

Idee: Abfragebedingungen und Operationen in Tabellenskelett eintragen.

Relationenname	***Attributname***	***Attributname***	***Attributname***
Tupelkommandos	*Bedingungen*	*durch Konstanten*	*und Variablen*

Beispiel 4–18 QBE für Telefonnummern > 3999

Tabelle der Telefonnummern > 3999 mit Name, Tel-Nr., FB und FB-Bezeichnung

TLIST	**NAME**	**FB**	**TEL**
		V1	>3999

FBNAMEN	**FB**	**FBBEZ**
	V1	

- Im Beispiel ist V1 eine *Kopplungsvariable* (frei gewählter Bezeichner, im Orginalartikel *example* genannt)
- Bezeichner muß im original QBE mit Unterstrich anfangen

- Selektionsprädikat für TEL-Spalte ist > 3999
- Operationsfeld kann leer bleiben in dBase
- QBE: **P.** für Print
- Alle Operatoren mit Punkt abgeschlossen

Beispiel 4–19 QBE mit Konjunktion

Eine „und-Abfrage“ mit FB = 17 und TEL keine 4000er Nummer, ausgedrückt als Disjunktion (FB = 17 $\wedge$ TEL > 4999) $\vee$ (FB = 17 $\wedge$ FB < 4000).

TLIST	NAME	FB	TEL
		17	>4999
		17	<4000

Beispiel 4–20 QBE mit Disjunktion

Eine „oder-Abfrage“ mit FB 17 oder 4000er Nummer (Komma in einer Spalte immer als Konjunktion).

TLIST	NAME	FB	TEL
		17	
			<5000, >3999

Beispiel 4–21 Update in QBE

Alle FB 17 Nummern um 1000 erhöhen. In dBase mit Replace-Operator, in QBE ursprünglich **U.** für Update. Zu ändernde Spalte durch WITH

markiert in dBase, damit Selektionskriterium 17 ohne WITH in der selben Zeile möglich.

TLIST	NAME	FB	TEL
Replace		17	WITH TEL + 1000

Weitere Bedingungen angebbar über sog. „Bedingungsboxen“, im ersten Beispiel etwa zusätzlich

Bedingungen
V1 <> 17

Tabellen und Views anlegen in QBE durch Einfügeoperationen auf dem Katalog(!), in dBase menügesteuert. Tabellendefinition enthält Vergabe eines Namens für den Wertebereich (Typnamen) – nur original QBE.

Übung 4–11

Gesucht sind alle Teilnehmer aus TLIST mit gleicher Telefonnummer. Läßt sich in dBase als Sicht über zweimal die Tabelle TLIST und Bedingungsbox XVar <> YVar realisieren. Geben Sie die Abfrage an und prüfen Sie das Resultat!

5 Entwurf relationaler Datenbanken

5.1 Abhängigkeiten

Dieser Abschnitt greift die Diskussion aus Abschnitt 3.4 (*Fremdschlüssel* und *Integrität*) wieder auf; z. B. wenn in den Relationen

> BESTPOSTEN(BESTNR, DATUM, KDNR, POS, ARTNR, MENGE)
>
> KUNDEN(KDNR, KNAME, STADT)

KDNR *Primärschlüssel* in KUNDEN und *Fremdschlüssel* in BESTPOSTEN ist. BESTPOSTEN hänge ferner existentiell von KUNDEN ab.

Ziel ist nun die Vermeidung hängender Tupel (z.B. Bestellungen ohne Kunden) und anderer *Anomalien* durch Update-/Einfüge-/Entferne-Operationen.

Anomalien entstehen durch *Abhängigkeiten* und lassen sich vermeiden durch *Redundanzfreiheit*, z. B. tritt KDNR oben in allen BESTPOSTEN-Tupeln mit POS = 1, ..., *n* auf. Die Anomalie entsteht durch zwei unterschiedliche KDNR-Werte bei gleicher BESTNR.

Ähnlich sind zwei unterschiedliche ARTNR bei gleicher BESTNR und POS oder zwei unterschiedliche Kundennamen für die gleiche Kundennummer in Relation KUNDEN zu beurteilen.

Abhängigkeiten existieren also, wenn man Werte „raten“ kann, z. B. KDNR *x* unten:

BESTNR bestimmt KDNR

BESTNR	POS	KDNR
4711	1	100
4711	2	x
4712	1	200

Die Methode zur Vermeidung der Anomalien ist die *Normalisierung* der Schemata gemäß der Regeln der 1. - 5. Normalform (NF) und der *Boyce-Codd* (*BCNF*) *Normalform*, die strikt hierarchisch aufeinander aufbauen (vgl. Abbildung unten). Für die Praxis relevant sind 3. NF und BCNF.

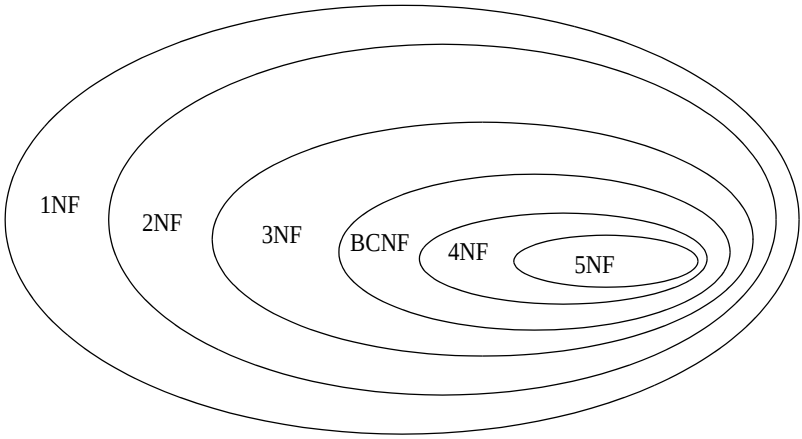


Abb. 5–1 *Hierarchische Anordnung der Normalformen*

Abhängigkeiten von Attributen werden unterteilt in

- funktional abhängig
- voll funktional abhängig
- transitiv abhängig
- mehrwertig abhängig

Beispiel 5–1

Eine schlecht entworfene Hochschul-Datenbank mit Fachbereichen (F) und Mitgliedern (M), deren Schlüssel NAME ist. F-M hat den kombinierten Schlüssel NAME-FB, da Mitglieder mehreren Fachbereichen angehören können (1. und 2. Mitgliedschaft). INTEGR ist ein Boolscher Wert für „Integrierten Studiengang“.

F

FB	BEZ	INTEGR
17	Math-Inf	F
7	WiWi	T
19	E-Tech	T

M

NAME	ORT	STATUS
Hans	Hopla	Prof
Emil	WAllee	WB
Anna	AVZ	Prof
Gabi	Hopla	VA
Rolf	Hopla	WB

F-M

NAME	FB	TEL	MITGLIED
Hans	17	4477	1
Hans	7	4477	2
Emil	19	3344	1
Gabi	?	2233	0
Anna	17	4476	1

Definition 5–1

Seien A und B Attributmengen des Relationenschemas R, d. h. $A \subseteq R$ und $B \subseteq R$. Wir sagen B ist von A *funktional abhängig*, kurz $A \rightarrow B$, wenn zu jedem Wert in A genau ein Wert in B gehört. Etwas formaler: es gilt $A \rightarrow B$, wenn in jeder gültigen Relation $r(R)$ für alle Tupelpaare $t1$, $t2$ aus der Gleichheit $t1[A] = t2[A]$ auch folgt $t1[B] = t2[B]$.

Man sagt auch: *A bestimmt B* oder *A* ist *Determinante* von *B*.

Beispiel 5–2

BEZ und FB in F sind gegenseitig funktional abhängig (BEZ $\rightarrow$ FB und FB $\rightarrow$ BEZ), weil Bezeichnung und Fachbereichsnummer 1:1 gekoppelt sind. Einseitige funktionale Abhängigkeit existiert für STATUS und das Schlüsselattribut NAME in M (NAME $\rightarrow$ STATUS), d. h.

- jeder Hochschulangehörige hat genau einen STATUS,
- STATUS ist funktional abhängig von NAME,
- NAME bestimmt STATUS,
- NAME ist Determinante von STATUS,
- mehrere Angehörige können den gleichen STATUS haben,
- NAME ist nicht funktional abhängig von STATUS (STATUS $\not\rightarrow$ NAME)
- f: W(NAME) $\rightarrow$ W(STATUS) ist eine Funktion.

Genauso ist ORT abhängig von NAME in M (NAME $\rightarrow$ ORT) und, wenn jeder höchstens ein Telefon hätte, TEL von NAME-FB in F-M ({NAME, FB} $\rightarrow$ TEL).

Mit dem Begriff der funktionalen Abhängigkeit können wir auch *Schlüssel* definieren: die Attributmenge K ist ein Schlüssel von Relationenschema R, wenn $K \rightarrow R$, d.h. aus $t1[K] = t2[K]$ folgt $t1 = t2$.

Ferner nennen wir eine Abhängigkeit $A \rightarrow B$ *trivial*, wenn $B \subseteq A$.

Definition 5–2

Eine Abhängigkeit $X \rightarrow Y$ heißt *voll*, wenn es keine echte Teilmenge $Z \subset X$ gibt, so daß $Z \rightarrow Y$. Gibt es eine solche echte Teilmenge Z, dann heißt $X \rightarrow Y$ *partielle* Abhängigkeit.

Beispiel 5–3

Attribut MITGLIED mit 1 für Erstmitglied, 2 für Zweitmitglied, 0 für nicht zutreffend, ist voll funktional abhängig von {NAME, FB}:

- die Kombination aus Name und Fachbereich bestimmt eindeutig die Mitgliedschaft,
- weder NAME noch FB allein bestimmen den MITGLIED-Wert, da zu einem Namen und einem Fachbereich mehrere MITGLIED-Werte auftreten.

Definition 5–3

Definition: Ein Attribut heiße *prim* wenn es Teil eines Schlüsselkandidats ist, sonst *nicht-prim*.

> **Hinweis**: Als *Schlüsselkandidaten* wurden im Abschnitt 3.3 *minimale* Mengen von Attributen bezeichnet, die eine eindeutige Identifizierung von Tupeln erlauben.

Beispiel 5–4

In F-M sind NAME und FB offensichtlich prim. Andere Schlüsselkandidaten existieren nicht, damit sind TEL und MITGLIED nicht-prim. In F sind FB und BEZ jeweils Schlüssel und damit prim.

Definition 5–4

Seien X und Y Attributmengen aus R, $X \subseteq R$ und $Y \subseteq R$, wobei X Determinante von Y ist, aber nicht umgekehrt, d.h. $X \rightarrow Y$ und $Y \not\rightarrow X$. Sei A

$\in R$ ein Attribut, das nicht in X oder Y auftrete und gelte $Y \rightarrow A$. Dann ist A *transitiv abhängig* von X.

Beispiel 5–5

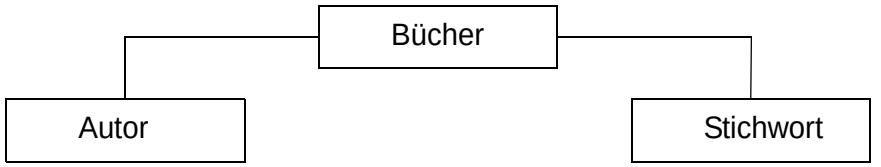
Beispiel: Man erweitere M zu M' durch die Standortadresse ADRESSE, die funktional abhängig von ORT ist. ORT ist abhängig vom Schlüssel NAME, NAME ist nicht abhängig von ORT (mehrere Mitarbeiter am selben ORT). ADRESSE hängt damit transitiv von NAME ab.

M'

NAME	ORT	STATUS	ADRESSE
Hans	Hopla	Prof	Holländischer Platz
Emil	WAllee	WB	Wilhelmshöher Allee
Anna	AVZ	Prof	AVZ Oberzwehren
Gabi	Hopla	VA	Holländischer Platz
Rolf	Hopla	WB	Holländischer Platz

Funktionale Abhängigkeiten $A \rightarrow B$, in denen B von A abhängt, schließen gewisse Tupel aus: NAME $\rightarrow$ STATUS schließt aus, daß ein Name mit zwei unterschiedlichen Statusangaben (z. B. einmal 'WB', einmal 'Prof') auftritt; eines der beiden Tupel wäre zu streichen.

Bei den *mehrwertigen Abhängigkeiten* ist gerade das Gegenteil der Fall; sie erzeugen fehlende Tupel. Sie treten auf bei Kombinationen von zwei sog. *Wiederholungsgruppen*, z.B. in einer Literaturdatenbank die Gruppen AUTOR (ein Buch kann mehrere Verfasser haben) und STICHWORT (zu einem Buch gehören meist mehrere Stichwörter).



Damit wären AUTOR, bzw. STICHWORT mehrwertig abhängig von BÜCHER.

Definition 5–5

Definition: In einer Relation R mit disjunkten Attributmengen A, B, C ist C *mehrwertig abhängig* von A (kurz: $A \longrightarrow\!\!\!\!\blacktriangleright\!\!\blacktriangleright C$), wenn zu jeder Wertekombination von (A, B) auch jeder Wert aus C auftritt. Wegen der Symmetrie der Hierarchie ist genauso B von A mehrwertig abhängig, also $A \longrightarrow\!\!\!\!\blacktriangleright\!\!\blacktriangleright B$.

Definition 5–6 (aus [KS86], S.200f)

Sei R ein Relationenschema und sei $X \subseteq R$ und $Y \subseteq R$. Eine *mehrwertige Abhängigkeit* $X \longrightarrow\!\!\!\!\blacktriangleright\!\!\blacktriangleright Y$ existiert, wenn in jeder gültigen Relation $r(R)$ für alle Paare von Tupeln $t1$ und $t2$ mit $t1[X] = t2[X]$ auch Tupel $t3$ und $t4$ in R auftreten mit

- $t1[X] = t2[X] = t3[X] = t4[X]$
- $t3[Y] = t1[Y]$
- $t3[R - Y] = t2[R - Y]$
- $t4[Y] = t2[Y]$
- $t4[R - Y] = t1[R - Y]$

Beispiel 5–6

'Hans' ist nicht nur Mitglied in zwei Fachbereichen, er hat auch zwei Telefonnummern für normales Telefon und Telefax. Die ersten beiden Tupel der Tabelle unten enthalten zwar bereits alle Informationen, jedoch sollten beide Nummern mit allen NAME, FB Kombinationen auftreten. (TEL, BEMERKUNG) ist mehrwertig abhängig von NAME, bzw. FB ist mehrwertig abhängig von NAME. Man beachte, daß NAME nicht funktional abhängig von FB ist und auch nicht umgekehrt. Da sich mehrere Leute ein Telefon teilen können und jemand mehrere Telefonnummern haben kann, sind auch NAME und TEL nicht funktional abhängig.

MVD (NAME $\longrightarrow\!\!\!\!\!\rightarrow$ TEL, BEMERKUNG, bzw. NAME $\longrightarrow\!\!\!\!\!\rightarrow$ FB)

NAME	FB	TEL	BEMERKUNG
Hans	17	4477	Tel
Hans	7	4478	Fax
Hans	17	4478	Fax
Hans	7	4477	Tel

Die mehrwertige Abhängigkeit (engl. *multivalued dependency*) würde nicht existieren, wenn 'Hans' im FB 17 unter 4477 und im FB 7 unter 4478 erreichbar wäre und man genau diese Aussage durch die Tabelle ausdrücken wollte.

Definition 5–7

Eine mehrwertige Abhängigkeit $X \longrightarrow\!\!\!\!\!\rightarrow Y$ ist *trivial*, wenn $Y \subseteq X$ oder $X \cup Y = R$.

Übung 5–1

Geben Sie eine Tabelle für das Beispiel mit BÜCHER, AUTOR und STICHWORT an!

5.2 Normalformen

Definition 5–8

Eine Relation ist in der 1. Normalform (1NF), wenn alle Attribute nur atomare Werte enthalten.

Die folgende Tabelle ist nicht in der 1. Normalform (¬1NF). Die Vor- und Nachteile der Algebra für geschachtelte relationale Datenbanken sind z.Zt. in der Diskussion, erste DBMS-Prototypen existieren.

Eine NF^2 (Non First Normal Form) Tabelle sähe etwa wie folgt aus.

UNI

{}UNI_MITGLIEDER			
NAME	{}FBS	{}TELS	
	FBNR	TELNO	BEMERKUNG
Hans	17	4477	Telefon
		4478	Fax
	7	4479	Modem
Emil	19	3344	Büro
		3343	Sekretariat
Anna	99	{ }	

Übung 5–2

Geben Sie eine NF^2-Tabelle für das Bücherbeispiel von oben an!

Definition 5–9

Eine Relation ist in der *2. Normalform* (2NF), wenn sie in 1. Normalform ist und jedes Attribut entweder prim oder voll funktional abhängig von jedem Schlüsselkandidaten ist.

Beispiel 5–7

In der Relation F-M von oben sei jedem Mitglied der Hochschule höchstens eine Telefonnummer zugeordnet. Dann ist F-M nicht in der 2NF, da das Attribut TEL nicht prim (Teil eines Schlüssels) und nicht voll funktional abhängig ist vom Schlüssel {NAME, FB}. Tatsächlich ist TEL bereits funktional abhängig von NAME.

Dagegen ist MITGLIED - wie oben ausgeführt - voll funktional abhängig von {NAME, FB}. Um die Relation F-M in die 2NF zu bringen, genügt es also, wenn TEL aus F-M in die Relation M wandert (vgl. M2 unten).

Hinweis: Die 2. Normalform kann nur verletzt werden, wenn ein Schlüsselkandidat zusammengesetzt ist. Sie spielt in der Normalisierungstheorie kaum noch eine Rolle.

F-M-2

NAME	FB	MITGLIED
Hans	17	1
Hans	7	2
Emil	19	1
Gabi	?	0
Anna	17	1

M2

NAME	ORT	STATUS	TEL
Hans	Hopla	Prof	4477
Emil	WAllee	WB	3344
Anna	AVZ	Prof	4476
Gabi	Hopla	VA	2233
Rolf	Hopla	WB	?

Übung 5–3

Wieso kann man leicht zeigen, daß M2 auch in der 2NF ist?

Definition 5–10

Ein Relationenschema R ist in der *3. Normalform* (3NF), wenn 1. Normalform vorliegt und für alle Abhängigkeiten $X \rightarrow A$ mit $X \subseteq R$, $A \in R$, gilt

- die Abhängigkeit $X \rightarrow A$ ist trivial, oder
- X ist Schlüssel von R, oder
- A ist prim.

Beispiel 5–8

In der Relation M' mit dem Attribut ADRESSE ist ADRESSE (Standortbezeichnung) abhängig von ORT, ORT ist nicht Schlüssel und ADRESSE ist nicht prim. Damit genügt M' nicht der 3NF, wohl aber der 2NF, da der Primärschlüssel NAME elementar ist.

Durch Trennung in zwei Relationen M''(NAME, ORT, STATUS) und STANDORTE(ORT, ADRESSE) wird auch die 3. Normalform erreicht.

STANDORTE

ORT	ADRESSE
Hopla	Holländischer Platz
AVZ	Oberzwehren
WAllee	Wilhelmshöher Allee

Übung 5–4

Zeigen Sie, daß F-M-2 in 3NF ist!

> **Hinweis:** Eine alternative Formulierung der 3. Normalform ist, für alle abhängigen, nicht-primen Attribute *A* in Relation *R* zu fordern, daß sie nicht transitiv von einem Schlüssel in *R* abhängen.

Übung 5–5

Zeigen Sie, daß ADRESSE in M' transitiv abhängig von NAME war.

Definition 5–11

Eine Relation ist in *Boyce-Codd Normalform* (BCNF), wenn jede Determinante ein Schlüsselkandidat ist.

Anders ausgedrückt: für alle funktionalen Abhängigkeiten $X \rightarrow A$ des Relationenschemas R mit $X \subseteq R$ und $A \in R$ gilt entweder

- $X \rightarrow A$ ist eine triviale Abhängigkeit (also $A \in X$) oder

- *X* ist Schlüssel von *R*.

Im Gegensatz zu 3NF ist die Alternative „oder *A* ist prim“ („*A* ist Teil eines Schlüsselkandidaten“) weggefallen. BCNF ist damit eine Verschärfung gegenüber 3NF.

Hinweis: Jedes zwei-attributige Relationenschema ist in BCNF.

Beispiel 5–9

In M2 ist Schlüssel NAME Determinante von ORT, STATUS und TEL. Kein anderes Attribut ist Determinante, damit ist M2 nicht nur in 3NF sondern auch schon in BCNF. STANDORTE ist automatisch in BCNF, weil es nur zwei Attribute hat.

Übung 5–6

Ist F-M-2 in BCNF?

Übung 5–7

Für die folgende Tabelle FNT (FB-NAME-TEL) gelte: mehrere Leute des selben Fachbereichs können sich ein Telefon teilen, jeder hat aber höchstens eine Nummer. Ferner gibt es mehrere Leute gleichen Namens, jedoch nicht im selben Fachbereich.

Damit seien {NAME, FB} und {NAME, TEL} Schlüssel und es existieren die folgenden Abhängigkeiten:

- {NAME, FB} $\rightarrow$ TEL und
- TEL $\rightarrow$ FB.

FNT

FB	NAME	TEL
17	Hans	4477
19	Hans	3344
17	Uwe	4477

Ist FNT in 3NF? Wenn ja, ist FNT sogar in BCNF? Begründung!

Übung 5–8

Welche Aufteilung bringt FNT in BCNF? Sind die Tabellen „natürlich“?

Übung 5–9

[Ullm, S. 384] In Amerika (und inzwischen auch bei uns) gibt es Postleitzahlen für Stadtbezirke. Eine Tabelle mit den Attributen Stadt, Straße und Postleitzahl ist dann in 3NF aber nicht in BCNF. Schlüssel sind {Stadt, Straße} und {Straße, Postleitzahl}. Wie lauten die Abhängigkeiten? Was verletzt BCNF?

Übung 5–10

Gegeben sei die Relation vom Anfang des 5. Kapitels:

> BESTPOSTEN(BESTNR, DATUM, KDNR, POS, ARTNR, MENGE).

Die Schlüsselkandidaten seien {BESTNR, POS} und {BESTNR, ARTNR}, d.h. gleiche Artikelnummern sind unter einer Position in der Bestellung zusammenzufassen.

- Wie lauten die Abhängigkeiten?
- Warum ist BESTPOSTEN nicht in der 2. Normalform?

Übung 5–11

Die Relation oben werde aufgeteilt in

BESTELLUNGEN(BESTNR, DATUM, KDNR) und
POSTEN(BESTNR, POS, ARTNR, MENGE).

Geben Sie alle Schlüsselkandidaten und Abhängigkeiten an! Genügen die neuen Relationen 3NF und sogar BCNF?

5.3 BCNF versus 3NF

Die Unterschiede zwischen BCNF und 3NF sind z.T. recht subtil.

A

NAME	FB	BEZ
Hans	17	Mathematik-Informatik
Hans	19	Elektro-Technik
Anna	17	Mathematik-Informatik
Emil	18	Physik

Schlüsselkandidaten sind {NAME, FB} und {NAME, BEZ}. Damit sind alle Attribute prim und A ist in 3NF. Die Abhängigkeiten sind {NAME, FB} $\rightarrow$ BEZ (partiell) und FB $\rightarrow$ BEZ, bzw. {NAME, BEZ} $\rightarrow$ FB und BEZ $\rightarrow$ FB.

Weil FB nicht Schlüsselkandidat ist, aber FB $\rightarrow$ BEZ gilt, ist A nicht in BCNF.

Man sieht, daß die 3NF einen gewissen Schutz vor Anomalien bietet. Das Tupel (’Hans’, 17, ’Physik’) ließe sich nicht einfügen, weil die zugesicherte Schlüsseleigenschaft von {NAME, FB} verletzt würde. Andererseits könnte man (’Fritz’, 17, ’Physik’) einfügen, solange es nicht bereits Tupel (’Fritz’, *x*, ’Physik’) oder (’Fritz’, 17, *y*) für beliebige *x* und *y* gibt. Somit läßt sich aber die 1:1 Abhängigkeit von FB und BEZ nicht wirklich durchsetzen, solange wir nicht eine getrennte Relation dafür schaffen.

Im folgenden Beispiel hänge die Art der Promotion vom Fachbereich ab, jedoch nicht umgekehrt. Einziger Schlüsselkandidat ist {NAME, FB}.

B

NAME	FB	PROMOTION
Hans	17	rer.nat
Hans	19	Dr. ing
Anna	17	rer.nat.
Emil	18	rer.nat.

PROMOTION ist nicht prim, FB kein Schlüssel, wegen FB $\rightarrow$ PROMOTION ist B nicht in 3NF. B ist sogar nicht einmal in 2NF, weil PROMOTION nicht voll von {NAME, FB} abhängt.

Verwendet man die alternative Definition der 3NF, ist es schwierig zu erkennen, daß PROMOTION transitiv vom Schlüssel {NAME, FB} abhängt. Der Grund ist, daß die 1. Abhängigkeit {NAME, FB} $\rightarrow$ FB trivialer Natur ist, trotzdem aber als Abhängigkeit zählt. Ferner hängt natürlich {NAME, FB} nicht von FB ab, womit die Transitivität von PROMOTION gegeben ist.

C

NAME	FB	PROMOTION
Hans	17	rer.nat
Hans-Peter	19	Dr. ing
Anna	17	rer.nat.
Emil	18	rer.nat.

Zuletzt betrachten wir C mit der Zusicherung, daß NAME Schlüssel von C ist. Damit ist C automatisch in 2NF. FB und PROMOTION sind nicht prim, PROMOTION hängt transitiv vom Schlüssel NAME ab, somit ist C nicht in 3NF.

Durch Trennung in C1 und C2 entstehen zwei Tabellen in BCNF. Das Problem der hängenden Tupel läßt sich damit aber nicht vermeiden: die PROMOTION eines (’Rolf’, 99) hinge weiterhin in der Luft!

C1

NAME	FB
Hans	17
Hans-Peter	19
Anna	17
Emil	18

C2

FB	PROMOTION
17	rer.nat.
19	Dr. ing
18	rer.nat.

Wenn BCNF sicherer als 3NF ist, warum bringt man nicht alle Tabellen durch entsprechende Aufspaltung in BCNF? Antwort:

- nicht alle Tabellen lassen sich „verlustfrei“ in BCNF, aber
- jede Tabelle läßt sich „verlustfrei“ in 3NF bringen.

Das Adjektiv „verlustfrei“ für eine Aufteilung läßt sich präzisieren zu

- kein *Informationsverlust* beim Wiederzusammensetzen (*lossless join*)
- lokale Prüfung von *Abhängigkeiten* möglich (*dependency preservation*).

Beispiel 5–10

Die FNT-Tabelle oben ist das kanonische Beispiel für eine Relation, die nicht in BCNF ist und bei deren Aufspaltung ein Informationsverlust ein-

tritt. Es galt {FB, NAME} → {TEL} und {TEL} → {FB}. Nach der Aufspaltung in FN(FB, NAME) und NT(NAME, TEL) ergibt der natürliche Join von FN mit NT die Tabelle unten, d. h. die Abhängigkeit {FB, NAME} → {TEL} ging verloren.

FN

FB	NAME
17	Hans
19	Hans
17	Uwe

NT

NAME	TEL
Hans	4477
Hans	3344
Uwe	4477

Join(FN, NT)

FB	NAME	TEL
17	Hans	4477
17	Hans	3344
19	Hans	4477
19	Hans	3344
17	Uwe	4477

Zum Verständnis benötigt man die Menge *F* von Beziehungen eines Relationenschemas *R*. Diese Menge *F* enthält alle angegebenen Beziehungen. Weitere Beziehungen lassen sich aus *F* ableiten, z.B. über Transitivität: aus $A \rightarrow B$ und $B \rightarrow C$ folgt ($\Rightarrow$) $A \rightarrow C$. Mit *F* und den daraus ableitbaren Beziehungen bildet man den *transitiven Abschluß* von (engl. *closure of*) *F*, kurz F^+, mit $F^+ = \{X \rightarrow Y \mid X \subseteq R, Y \subseteq R, F \Rightarrow X \rightarrow Y\}$.

Beispiel 5–11

R ={FB, NAME, TEL} von oben mit *F* = {{FB, NAME} → {TEL}, {TEL} → {FB}}. Wie noch zu zeigen, umfaßt F^+ die Abhängigkeiten

- {TEL} → {FB} und {TEL} → {FB, TEL}

- {FB, NAME} → {FB, NAME, TEL},
  {FB, NAME} → {FB, NAME}
  {FB, NAME} → {NAME, TEL}, {FB, NAME} → {FB, TEL}
  {FB, NAME} → {TEL},
  {FB, NAME} → {FB} und {FB, NAME} → {NAME}
- {TEL, NAME} → {FB, NAME, TEL},
  {TEL, NAME} → {TEL, NAME}
  {TEL, NAME} → {FB, NAME}, {TEL, NAME} → {FB},
  {TEL, NAME} → {TEL} und {TEL, NAME} → {NAME}
- {FB, NAME, TEL} → *Y*, für alle $Y \subseteq R$.

Weil es so aussieht, als ob alles in F^+ wäre: {FB, TEL} → {NAME} ist z.B. nicht enthalten.

Wie lassen sich die Abhängigkeiten in F^+ errechnen? Recht einfach durch die drei *Armstrongschen Axiome* [Arm74]:

- *Reflexivität*: Wenn $Y \subseteq X$, dann folgt $X \rightarrow Y$, die natürlichen oder trivialen Abhängigkeiten.
- *Erweiterung*: Wenn $X \rightarrow Y$ gilt und Z eine Teilmenge von R ist, dann gilt auch $XZ \rightarrow YZ$ (XZ ist kurz für $X \cup Z$).
- *Transitivität*: Wenn $X \rightarrow Y$ und $Y \rightarrow Z$, dann auch $X \rightarrow Z$.

Die Armstrongschen Axiome sind zusammen hinreichend und notwendig zur Berechnung des Abschlusses von F und die angegebenen Abhängigkeiten gelten in jeder von R erzeugten Relation r.

Man erleichtert sich die Arbeit, wenn man auch die folgenden zusätzlichen Regeln verwendet:

- *Vereinigung*: $X \rightarrow Y$ und $X \rightarrow Z$ impliziert $X \rightarrow YZ$.
- *Pseudotransitivität*: $X \rightarrow Y$ und $WY \rightarrow Z$ impliziert $WX \rightarrow Z$.
- *Aufspaltung*: Gilt $X \rightarrow Y$ und $Z \subseteq$ Y, dann gilt auch X → Z.

Übung 5–12

Zeigen Sie, daß sich F^+ im Beispiel oben mit den hier genannten Axiomen und Regeln herleiten läßt!

Der wesentliche Nutzen von F^+ ist, daß damit die Äquivalenz von Abhängigkeitsmengen F und G definiert werden kann:

F und G sind *äquivalent* gdw. $F^+ = G^+$.

Damit lassen sich wiederum für ein Relationenschema R gewisse kanonische Abhängigkeitsmengen angeben, speziell die *minimale Hülle*, die z.B. die Eigenschaft hat, daß auf den rechten Seiten der Abhängigkeiten *nur einzelne Attribute* auftreten, links keine überflüssigen Attribute genannt werden und daß sie für jedes F existiert, allerdings nicht notwendigerweise eindeutig.

5.4 Verlustfreie und abhängigkeitserhaltende Aufteilungen

Die Aufteilung eines Relationenschemas R in Teilschemata R_1, R_2, ..., R_n mit $R = R_1 \cup R_2 \cup ... \cup R_n$ entspricht einer Projektion $r_i = \pi_{Ri}(r)$. Das Wiederzusammensetzen damit einem natürlichen Join über alle r_i. Dabei enthält der Join über alle Teilrelationen immer mindestens alle Tupel aus der ursprünglichen Relation r, d.h. der „Verlust an Information“ besteht darin, daß zu viele Tupel aufgeführt werden (vgl. den Join von FN und NT oben).

Für gegebene Menge F von Abhängigkeiten nennen wir eine Aufteilung $\{R_1, R_2, ..., R_n\}$ eines Schemas R *verlustfrei* (*lossless-join partition*) wenn für alle unter F legalen Relationen $r(R)$ gilt

$$r = \bowtie \pi_{Ri}(r) \text{ für } i = 1, ..., n$$

Eine solche Aufteilung ist immer möglich, auch bei BCNF Teilschemata. Ob die Aufteilung einen verlustfreien Join ermöglicht, läßt sich auch leicht prüfen, sofern man F^+ bereits hat (aufwendig!): eine Aufteilung von R in R_1 und R_2 erlaubt einen verlustfreien Join wenn wenigstens eine der beiden Abhängigkeiten

- $R_1 \cap R_2 \rightarrow R_1$
- $R_1 \cap R_2 \rightarrow R_2$

in F^+ ist.

Beispiel 5–12

Die Aufteilung von FNT in R_1 = {FB, NAME} und R_2 = {NAME, TEL} oben zeigte einen Verlust von Abhängigkeiten. Sie ergibt auch keinen verlustfreien Join, denn $R_1 \cap R_2$ = {NAME} und weder {NAME} $\rightarrow$ {FB, NAME} noch {NAME} $\rightarrow$ {NAME, TEL} sind in F^+.

Die Aufteilung in R_1 = {FB, TEL} und R_2 = {NAME, TEL} ist jedoch verlustfrei, denn aus {TEL} $\rightarrow$ {FB} in F und der trivialen Abhängigkeit {TEL} $\rightarrow$ {TEL} entsteht durch Vereinigung {TEL} $\rightarrow$ {FB, TEL} in F^+.

FT

FB	TEL
17	4477
19	3344

NT

NAME	TEL
Hans	4477
Hans	3344
Uwe	4477

Join(FT, NT)

FB	TEL	NAME
17	4477	Hans
17	4477	Uwe
19	3344	Hans

Tatsächlich entsteht beim Join wieder die ursprüngliche Tabelle FNT aus Übung 5–7 .

Wir zeigen jetzt, daß auch bei dieser Aufteilung ein Verlust von Abhängigkeiten eintritt, d. h. die Aufteilung ist nicht *abhängigkeitsbe-*

wahrend (*dependency preserving*). Dazu definieren wir analog zur Projektion der Tabelle r auf r_i die Einschränkung von F auf F_i als die Teilmenge der Abhängigkeiten aus F^+, die nur Attribute aus R_i enthalten. Die Einschränkung F_i hat den Vorteil, daß die Zusicherungen der Abhängigkeiten lokal, also nur mit den Attributen aus der Unterrelation, und damit sehr effizient erfolgen kann.

Folgt aus den lokalen Zusicherungen über den Einschränkungen auch die globale Zusicherung, nennen wir die Aufteilung abhängigkeitsbewahrend, d.h. es muß gelten

$$F^+ = \left(\bigcup_{i=1}^{n} F_i \right)^+$$

> **Hinweis:** Die Vereinigung der F_i ist im allgemeinen ungleich F. Deshalb muß der aufwendigere Weg der Überprüfung der Aufteilung über den Abschluß von F gehen. Dieser Test wird aber nur „einmal“ beim Datenbankentwurf gemacht.

Beispiel 5–13

$F_1 = F_{FT}$ darf nur die Attribute FB und TEL enthalten. Damit F_1 = {{TEL} → {FB}, {TEL} → {TEL, FB}} und $F_2 = F_{TN}$ darf nicht FB enthalten, wodurch nur triviale Abhängigkeiten entstehen: F_2 = {{TEL, NAME} → {NAME, TEL}, {TEL, NAME} → {NAME} und {TEL, NAME} → {TEL}}.

Damit enthält aber $(F_1 \cup F_2)^+$ nicht die bereits in F auftretende Abhängigkeit {FB, NAME} → {TEL} und ist damit ≠ F^+. Intuitiv kann man aus den Beispieltabellen erkennen, daß in NT ein Tupel (’Uwe’, 4000) eingefügt werden kann, das den trivialen Abhängigkeiten von F_2 genügt. In FT wird eingefügt (17, 4000), das der lokalen Abhängigkeit {TEL} → {FB} (keine Telefonnummer für zwei FB’s) genügt. Beim Join entsteht dann (17, ’Uwe’, 4000) und damit wird global {FB, NAME} → {TEL} verletzt. Gleichzeitig fällt auf, daß dieses Tupel gegen die Annahme verstößt, {FB, NAME} sei ein Schlüsselkandidat.

Übung 5–13

Zeigen Sie durch die Angabe der F_i, daß die erste Aufteilung von FNT in R_1 = {FB, NAME} und R_2 = {NAME, TEL} auch nicht abhängigkeitserhaltend ist!

Da wir damit alle sinnvollen Aufteilungen, die nicht alle drei Attribute aus FNT enthalten, getestet haben, folgt daraus der formelle Beweis, daß es Relationen und Abhängigkeiten gibt, die nicht abhängigkeitserhaltend aufgeteilt werden können.

Auf die Angabe der Verfahren zur Erzielung von verlustfreien, abhängigkeitserhaltenden (soweit möglich!) Aufteilungen wird hier verzichtet (vgl. etwa [KS86], S.194).

Übung 5–14

Geben Sie eine verlustfreie, abhängigkeitserhaltende Aufteilung von BESTPOSTEN an!

Als letzter Hinweis möge genügen, daß man nicht jede Relation „auf Teufel komm raus“ normalisieren soll. Im Beispiel M2 mit NAME, ORT, STATUS, TEL war NAME der Schlüssel und alle anderen Attribute hingen von NAME ab. M2 ist bereits in BCNF, ließe sich aber noch weiter in 3 zwei-attributige Relationen (NAME, ORT), (NAME, STATUS) und (NAME, TEL) aufteilen.

Für die Abfrage „Nenne Namen und Telefonnummer aller Verwaltungsangestellten ([STATUS] = ’VA’) am Hopla“ wären jetzt aufwendige Joins nötig.

5.5 Die 4. Normalform

Die Bedingung für die Erfüllung der *4. Normalform* (4NF) ist analog zu BCNF mit dem Unterschied, daß *funktionale Abhängigkeit* durch *mehrwertige Abhängigkeit* ersetzt wird. Danach ist ein Relationenschema in 4NF wenn für alle nicht-trivialen mehrwertigen Abhängigkeiten der Form $X \longrightarrow\!\!\!\!\blacktriangleright\!\!\blacktriangleright\ Y$ mit $X \subseteq R$, $Y \subseteq R$ gilt: X ist Schlüssel von R.

Hinweis: Eine mehrwertige Abhängigkeit $X \twoheadrightarrow Y$ hieß trivial, wenn $Y \subseteq X$ oder $R = X \cup Y$.

Beispiel 5–14

Wir betrachten die Tabelle MVD, bei der Personen mehreren Fachbereichen angehören und mehrere Telefone besitzen können.

MVD

NAME	FB	TEL	BEMERKUNG
Hans	17	4477	Tel
Hans	7	4478	Fax
Hans	17	4478	Fax
Hans	7	4477	Tel
Uwe	19	3344	Büro
Uwe	19	3343	Sekretariat

Durch Trennung in die Tabellen NF und NTB wird 4NF erreicht.

NF

NAME	FB
Hans	17
Hans	7
Uwe	19

NTB

NAME	TEL	BEMERKUNG
Hans	4477	Tel
Hans	4478	Fax
Uwe	3344	Büro
Uwe	3343	Sekretariat

Übung 5–15

Bringen Sie die Büchertabelle von oben in 4NF!

6 Realisierungen des Transaktionsprinzips

6.1 Sicherung vor Datenverlust

Wesentlicher Vorteil eines DBMS gegenüber einem Dateisystem ist - neben der Trennung der physischen von der logischen Sicht der Daten - das Transaktionsprinzip.

Hierzu werden DB-Operationen geklammert mit

```
BEGIN-TRANSACTION
 ...
 Operationen
 ...
COMMIT-TRANSACTION bzw. ABORT-TRANSACTION.
```

Transaktionen bieten Sicherheit vor Datenverlust und Verfälschung (Inkonsistenzen) bei den drei Arten von Versagenssituationen

- *Transaktionsabbruch*
- *Systemfehler* („Systemabsturz“)
- *Verlust des Datenträgers* („Plattencrash“)

Beispiel 6–1 Versagenssituationen

- *Transaktionsabbruch* wegen nicht verfügbarer Resourcen, z. B. Kunde möchte einen Flug buchen FRA - HEL über HAM, FRA - HAM ok und reserviert, HAM - HEL ausgebucht. Dann freigeben FRA - HAM, wenn Kunde Reiseplan ganz aufgibt oder anderen Tag/Zeit nennt.

- *Systemfehler* durch Programmfehler im DBMS, Betriebssystem, Anwendungsprogramm (Division durch Null), Hardwarefehler, Datei voll, Platte voll, Übertragungsfehler, etc. Flüchtige Daten sind weg (Hauptspeicher, Seitenpuffer), Daten auf Platte nicht verloren, ggf. letzter Satz nur partiell geschrieben.
- *Verlust des Datenträgers* bei Headcrash der Platte, Brand oder Vandalismus im Rechenzentrum, Diebstahl der Datenträger.

Unterscheidung der Datenspeicherung in drei Klassen:

- *flüchtig* (HS, Puffer), engl. *volatile*
- *nicht-flüchtig* (Platte, Band), engl. *persistent*
- *sicher, stabil* (Kopien auf unabhängigen Datenträgern), engl. *stable storage*

Dabei ist zu beachten

- Write-Befehl verändert üblicherweise nur den Puffer, in dem die Seite sich befindet.
- Puffer (Datenblock) wird erst später bei Verdrängung physisch geschrieben.
- Verdrängung kann erzwungen werden durch *flush-Operation*, sog. *forced write*.

Ferner sind sehr subtile Fehler möglich, analog zu „Datei selbst geändert, aber Verweis auf Datei (Verzeichniseintrag) noch nicht geändert“, bzw. „... aber Blockzuteilungstabelle noch nicht modifiziert“. Fehlertoleranz ist auch abhängig von den Datenstrukturen, z. B. B-Baum versus Hashtabellen. Eine erschöpfende Untersuchung mit praktischen Messungen findet sich in [Küsp85], einen hervorragenden Vergleich mit Beschreibung der ARIES-Methode, auf die wir unten eingehen, enthält [MHLPS].

Transaktionsprinzip verlangt Einhaltung der sog. *ACID-Eigenschaften*:

- *Atomicitiy* (atomares Verhalten)
  „Alles-oder-Nichts-Prinzip“, d. h. Transaktion wird vollständig

durchgeführt bis zum Commit oder bei Abbruch ganz ungeschehen gemacht.

- *Consistency* (Bewahrung der Konsistenz, ~ der logischen Integrität)
  jede mit COMMIT abgeschlossene Transaktion bewahrt per definitionem die Konsistenz der Datenbasis.

- *Isolation*
  schützt den einzelnen Benutzer vor den Auswirkungen des *Mehrbenutzerbetriebs*, d.h. Ausführung der Transaktion erfolgt, wie wenn Benutzer alleine wäre; wird meist durch *Sperren von Datensätzen* erreicht.

- *Durability* (Beständigkeit der Daten)
  garantiert, daß mit Commit abgeschlossene Transaktionen auch über Systemabstürze und Datenträgerverlust hinaus ihre Gültigkeit dadurch behalten, daß die gemachten Modifikationen sicher gespeichert sind.

Implementierungstechniken im wesentlichen:

- Datenbank zurückfahren, *undo-Operationen*, Auswirkungen noch nicht abgeschlossener Transaktionen ungeschehen machen.

- Datenbank vorwärtsfahren, *redo-Operationen*, verlorene Auswirkungen bereits abgeschlossene Transaktionen wieder herstellen („repeat history“).

Dazu benötigt man:

- *Sicherungspunkte (check-points)*
  markiert konsistenten Zustand der Datenbank zum Zeitpunkt t_i.

- *Plattenkopien (dump)*
  auf stabilem Speicher, z. B. einmal wöchentlich.

- *Log-Dateien, log files, write-ahead logging (WAL-Protokoll)*
  werden in stabilen Speicher geschrieben und enthalten Aufzeichnungen über Start, Operationen und Daten, Abschluß von Operationen; alternativ

- *Schatten-Seiten (shadow pages)*
  von jeder zu modifizierenden Seite existieren wenigstens zwei Versionen.

Zwei Vorgehensweisen möglich bei Log-Dateien:

- *verzögerte Änderungen*, analog zu Zurückkopieren (copy back) bei Cache-Speichern
- *sofortige Änderungen*, analog zu Durchschreiben (write-through)

Verzögertes Ändern (deferred update)

- Transaktion T_j schreibt mit Beginn ihrer Ausführung einen Datensatz „Start T_j“ in Log-Datei.
- Ausführung einer Schreiboperation führt zu einem Datensatz „T_j schreibt neuen Wert *x* in Adresse *p*:*s*“.
- Der logische Abschluß der Transaktion wird mit Satz „T_j Commit“ festgehalten.
- Jetzt Log-Information benutzen, um Änderungen in der Datenbank vorzunehmen.
- Bricht Transaktion vor dem Schreiben von „T_j Commit“ ab, Log-Information einfach ignorieren.
- Kommt es zu einem Fehler nach Schreiben von „T_j Commit“, muß ganze Transaktion mit der Log-Information wiederholt werden (redo).
- Da auch diese Wiederholung „abstürzen“ kann, müssen die Schreiboperationen idempotent sein (mehrfache Ausführung wirkt wie einfache Ausführung); schwierig bei Entferne-Operationen.
- Deshalb auch Speicherung von neuem Wert (sog. *after image*), nicht Unterschiedsbetrag (sog. *delta-Wert*).
- Wiederholung der Transaktionen setzt immer an letztem *Checkpoint* auf und fährt DB vorwärts.

Beispiel 6–2

Erläutern Sie am Beispiel „Abbuchen € 1000,- von Konto A mit ursprünglich Saldo 3000,- und Zubuchen auf Konto B mit ursprünglich Saldo 5000,-“ den Ablauf der Transaktion. Berücksichtigen Sie den Abbruch zu verschiedenen Zeitpunkten!

Sofortige Änderungen (immediate updates)

- In der Log-Datei werden neben „Start T_j“ und „T_j Commit“ nur die Änderungswerte (deltas) oder alter und neuer Wert (Vorher- und Nachher-Kopie, *before and after image*) gespeichert und zwar bevor die eigentlichen Änderungen in der Datenbasis vorgenommen werden.
- Im Falle eines Fehlers wird die Datenbasis zurückgerollt bis zum letzten *Sicherungspunkt*, d.h. mittels der Log-Information werden alle Operationen ungeschehen gemacht (undo). Hierzu benötigt man die inversen Operationen (sog. *Kompensationsoperationen*).
- Danach werden alle abgeschlossenen Transaktionen – erkennbar an „T_j Start“ mit folgendem „T_j Commit“ - wiederholt (redo).

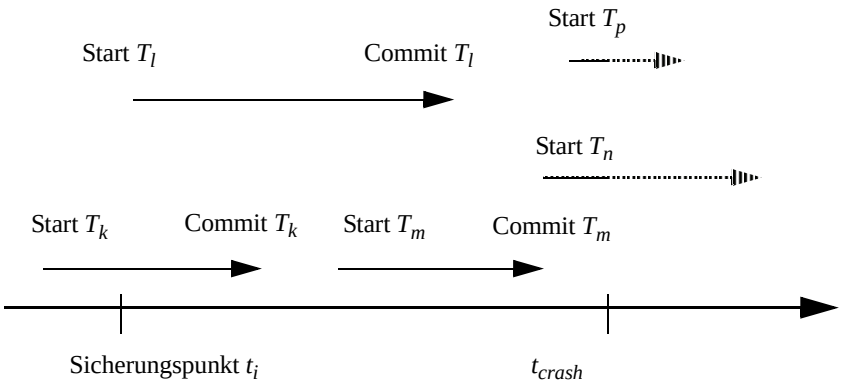


Abb. 6–1 *Überblick über offene und abgeschlossene Transaktionen zum Zeitpunkt eines Systemcrashs*

Auch hier wieder Fehler beim Vor-/Zurückrollen möglich. Dann entweder Log-Datei für Undo-/Redo-Operationen oder Redo-/Undo-Operationen sind idempotent.

Übung 6–1

Welche Vor- und Nachteile ergeben sich für sofortige Änderungen versus zurückgestellte Änderungen? Bei welcher Art müssen Satzsperren länger gehalten werden?

Doppelseiten (shadow pages)

Alternative zu Log-Dateien, manchmal Mischung mit Log-Dateien (z.B. System R).

Der Datenbereich wird in Seiten aufgeteilt, nicht notwendigerweise physisch fortlaufend. Es existieren zwei Seitentabellen:

- Schattentabelle (ST)
- Aktuelle Tabelle (AT)

ST wird während der Transaktion nicht verändert und steht in nicht-flüchtigem Speicher an fest vereinbarter Stelle (ähnlich wie FAT bei DOS oder i-list bei UNIX).

Beim 1. schreibenden Zugriff auf eine SeiteAT[j] = x wird in AT eine leere Seite (z.B. y) gesucht und AT[j] wird auf y gesetzt. Der neue Wert wird dann in Seite y geschrieben. Nach einem Commit werden alle betroffenen Seiten auf die Platte durchgeschrieben (flush-Operation) und AT wird auf die Platte kopiert, jedoch nicht über ST. Die neue Adresse von AT wird vermerkt, AT wird zu ST und die nächste Transaktion kann anlaufen.

Bei einem Fehler liefert ST den Zustand vor Start der Transaktion, damit wird kein undo benötigt. Bereits abgeschlossene Transaktionen, deren AT zu ST wurde, brauchen auch kein redo, da ihr neuer Zustand gesichert geschrieben wurde.

Zusammenfassung

Shadow pages gewöhnlich schneller als logging, führt aber zu schlechtem Lokalitätsverhalten und verlangt nach periodischem Einsammeln von leeren Seiten (garbage collection). Schwieriger zu implementieren bei

hohem Parallelitätsgrad der Transaktionen. Wurde im System R benutzt zusammen mit Logging, nicht jedoch in DB2, heute nur noch in SQL/DS anzutreffen.

Log-Dateien lassen sich für Versionsmanagement ausnutzen, damit sog. *As-of-Abfragen* möglich, z. B. Lagerbestand von ArtNr. 4711 per (as of) 1.1.1991. An den Datensatz wird eine Kette der Änderungen mit Änderungsdatum gehangen, die rückwärts in die Vergangenheit führt. Für einen gelöschten Satz wird ein sog. „Grabstein“ gesetzt.

Log-Dateien werden eingesetzt in folgenden kommerziellen Systemen und Prototypen:

- IBM AS/400, DB2, IMS, OS/2 Extended Edition Database Manager, Quicksilver, Starburst, System/38
- CMU Camelot
- Unisys DMS/1100
- Tandem Encompass und NonStop SQL
- Informix Informix-Turbo
- Honeywell MRDS
- MCC ORION
- SYNAPSE
- DEC VAX DBMS und VAX Rdb/VMS

Heutige Themen:

- Versionsmanagement
- lange Transaktionen (> einige Tage, Wochen) z. B. im CAD-Bereich
- geschachtelte Transaktionen
- Transaktionen im Multimedia-Bereich

- bei verteilten Systemen, z. B. *2-Phasen-Commit-Protokoll* bei Übertragung auf gestörten Kanälen

Übung 6–2

Vergleichen Sie die Sicherungstechniken mit der Praxis bei Textverarbeitungsprogrammen.

Übung 6–3

Bei einer Multimedia-Datenbank werden an einer Pixelgraphik (20 MB) geringe Retouschen vorgenommen. Welche Art der Sicherungsmaßnahmen wird man sich wünschen?

Übung 6–4

Die Grundsätze ordnungsgemäßer Buchhaltung verlangen, daß Änderungen nicht durch Überschreiben (das berühmte Verbot zu radieren!) erfolgen und daß sich für jeden Posten ein *Buchungspfad* ermitteln läßt. Welche Sicherungstechnik kommt dem entgegen?

6.2 Die ARIES-Methode

Um die Verfahren zur Transaktionssicherung etwas genauer beschreiben zu können, gehen wir hier auf die ARIES-Methode ein. ARIES steht für *Algorithm for Recovery and Isolation Exploiting Semantics* und ist die in vielen kommerziellen DB-Produkten der IBM, spez. DB2, eingesetzte Recovery-Technik. Ihre Beschreibung ist dem Artikel von Mohan et al. [MHLPS92] entnommen und findet sich auch in dem Datenbankbuch von Kemper und Eickler [KE96].

ARIES arbeitet mit einem *log*, also einer Datei, die man sich als unendliche, serielle Datei zum Festhalten des Fortschritts von Transaktionen vorstellen kann.

Jeder Satz im log bekommt eine eindeutige *Log Sequence Number* (*LSN*) zugeordnet, meist die monoton wachsende logische Satznummer

des log. Neue Sätze für das log werden dabei zuerst in flüchtige Puffer geschrieben und dann von Zeit zu Zeit, z.B. bei einem Commit, mittels der *force*-Operation in stabilen Speicher geschrieben. Man nennt dies „*forcing the log up to a certain LSN*“ (Durchschreiben des log bis zu einer LSN).

Drei Arten von Sätzen werden unterschieden

- undo-redo-log-records
- undo-only-log-records
- redo-only-log-records.

Undo-redo-Information kann dabei *physisch* als before- und after-image oder *operational* in der Art „addiere 5 to Feld 4 von Satz 15“ gespeichert werden. Operation logging erlaubt potentiell mehr Parallelität durch Ausnutzung semantischer Information als striktes Sperren mit X-Sperren die während des ganzen Commits gehalten werden.

ARIES verwendet ferner das weithin anerkannte WAL (write ahead logging) Protokoll, bei dem „update-in-place“, also schreiben an den alten Speicherplatz“ erfolgt, was voraussetzt, daß zumindestens die undo-Information für dieses Update auf das log in stabilem Speicher herausgeschrieben wurde. Um die Einhaltung des Protokolls zu überwachen, steht in jeder Seite die LSN des log-Satzes, der die letzte Änderung auf dieser Seite verursacht hat.

Der Transaktionsstatus wird auch im log aufgezeichnet und keine Transaktion ist abgeschlossen, bevor nicht ihr Commit-Status und alle sie betreffenden log-Daten auf stabilem Speicher gesichert sind, wozu das log bis zur LSN des Commit-Satzes auf stabilen Speicher durchgeschrieben sein muß.

6.3 Mehrbenutzerzugriff

Mehrbenutzerbetrieb verlangt die Isolierung der einzelnen Transaktion vor Auswirkungen des quasi-parallelen Betriebs (vgl. ACID-Eigenschaften oben). Dies wird erreicht durch „Kontrolle der Nebenläufigkeit“ (*concurrency control*).

Beispiel 6–3 Platzbuchung

Buchen *n* Plätze auf Flug *x*

```
Lesen(Freie-Plätze, x);
Wenn Freie-Plätze >= n
   dann Freie-Plätze := Freie-Plätze - n
   sonst Buchung zurückweisen;
Schreiben(Freie-Plätze, x);
```

Zwei verzahnt ablaufende Transaktionen T_1 und T_2 können dabei ein falsches Ergebnis liefern (z.B. ein Platz noch frei und jede Transaktion will einen Platz).

T_1	T_2
Lesen(FP, *x*)	
	Lesen(FP, *x*)
Test positiv	
FP := FP - 1	
Schreiben(FP {= 0}, *x*)	
	Test positiv
	FP := FP - 1
	Schreiben(FP {= 0}, *x*)

Offensichtlich wurden beiden Kunden ein Platz vergeben, obwohl nur noch einer vorhanden war (sog. *lost update problem*). Dieses Ergebnis wäre nicht eingetreten, hätte man die Transaktionen T_1 und T_2 hintereinander (sequentiell) ablaufen lassen.

Bei der *Ablauffolge* (engl. *schedule*) $<T_1, T_2>$ hätte T_1 den Platz bekommen, bei der Folge $<T_2, T_1>$ hätte T_2 ihn bekommen. Damit kann man aber jetzt eine Korrektheitskriterium für den Mehrbenutzerzugriff definieren:

> eine nebenläufige Ablauffolge *S* von Operationen heißt *serialisierbar*, wenn es eine serielle Ablauffolge *S'* gibt, die in ihren Auswirkungen äquivalent zu *S* ist.

Die Äquivalenz zweier Folgen ist definiert durch die beiden Bedingungen:

- Jede der Leseoperationen in den Folgen liest Datenwerte, die von den selben Schreiboperationen in beiden Folgen erzeugt wurden.
- Die letzte Schreiboperation für jedes Datenobjekt ist die gleiche in beiden Folgen.

Kontrollmechanismen, die nur serialisierbare Folgen von Operationen erzeugen, garantieren dann korrekte Nebenläufigkeiten.

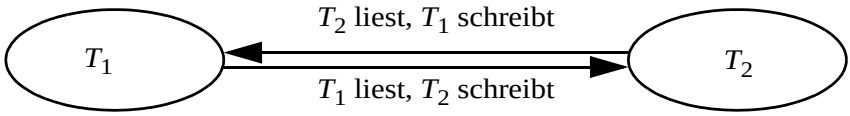
> Hinweis: Man beachte, daß nur Äquivalenz zu einem seriellen Ablauf gefordert wird. Laufen n Transaktionen T_1, T_2, ..., T_n parallel, gibt es $n!$ unterschiedliche serielle Folgen. Die Anzahl der möglichen Ablauffolgen der Operationen ist sehr viel größer, da jede Transaktion T_i aus m_i Operationen besteht. Der Ablauf der Lese-/Schreiboperationen innerhalb einer Transaktion wird als gegeben und nicht veränderbar angesehen.

Wird ein Datensatz gelesen bevor er geschrieben wird, läßt sich Serialisierbarkeit einer Ablauffolge effizient testen, indem ein Abhängigkeitsgraph gebildet und dieser auf Zyklen getestet wird (geht in $O(n^2)$ Schritten).

Der *Abhängigkeitsgraph* (engl. *precedence graph*) enthält eine Kante von Transaktion T_i nach Transaktion T_j ($T_i \rightarrow T_j$) wenn

- T_i einen Satz liest bevor T_j ihn schreibt, oder
- T_i einen Satz schreibt bevor T_j ihn liest.

Damit erhalten wir für das Beispiel oben den untenstehenden Zyklus.



Übung 6–5

Zeigen Sie, daß auch die folgende Ablauffolge nicht serialisierbar ist.

T_1	T_2
Lesen(FP, *x*)	
	Lesen(FP, *x*)
	Test positiv
	FP := FP - 1
	Schreiben(FP {= 0}, *x*)
Test positiv	
FP := FP - 1	
Schreiben(FP {= 0}, *x*)	

Die seriellen Folgen sind dann gerade die Folgen, die bei einer *topologischen Sortierung* des Abhängigkeitsgraphen entstehen. Entfällt die Annahme, daß ein Satz gelesen wird vor dem Schreiben, kann man Serialisierbarkeit allerdings nicht mehr effizient testen.

Übung 6–6

Bezeichne $R_i(x)$ eine Leseoperation von Transaktion T_i auf Datenobjekt *x*, $W_i(x)$ analog eine Schreiboperation. Gegeben seien die Transaktionen:

T_1: $R_1(A)$; A := A - 10; $W_1(A)$; $R_1(B)$; B := B + 10; $W_1(B)$;

T_2: $R_2(B)$; B := B - 20; $W_2(B)$; $R_2(C)$; C := C + 20; $W_2(C)$;

Sind die folgenden Ablauffolgen F_a und F_b serialisierbar?

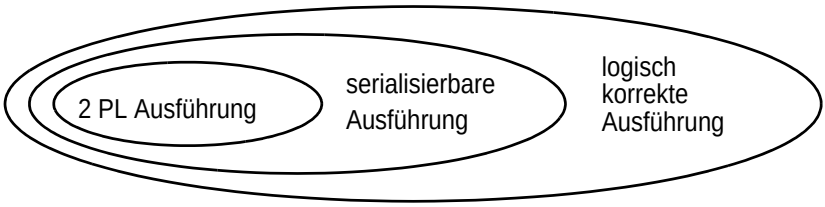
F_a: $R_1(A)$; $R_2(B)$; A := A - 10; B := B - 20; $W_1(A)$; $W_2(B)$; $R_1(B)$; $R_2(C)$; B := B + 10; C := C + 20; $W_1(B)$; $W_2(C)$;

F_b: $R_1(A)$; A := A - 10; $R_2(B)$; $W_1(A)$; B := B - 20; $R_1(B)$; $W_2(B)$; B := B + 10; $R_2(C)$; $W_1(B)$; C := C + 20; $W_2(C)$;

Korrekte Nebenläufigkeit von Transaktionen läßt sich im wesentlichen mit drei Techniken erreichen:

- 2-*Phasen Sperren* (2PL, 2-*phase locking*)
- *Zeitmarken*
- *optimistische Verfahren.*

Die Menge der möglichen Ablauffolgen ist unten angegeben und zeigt, daß es logisch korrekte Folgen gibt, die nicht serialisierbar sind und serialisierbare Folgen, die 2PL nicht zuläßt.



Sperren

Man unterscheidet

- S-Sperren (s-locks, *shared locks*) für gemeinsamen, lesenden Zugriff und
- X-Sperren (x-locks, *exclusive locks*) für ausschließenden, schreibenden Zugriff.

Dabei sind verschiedene *Granularitäten* möglich (Tupel sperren, Seite sperren, Datei sperren). Generell wird beim Zugriff auf ein Datenelement eine Sperre gesetzt. Ist das Element schon exklusiv gesperrt oder wenn exklusiver Zugriff auf S-gesperrtes Objekt erfolgen soll, muß die Transaktion warten oder aufgeben.

Heute häufig mehr Arten von Sperren, z.B. in ARIES

- S (shared)
- X (exclusive)
- IX (intention exclusive, Anmeldung für exklusiven Zugriff)
- IS (intention shared)

- SIX (shared intention exclusive, gemeinsamer Zugriff z.B. auf Tabelle, Anmeldung auf exklusiven Zugriff auf ein Untertupel)

Ein Objekt kann mehrere *Sperren* haben, solange sie verträglich sind.

Sperrkompatibilität nach [MHLPS]:

	S	**X**	**IS**	**IX**	**SIX**
S	✔		✔		
X					
IS	✔		✔	✔	✔
IX			✔	✔	
SIX			✔		

Das Setzen und Abfragen von Sperren benötigt ca. 100 Maschineninstruktionen. Da diese Operationen selbst vor Mehrbenutzerzugriff geschützt werden müssen, werden u. U. sog. *Latches* (Riegel) eingeführt, die nur etwa 10 Instruktionen benötigen und ähnlich wie *Semaphore* in Betriebssystemen arbeiten.

Mit Sperren wird das *Zweiphasen-Sperrprotokoll* (2PL) realisiert, das in der

- *Wachstumsphase* alle benötigten Sperren erlangt, und in der
- *Reduzierungsphase* Sperren aufgibt ohne neue zu erlangen.

Diese Technik ist hinreichend für Serialisierbarkeit, genügt aber nicht für Isolierung, wenn andere Transaktionen Resultate vor dem Commit lesen können. Deshalb üblicherweise

- Halten der X-Sperren bis zum Commit.

Damit ist aber die Gefahr einer *Verklemmung* gegeben, für deren Entstehung die folgenden 4 Umstände notwendig sind:

- exklusiver Zugriff (*mutual exclusion*)
- inkrementelle Zuteilung (*wait for*)

- kein Entzug (*no preemption*)
- zyklisches Warten (*circular wait*)

Beispiel 6–4 Verklemmung

Zwei Transaktionen T_1 und T_2. T_1 bucht 100,- € von Konto *A* auf *B* und T_2 bucht 200,- € von *B* auf *A*.

T_1	T_2
X-Sperren *A*	
	X-Sperren *B*
X-Sperre *B* verlangt, aber nicht erteilt	
	X-Sperre *A* verlangt, aber nicht erteilt
---------- Verklemmung ----------	
Lesen(*A*);	
Lesen(*B*);	
A := *A* - 100;	
B := *B* + 100;	
Schreiben(*A*);	
Schreiben(*B*);	
X-Sperren aufgeben	
	Lesen(*B*);
	Lesen(*A*);
	B := *B* -200;
	A := *A* + 200;
	Schreiben(*B*);
	Schreiben(*A*);
	X-Sperren aufgeben

Grundsätzlich zwei Möglichkeiten:

- *Verklemmungsvermeidung* (z. B. alle Sperren sofort anfordern; wenn nicht alle erlangbar, dann Abbruch)

- *Verklemmungsentdeckung* (meist über Zeitschranke und folgende Analyse der *Wartezyklen*) und Zurücksetzen.

Beides teuer: Entdeckung ist $O(n^2)$ Verfahren – Zyklenentdeckung! Ferner kann es zum *Aushungern* (*starvation*) von Transaktionen führen. Entzug von Sperren und Neustart der Transaktion erhöht Systembelastung und damit Wahrscheinlichkeit der erneuten Verklemmung.

Übung 6–7

Zeigen Sie, daß die Ablauffolge

F_1: $R_1(A)$; $A := A - 100$; $W_1(A)$; $R_2(A)$; $A := A + 200$; $W_2(A)$;
$R_1(B)$; $B := B + 100$; $W_1(B)$; $R_2(B)$; $B := B - 200$; $W_2(B)$;

serialisierbar ist! Ist F_1 mit 2PL realisierbar?

Übung 6–8

Zeigen Sie, daß die Ablauffolge

F_2: $R_1(A)$; $A := A - 100$; $R_2(B)$; $B := B - 200$; $W_1(A)$; $W_2(B)$;
$R_2(A)$; $A := A + 200$; $R_1(B)$; $B := B + 100$; $W_2(A)$; $W_1(B)$;

nicht serialisierbar ist! Ist F_2 trotzdem logisch zulässig?

Zeitmarken (time stamps)

Jede Transaktion T_i hat eine eindeutige Zeitmarke $TS(T_i)$, z. B. mittels Rechneruhr oder über einen Zähler. Transaktionen werden so ausgeführt, als ob sie in Zeitmarkenordnung serialisiert wären. Alle Operationen führen die Zeitmarke ihrer Transaktion mit. Speziell kann eine Schreiboperation nie jünger sein als eine vorher erfolgte Leseoperation auf dem selben Datenobjekt.

An jedem Datenobjekt x werden zwei Zeitmarken gespeichert:

- $RTM(x)$ mit der letzten (größten) Lesezeit
- $WTM(x)$ mit der letzten (größten) Schreibzeit

Es gilt nun der folgende einfache *Zeitmarkenalgorithmus*:

Sei TS die Zeitmarke einer Leseoperation auf Datenobjekt *x*.

Wenn TS < WTM(*x*)
dann Zurückweisen der Leseoperation, Zurückrollen
und Neustart der auslösenden Transaktion
mit neuer Zeitmarke
sonst Lesen ausführen, RTM(*x*) := max(RTM(*x*), TS)

Sei TS die Zeitmarke einer Schreiboperation auf Datenobjekt *x*.

Wenn TS < RTM(*x*)
dann Zurückweisen der Schreiboperation,
Zurückrollen und Neustart mit neuer Zeitmarke
sonst wenn TS < WTM(*x*)
dann ignoriere Schreiboperation
sonst Schreiben, WTM(*x*) := TS

Interessant hier das „Wegwerfen“ der veralteten Schreiboperation. Wäre ihr Wert benötigt worden, hätte es einen kleineren RTM(*x*)-Wert geben müssen und die jüngere Schreiboperation wäre nicht zum Zug gekommen.

Generell gibt es Ablauffolgen durch den Zeitmarkenalgorithmus, die durch 2PL nicht erzeugbar wären und umgekehrt.

Übung 6–9

Zeigen Sie, daß F_1 von oben mit dem allg. Zeitmarkenalgorithmus erzeugbar ist, wenn $TS(T_1) < TS(T_2)$!

Optimistische Verfahren

Idee: statt viel vorausschauendem Prüfaufwand bei seltenem, billigem Zurückrollen lieber nachträgliche Prüfung und erhöhten, aber seltenen Rücksetzaufwand.

Transaktionen werden in 3 Phasen aufgeteilt:

- *Lesephase* (Daten aus DBMS lesen und Resultate berechnen im HS)
- *Prüfphase* (Können die Daten zurückgeschrieben werden in serialisierbarer Folge?)
- *Schreibphase* (nur wenn Prüfung ok, sonst Zurückrollen und Neustart).

Zeitmarken für die Phasen werden mit den Transaktionen geführt, nicht bei den Objekten. Die von einer Transaktion betroffenen Datenobjekte bilden Mengen und die Prüfphase stellt sicher, daß für die Schnittmengen der Datenobjekte keine überlappenden Schreiboperationen stattfinden. Die Details seien hier übergangen.

Optimistische Verfahren eignen sich besonders für Anwendungen mit vielen Nur-Lese-Transaktionen. Insgesamt aber haben sich die angekündigten Verbesserungen durch optimistische Verfahren als zu optimistisch erwiesen.

6.4 Präzisierungen der Nebenläufigkeitsaspekte

Bei der Definition der Äquivalenz zweier Ablauffolgen (schedules) auf Seite 131 haben wir verlangt, daß

- Jede der Leseoperationen in den Folgen Datenwerte liest, die von den selben Schreiboperationen in beiden Folgen erzeugt wurden.
- Die letzte Schreiboperation für jedes Datenobjekt die gleiche in beiden Folgen ist.

Tatsächlich bezeichnet man dies als *Sichtserialisierbarkeit*. Daneben gibt es eine zweite Serialisierbarkeit, die *Konfliktserialisierbarkeit*, die nur *konfliktäre Operationen* betrachtet, die wir im Zusammenhang mit Abhängigkeitsgraphen betrachtet haben, also Operationenpaare zweier Transaktionen auf dem selben Datenobjekt, von denen mindestens eine Operation eine Schreiboperation ist.

Beide Formen der Serialisierbarkeit sind nicht ganz gleichwertig. Sichtserialisierbarkeit ist mächtiger (erlaubt mehr gültige Ablauffolgen) als Konfliktserialisierbarkeit, ist aber in der Praxis nicht effizient umsetz-

bar. Deshalb haben wir auch bisher Serialisierbarkeit definiert über die Sichtserialisierung, tatsächlich aber die Einhaltung der Konfliktserialisierbarkeit geprüft.

Ferner muß man genauer zwischen *Recovery* (*Wiederaufsetzen*) und *Concurrency Control* (*Steuerung der Nebenläufigkeit*) für Historien unterscheiden, den beides sind orthogonale Konzepte.

Die Abb. 6–2 aus [KE96] zeigt die Beziehung der Historienklassen zueinander.

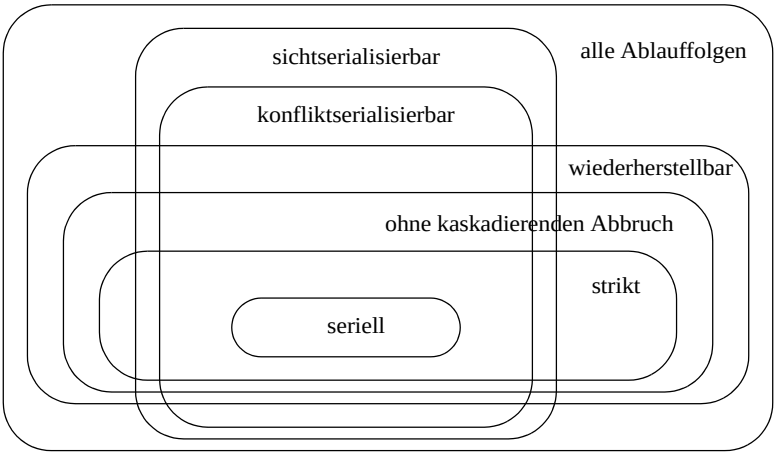


Abb. 6–2 *Beziehungen zwischen den Historienklassen*

Bei *Historien ohne kaskadierenden Abbruch* geben Transaktionen Änderungen an Datenwerten erst **nach dem commit** frei (kein *dirty read*). Bei *strikten Historien* dürfen auch veränderte Daten einer noch laufenden Transaktion T_j nicht überschrieben werden, d. h. macht T_j eine Schreiboperation $w_j(A)$ auf Datum A, und soll darauf eine Lese- oder Schreiboperation $o_i(A)$ einer Transaktion T_i folgen, muß T_j zuerst abort oder commit machen.

Die größtmögliche Übersicht zum Thema Transaktionen gibt auf 1070 Seiten das Buch von Jim Gray und Andreas Reuter [GR93]. Darin wird auch der Isolierungsgrad eines Transaktionssystems definiert. Die Tabelle 6–1 auf S. 142 [GR93, S. 401] faßt die Definitionen zusammen. Der SQL2 Standard erlaubt die Angabe der Isolationsgrade:

```
SET TRANSACTION ISOLATION LEVEL
            [ READ UNCOMMITTED ]
            [ READ COMMITTED ]
            [ REPEATABLE READ ]
            [ SERIALIZABLE ]
```

Table 7.9: Summary of isolation degree attributes.

Issue	Degree 0	Degree 1	Degree 2	Degree 3
Common Name	Chaos	Browse	Cursor stability	Isolated serializable repeatable reads
Protection Provided	Lets others run at higher isolation	0° and no lost updates	No lost updates No dirty reads	No lost updates No dirty reads Repeatable reads
Committed Data	Writes visible immediately	Writes visible at EOT	Same as 1°	Same as 1°
Dirty Data	You don't overwrite dirty data	0° and others do not overwrite your dirty data	0°, 1°, and you don't read dirty data	0°,1°,2° and others don't dirty data you read
Lock Protocol	Set short exclusive locks on data you write	Set long exclusive locks on data you write	1° and set short share locks on data you read	1° and set long share locks on data you read
Transaction Structure	Well-formed WRT writes	Well-formed WRT writes and two-phase WRT writes	Well-formed and two-phase WRT writes	Well-formed and two-phase
Concurrency	Greatest: only set short write locks	Great: only wait for write locks	Medium: hold few read locks	Lowest: any data touched is locked to EOT
Overhead	Least: only set short write locks	Small: only set write locks	Medium: set both kinds of locks but need not store read locks	Medium: set and store both kinds of locks
Rollback	Undo cascades can't rollback	Undo incomplete transactions	Same as 1°	Same as 1°
System Recovery	Dangerous updates may be lost and violate 3°	Apply log in 1° order	Same as 1°	Same as 1° or can rerun in any <<< order
Dependencies	None	WRITE → WRITE	WRITE → WRITE WRITE → READ	WRITE → WRITE WRITE → READ READ→ WRITE

Tab. 6–1 *Übersicht über die Isolationsgrade*

Dabei entspricht `Read Uncommitted` Grad 1 (browse) und ist nur für Read-only Transaktionen gestattet. `Read Committed` bedeutet Cursorstabilität (Grad 2), `Repeatable Read` ist fast Grad 3, `Serializable` ist dann voller Grad 3 (Isolation).

6.5 Zusammenfassung

- 3 Klassen von Synchronisationskontrolle:
  - 2-Phasen Sperren
  - Zeitmarkenalgorithmus
  - optimistische Verfahren
- 2 Methoden um nichtserialisierbare Ausführungen zu vermeiden:
  - Warten, typisch für 2PL, dann Verklemmungsentdeckung oder Vermeidung; erst bei Verklemmung Neustart
  - Neustart sofort für Zeitmarken- und optimistische Verfahren
- Konflikterkennung zwischen Transaktionen:
  - Schritt für Schritt (2PL und Zeitmarkenverfahren)
  - global (optimistische Verfahren)
- Serialisierung über:
  - Zugriffsfolge auf Datenobjekte (2PL: T_i vor T_j wenn T_i etwas vor T_j erlangt, das nicht beide belegen können)
  - Transaktionszeitmarken

7 Physische Datenorganisation

7.1 Grundlagen der Speichermedien

Wesentliches Element des ANSI-SPARC Architektur ist die Trennung des physischen (internen) Modells, das die Speicherorganisation und Zugriffsstrukturen definiert, von dem konzeptuellen Modell.

Kenntnisse der physischen Datenorganisation sind nützlich für

- Auslegung eines DBMS (Plattengrößen, etc.)
- Bestimmung von Indexen (SQL: create index ...)
- Unterstützung optimaler Ballung der Daten (clustering)
- grundsätzliche Vorstellungen über Leistungsfähigkeit von Speicher- und Zugriffsformen.

Grundlage aller Überlegungen ist die Speicherhierarchie mit

- Primärspeicher
  flüchtig, schnell, begrenzte Kapazität, gleichteurer Zugriff auf alle Adressen, kleine Granularität des Zugriffs (Byte, Wort)
- Sekundärspeicher
  üblicherweise Platten, billiger als HS, größer, sehr viel langsamer (meist mind. um Faktor 10^5), große Granularität des Zugriffs (z. B. seitenweise mit 4-8 KB je Seite), unterschiedlich teurer Zugriff je nach Distanz zum vorherigen Zugriff, nicht flüchtig (aber auch nicht 100 % ausfallsicher!).

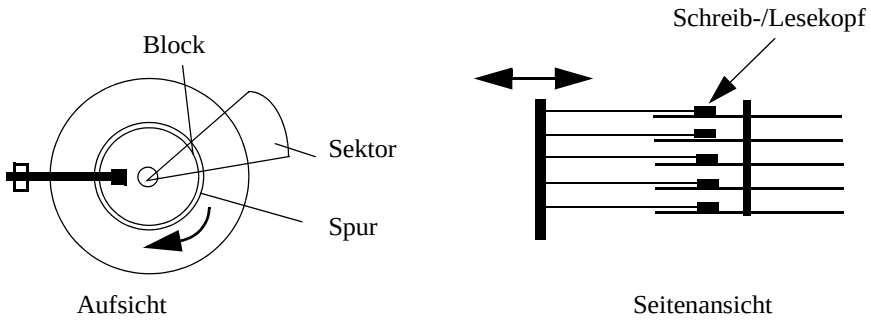


Abb. 7–1 *Schematischer Aufbau einer Festplatte*

Die Abb. 7–1 zeigt den schematischen Aufbau einer Festplatte. Übereinanderliegende Spuren bilden einen sog. Zylinder. Alle Spuren eines Zylinders sind gleichzeitig im Zugriff.

Die gesamte Zugriffszeit auf einen Datenblock setzt sich zusammen aus der Kopfbewegungszeit (*seek time*), der Drehwartezeit (*latency time*) und der eigentlichen Datenübertragungszeit (*transfer time*), die wie die Drehwartezeit abhängig von der Umdrehungsgeschwindigkeit ist. Daneben fällt Zeit für das Übertragungsprotokoll und die Operationen des Adapters an.

Neben Primär- und Sekundärspeicher werden häufig Bänder (Kassetten, usw.) als off-line Archivierungsspeicher eingesetzt. Man spricht dann von *Tertiärspeichersystemen*.

Übung 7–1

Der HS-Zugriff dauere 70 ns, der Plattenzugriff 14 ms. Um welchen Faktor ist der Plattenzugriff langsamer? Stellen Sie den Vergleich an zu einem Zugriff auf einen Eintrag in einem Buch auf dem Schreibtisch (20 s) und den Zugriff über Fernleihe in der Bibliothek!

Übung 7–2

Wenn der Zugriff auf Information über Fernleihe so viel langsamer ist, worauf wird man dann besonders achten? Wie lassen sich die Aussagen auf Plattenzugriffe übertragen?

Übung 7–3

Wie groß ist bei zufälligem Zugriff die Drehwartezeit?

7.2 Mehrfachplatten (disk arrays)

In größeren DBMS werden in der Regel mehrere Platten zum Einsatz kommen. Neben der simplen Vergrößerung der Speicherkapazität werden Mehrfachplatten heute auch eingesetzt zur

- Beschleunigung des Zugriffs (*data striping*, Verteilung der Daten auf unabhängige Platten, paralleler Zugriff) und zur
- Datensicherung durch redundante Speicherung.

Der damit verbundene Begriff RAID (redundant array of independent disks) stammt aus dem Jahr 1988[1] und bezeichnet mit einer nachgestellten Ziffer (RAID0 bis RAID6) einen Grad der Parallelität.

Die Aufteilung des logisch einheitlichen Datenraums auf mehrere Platten kann dabei durch

- unabhängige Adressierung jeder Platte (konventioneller Ansatz) oder durch
- parallele Verteilung auf mehrere Platten (disk striping, data ~) erfolgen.

Der letzte Ansatz ermöglicht häufig bessere Lastverteilung und verbessertes Zugriffsverhalten und ähnelt der Technik der Speicherverschränkung

1. D. Patterson, G. Gibson, and R. Katz, „A Case for Redundant Arrays of Inexpensive Disks (RAID),“ Proc. ACM SIGMOD, Int. Conf. Management of Data, ACM Press, New York, May 1988, pp. 109-116

(*memory interleaving*) auf *N* Speicherbänke mittels einer *modulo N* Adreßspaltung.

Übung 7–4

Warum wird disk striping praktisch immer mit einer redundaten Speichertechnik verbunden? Hinweis: Man beachte die Ausfallwahrscheinlichkeit für ein System mit *N* unabhängigen Platten und die Auswirkungen des Ausfalls auf den Gesamtdatenbestand!

Die Abb. 7–2 zeigt drei möglichen Aufteilungen des Datenraums:

a) mit unabhängigen Platten
b) streifenweise Aufteilung mit kleiner Granularität
c) streifenweise mit grober Granularität.

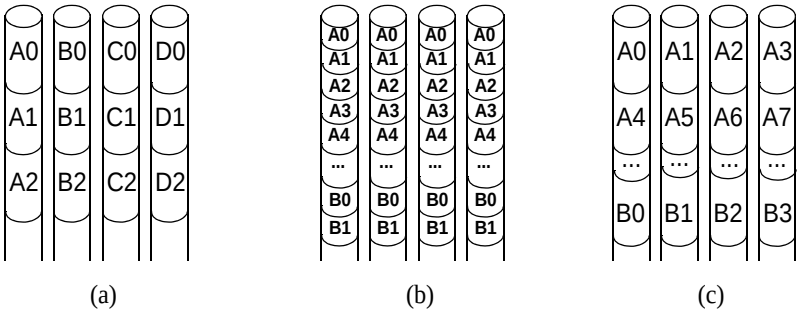


Abb. 7–2 *Verschiedene Aufteilungen des Datenraums*

Im Fall (a) muß der Datenadministrator die Verteilung der Daten (Tabellen, Indexe, usw.) nach geschätzten Zugriffshäufigkeiten selbst planen. Die Aufgabe ist allerdings nicht effizient lösbar, da das Problem NP-vollständig ist.

Im Fall (b) kann die Einheit der Verteilung 1 bit, 1 byte oder ein Sektor (512 bytes) sein. Die Lastverteilung auf die *N* Platten ist absolut gleichmäßig, die Transferrate ist um den Faktor *N* höher als bei der einzelnen Platte, die Positionierungs- und Drehwartezeit ist gleich oder geringfügig höher als bei einer einzelnen Platte. Allerdings kann jetzt nur ein Zugriffs-

wunsch zu einem Zeitpunkt erfüllt werden, da alle *N* Platten davon betroffen sind. Damit bietet sich dieses Schema für Anwendungen an, bei denen die Transferzeiten die Servicezeiten dominieren.

Im Fall (c) kooperieren nicht notwendigerweise alle Platten bei einem Zugriffswunsch. Bei großen Transferwünschen werden viele oder alle Platten parallel übertragen, während kleinere Wünsche unabhängig und gleichzeitig ausgeführt werden können. Dadurch stellt sich eine natürliche, zufällige Lastverteilung ein, weshalb auch die Autoren in [GWHP94] dieser Aufteilung das größte Potential bescheinigen.

Wie in der Übung oben angesprochen, erhöht sich die Ausfallwahrscheinlichkeit des Gesamtsystems ohne Redundanz bei Annahme unabhängiger, exponentialverteilter MTTF (mean time to failure) umgekehrt proportional zur Anzahl der Platten. Liegt die MTTF einer Platte heute bei etwa 200 000 bis 1 Million Betriebsstunden (also rund 20 - 100 Jahre), verkürzt sich die MTTF eines nicht-redundanten Systems mit einem Dutzend Platten auf Monate oder gar Wochen. Diese Tatsache zusammen mit fallenden Speicherpreisen wird zunehmend zu Plattensystemen mit fest eingebauter Redundanz führen.

Die einfachste und älteste Form der Redundanz ist das Halten einer zweiten oder weiterer (allgemein *D*) Kopie auf unabhängigen Platten. Diese Technik ist unter den Namen Plattenspiegeln (disk mirroring), Schattenplatten (disk shadowing), Plattenduplex und neuerdings als RAID1 bekannt. Die Kosten dieser Redundanz sind grundsätzlich proportional zu D-1. Wegen der hohen Kosten wird man auch selten mit mehr als zwei ($D = 2$) Kopien arbeiten. Dabei zählt als Kosten besonders auch der vervielfachte Schreibaufwand.

Die Abb. 7–3 zeigt drei Verteilungsschemata für die einfache Replikation.

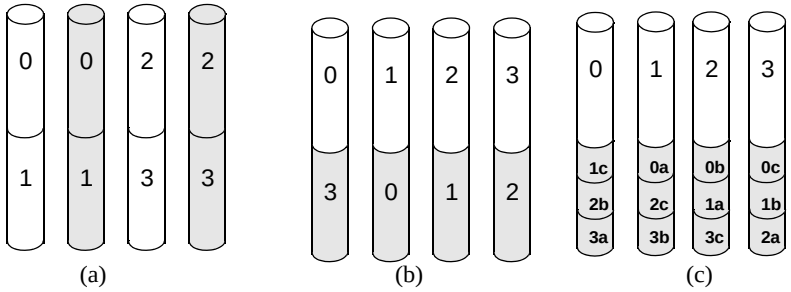


Abb. 7–3 *Verteilungsschemata für die einfache Replikation*

Im Fall des traditionellen Spiegelns (a) kann das System den Ausfall von *D*-1 identischen Platten verkraften. Auch kann der Ausfall von mehren Platten noch zu einem Überleben des Gesamtsystems führen, solange darunter nicht *D* Platten des selben Inhalts sind. Allerdings verteilt sich nach einem Ausfall die Last ungleichmäßig, wodurch die beschädigte Teilmenge zu einem Flaschenhals des Systems werden kann.

Um diesen Effekt auszugleichen kann man sog. Entzerren (*declustering*) verwenden, im Fall (b) sog. *chained declustering*. Dabei wird die Platte in *D* Sektionen aufgeteilt mit den Primärdaten in der ersten Sektion und Kopien in den folgenden Sektionen, wobei die Kopien gerade um eine Platte immer versetzt gespeichert sind.

Bei einem Ausfall kann durch eine intelligente Zugriffsplanung die Lastverteilung besser aufrechterhalten werden. Allerdings verringert sich die Ausfallsicherheit. Dies gilt besonders im Fall (c), dem sog. *interleaved declustering*, bei dem die 2. Kopie jeder Platte in mehrere Sektionen zerlegt über alle anderen Platten verteilt wird.

Übung 7–5

Seien die Platten in der Abbildung oben von links nach rechts mit Nr. 1 bis Nr. 4 durchnumeriert. Im Fall (a) kann nach Verlust von Nr. 1 noch der Verlust von Nr. 3 oder Nr. 4 toleriert werden. Wie sieht es bei (b) und (c) aus?

Der große Platzaufwand bei der Replikation läßt sich vermindern - meist zu Lasten eines größeren Schreib-/Leseaufwands - durch Halten von Paritätsdaten, ähnlichs den fehlerentdeckenden und fehlerkorrigierenden Codes in der Datenübertragung.

Übung 7–6

Ein besonders einfaches Schema verwendet im Prinzip drei Platten, wobei die dritte Platte das XOR-Bit der beiden entsprechenden Bits der beiden anderen Platten speichert. Überzeugen Sie sich, daß bei Verlust einer der drei Platten die fehlende Information aus den verbleibenden beiden Platten rekonstruiert werden kann! Wie hat sich der Platzbedarf gegenüber reiner Replikation verringert? Welchen Aufwand erfordern Schreiboperationen?

Die drei hauptsächlichen Möglichkeiten, die Paritätsdaten zu speichern, werden in der Abb. 7–4 gezeigt, wobei (a) eine eigene Paritätsplatte, (b) Streifenparität (*striped parity*) und (c) entzerrte Parität (*declustered parity*) darstellt. Ap repräsentiert den Paritätsblock der Datenbits im Block A.

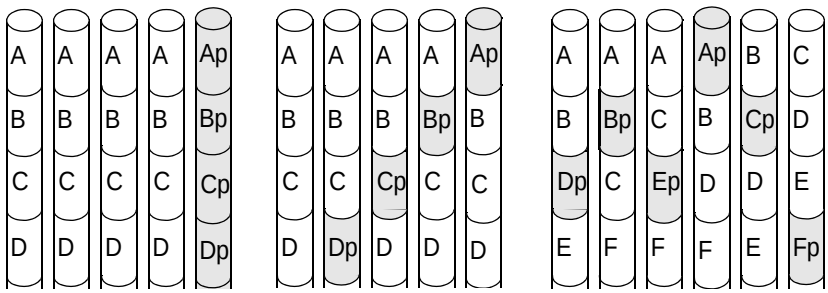


Abb. 7–4 *Anordnung der Paritätsdaten*

Das Schema mit eigener Paritätsplatte verlangt einen Zugriff auf diese Platte für alle Schreiboperationen auf den anderen Platten. Es funktioniert demnach nur gut in Verbindung mit der streifenweisen Verteilung bei klei-

ner Granularität (Fall (b) in Abb. 7–2), da hier nur jeweils ein Zugriff zu einem Zeitpunkt möglich ist und dieser alle Platten gleichzeitig betrifft.

Üblicherweise tolerieren die obigen Anordnungen nur den Ausfall einer Platte. Bei Einsatz von zwei Paritätsbits und Verwendung von 2-Reed-Solomon-Codes können auch der Ausfall zweier beliebiger Platten abgefangen werden. Andere Codes können ggf. auch noch höhere Ausfallsicherheit bieten, allerdings wird der Speicheraufwand dann unverhältnismäßig hoch, zumal die Zuverläßigkeit der Platten durch technische Maßnahmen von Jahr zu Jahr steigt.

Zu beachten ist ferner der Wiederherstellungsprozeß (*rebuild*) nach Ausfall einer Platte, zumal das System dann keinen weiteren Ausfall vertragen kann. Die Wiederherstellung kann während Aufrechterhaltung des laufenden Betriebs oder auf einen Schlag bei Unterbrechung des Betriebs erfolgen. Häufig werden dabei zusätzliche Platten aktiviert, die vorher leer mitliefen (*hot spares*).

Die folgende Matrix aus [GWHP94] zeigt mögliche Kombinationen der besprochenen Techniken. Man beachte dabei, daß die sog. RAID-Ebenen nicht logisch aufeinander aufbauen und auch nicht notwendigerweise einen Qualitätsmaßstab abgeben. Ein Eintrag U/C bedeutet dabei „unwahrscheinliche Kombination“.

Datenorganisation		**Unabhängige Adressierung**	**Streifenweise Aufteilung, kleine Granul.**	**Streifenweise Aufteilung, große Granul.**
keine Redundanz		konventionell	synchronized disk interleaving (Kim 1986)	RAID0 (Livny, Koshafian, Boral 1987)
Replikation	2 Kopien	Spiegeln, RAID 1	-	RAID0+1
	N Kopien	Schattenplatten	-	-
	entzerrt	chained, interleaved	U/C	-

Datenorganisation		Unabhängige Adressierung	Streifenweise Aufteilung, kleine Granul.	Streifenweise Aufteilung, große Granul.
Parität	Platte	-	RAID3 (Kim 1986)	RAID4 (Ouchi 78)
	Streifen	Gray et al. 1990	U/C	RAID5
	entzerrt	-	U/C	Holland et al. 1992
Two-Reed-Solomon Code		-	-	RAID5+; RAID6
Hamming Code (ECC)		-	RAID2	-

Tab. 7–1 *Matrix der Datenorganisation (aus [GWHP94])*

Weitere (neue?) RAID-Begriffe werden in letzter Zeit auch von Herstellern für ihre Produkte eingeführt und sind in der Tabelle nicht aufgeführt.

7.3 Der Datenbankpuffer

In Kapitel 1 hatten wir auf die neuere Technik der „memory mapped files“ aufmerksam gemacht, mit der sich die Technik des *Pointer Swizzling* realisieren läßt.

In der klassischen Datenbanktechnik wird hiervon kein Gebrauch gemacht. Vielmehr findet ein Kopier- und ggf. Übersetzungsprozeß zwischen den persistenten Daten auf der Platte und der DB-Anwendung im HS statt. Hierzu werden die Daten von der Platte in spezielle HS-Bereiche, die sog. *Puffer,* gelesen und dort direkt oder nach einem nochmaligen Kopiervorgang in den Anwendungsbereich bearbeitet und anschließend zurückgeschrieben.

Ob sich das DBMS dabei auf das Dateisystem des Betriebssystems abstützt, indem es je Tabelle und je Index eine oder mehrere Dateien anlegt, oder ob es eine große Datei belegt und diese intern unterteilt, oder

ob es gar eine Partition beansprucht und diese als *raw device* verwaltet, ist für die folgende Diskussion unerheblich.

Übung 7–7

Wie wirken sich die gerade angesprochenen Alternativen auf die Sicherheit, Portabilität, Geschwindigkeit aus?

Gleichermaßen sei offengelassen, ob die DBMS-Puffer identisch sind mit den E/A-Puffern oder hierzwischen nochmals ein Kopiervorgang stattfindet. E/A-Puffer werden vom E/A-Untersystem des Betriebssystems (den Kanälen, bzw. dem DMA-Prozessor bei Microcomputern) genutzt, um überlappend mit der Tätigkeit der CPU Daten von der Peripherie in den HS, bzw. umgekehrt, zu übertragen.

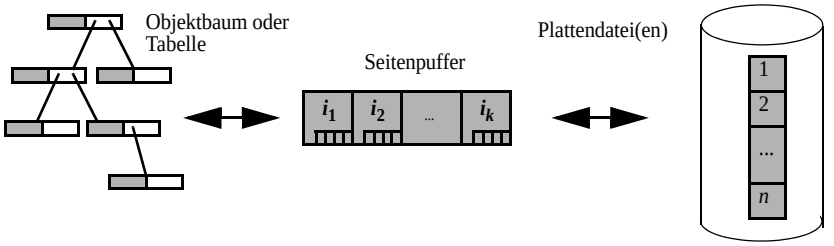


Abb. 7–5 *Klassisches DBMS-Speichersystem*

Festzuhalten ist lediglich, daß das DBMS einen meist zusammenhängenden Bereich von *k* Blöcken (Seiten) verwaltet, der *Pufferrahmen* heißt. In diesen Rahmen werden Seiten geladen und aus ihm müssen Seiten verdrängt werden, wenn für eine neue Seite kein Platz mehr vorhanden ist.

Diese Aufgabe ist analog zur Seitenverdrängung im virtuellen Speicher in einem BS mit *paging*. Hier wie dort wird die Strategie sein, die Seite zu verdrängen, die man die längste Zeit nicht mehr benötigen wird (Belady's Strategie). Diese nicht implementierbare Strategie kann mittels LRU (least recently used), FIFO (first in first out), Second Chance, oder Zählern (simulierte Uhr) angenähert werden.

Übung 7–8

In einem Seitenrahmen für 3 Seiten werde die folgende Zugriffsfolge abgearbeitet.

```
1, 2, 2, 5, 3, 4, 2, 3, 4, 5, 1, 4, 5, 2, 1, 3
```

Man beschreibe den Verdrängungsprozeß bei Einsatz von Belady's Strategie, LRU, FIFO und Zählern an jeder Seite, die die Zugriffshäufigkeit festhalten.

Wie beim Paging gibt es an den Seiten Flags zur Angabe, ob eine Seite modifiziert wurde (nach schreibendem Zugriff setze *dirty bit* = 1) und ob sie im HS gehalten werden muß (*fix bit*).

Im Zusammenhang mit Transaktionen wurde auch daraufhingewiesen, daß ein Schreiben in die Puffer hinein nicht gleichzeitig ein Sichern der Daten auf der Platte bedeutet. Erst wenn die Puffer zwangsweise herausgeschrieben wurden (*forced write*, *flush*) und der Schreibvorgang vollständig erfolgte, kann man davon ausgehen, daß die Änderungen persistent wurden[1]. Für die folgende Matrix bezeichne *force* eine Strategie, bei der mit dem **commit** alle modifizierten Seiten sofort auf die Platte geschrieben werden. Mit ¬*force* werde eine Strategie bezeichnet, die diese Änderungen nicht sofort in die materialisierte Datenbasis einbringt, weshalb die Log-Datei hierfür auch besondere Redo-Einträge braucht.

Die Strategien *force* und ¬*force* lassen sich kombinieren mit *steal* und ¬*steal*, wobei letztere Alternative andeutet, ob eine Transaktion modifizierte Seiten einer anderen, noch aktiven Transaktion verdrängen (stehlen) darf, oder nicht.

Nach diesen Überlegungen erscheint besonders die Kombination *force* mit ¬*steal* günstig zu sein, da alle Änderungen abgeschlossener Transaktionen und keine Änderungen noch laufender Transaktionen dauerhaft gespeichert sind. Allerdings benötigt diese Strategiekombination ähnlich

1. Werden die Seiten nicht automatisch an die selbe Stelle zurückgeschrieben, sondern gibt es eine Zuordnungstabelle logische Seitennummer:physische Seitennummer, dann ist der Schreibvorgang eigentlich nur dann abgeschlossen, wenn auch dieser Teil der Zuordnung persistent gemacht wurde.

wie die write-through-Strategie bei Cache-Speichern eine hohe Bandbreite, da laut [KE96] viele Seiten geschrieben werden, ohne daß es zu ihrer Verdrängung kommt. Ferner erlaubt diese Strategie nur das Sperren ganzer Seiten und behindert dadurch die mögliche Parallelität bei Transaktionen. Zuletzt ist noch darauf zu achten, daß Sicherungsmaßnahmen getroffen werden, damit alle Schreibvorgänge einer Seite *atomar* (alles oder nichts) ablaufen.

	force	**¬force**
¬steal	kein Redo kein Undo	Redo kein Undo
steal	kein Redo Undo	Redo Undo

Tab. 7–2 *Pufferstrategien*

Eine Möglichkeit, atomares Schreiben der Seiten sicherzustellen, ist für jede Seite zwei Exemplare (twin blocks) vorzuhalten und erst nach dem Schreiben der Seite das aktuell-bit auf diese Seite zu setzen. Nachteil hier ist die Verdopplung des Speicherplatzes. Das in Kapitel 6 genannte Schattenseiten-Verfahren vermeidet diesen Nachteil, indem nur für die gerade modifizierten Seiten eine Kopie existiert, hat aber so gravierende andere Nachteile, daß sein Einsatz laut [KE96] heute nicht mehr aktuell ist. Somit verwenden fast alle DBMS heute eine (direkte) Einbringstrategie, die als *update-in-place* bezeichnet wird.

7.4 Die Adressierung von Datenobjekten

Für die Abspeicherung einer Relation (Tabelle) kann man im einfachsten Fall eine Datei mit wahlfreiem Zugriff anlegen, deren Sätze (*Records*, Blöcke) gerade die *Tupel* aufnehmen, wobei man vom Regelfall ausgeht, nachdem alle Tupel die selbe Länge haben. Das Betriebsystem errechnet dann bei einem Zugriff auf das *i*'te Tupel (`seek` *i*) die Startadresse in der Datei aus (*i*-1)*Satzlänge und positioniert einen logischen Lese-/Schreibzeiger dorhin.

Diese direkte Implementierung wird man nur bei den einfachsten DBMS finden. Aufgrund der Anforderungen an den Mehrbenutzer- und Transaktionsbetrieb, also für Log-Dateien, variabel lange Felder (z.B. char-Attribute, Multimedia-Daten, Null-Werte), Objektmehrfachnutzung mit invarianten Objektbezeichnern, usw. werden nur die Seiten selbst (da fester Länge) in der oben angedeuteten Weise verwaltet, also als Folge von Blöcken einer Datei.

Die Zuordnung Tupel ↔ Seite, oder allgemein Datenobjekt ↔ Seite, nimmt dann das DBMS selbst vor, wobei es darauf achten wird, daß ein Tupel sich nicht über mehr als eine Seite erstreckt. Somit taucht die Frage auf, wie das DBMS seine Datenobjekte identifiziert und diese auf Plattenadressen, d.h. eine Seitennummer und ein Byte-offset in dieser Seite abbildet.

Hierzu bestehen drei Möglichkeiten:

- Es gibt auch auf interner Systemebene keine eindeutigen Objektidentifikatoren; man arbeitet mit relativen Satznummern (RSN), d. h. es gibt ein 1., 2., ..., allgemein *n*-tes Objekt (Tupel einer Relation) und man kann einen Cursor darauf setzen oder es wird auf Wunsch in einen speziellen Lese-/Schreibbereich gebracht. Als Folge einer Suche nach Inhaltskriterien erhält man die RSN des gesuchten Satzes, die jedoch nicht invariant ist (Einfügen oder Entfernen davor ändert die RSN!).
- Der Objektidentifikator ist gleichzeitig die physische Adresse. Man spricht dann auch von TID (tuple identifier) oder RID (record identifier). Müssen Objekte physisch ihren Platz tauschen, z. B. weil sie wachsen und nicht mehr an den alten Platz passen oder als Folge einer Neuorganisation zur Verbesserung des Lokalitätsverhaltens, so müssen nach außen die RIDs unverändert (invariant) bleiben, etwa wenn ein Index darauf bezug nimmt.
- Man unterscheidet logische OIDs von physischen Plattenadressen und stellt die Zuordnung über eine Tabelle her. OIDs bleiben für die Lebensdauer eines Objekts unverändert und werden auch nicht recycled. Wandert ein Objekt an eine andere Stelle, ändert man den

Eintrag in der Zuordnungstabelle. Diese Sicht wird von den meisten OO-DBMS eingenommen.

Die erste Organisationsform ist sehr inflexibel. Zur Herstellung einer Beziehung zwischen einem Schlüssel, z. B. einer Artikelnummer, und der RSN könnte man sich vorstellen

- die Datei nach dem Schlüssel zu sortieren; die Datei wirkt dann wie eine lineare, sequentiell gespeicherte Liste, d. h. man kann binär Suchen, aber Einfügen/Entfernen ist nur schlecht möglich.
- Man verwendet die RSN als Schlüssel $\rightarrow$ sehr inflexibel (Artikelnummer 1, 2, ... ?).
- eine Tabelle für die Zuordnung anzulegen; verlagert das Problem auf die Tabellenorganisation.
- eine Schlüsseltransformation vorzunehmen, z. B. Hashing. Hierzu später mehr.

Die zweite Organisationsform fand erstmals beim System R Anwendung. Wegen der interessanten Details (und weil wir sie in Kassel nachimplementiert haben) sei sie im folgenden Abschnitt im Detail dargestellt.

7.5 Das RID-Konzept

Ein Record Identifier (RID) ist eine Plattenadresse, die sich aus der Seitennummer, in der sich der Record (das Objekt, das Tupel) befindet, und einem kleinen Indexwert zusammensetzt. Der Index zeigt in eine kleine Tabelle, die sich in der Seite selbst befindet.

Durch diesen Index erreicht man die Verschieblichkeit der Bytestrings innerhalb der Seiten. Übliche Aufteilungen für 4-Byte-RIDs sind drei Bytes für die Seitennummer und ein Byte für den Index. Die Abb. 7–6 zeigt das grundlegende Schema, wie es in fast allen DB-Büchern beschrieben wird.

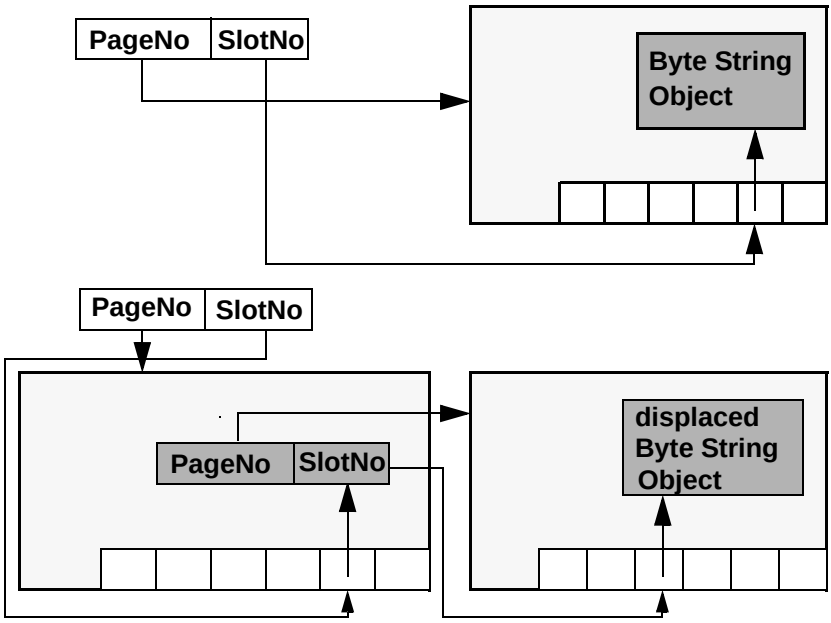


Abb. 7–6 *Das RID-Konzept*

Das Bild läßt auch erkennen, wie bei einem notwendigen Umzug eines Objekts zu verfahren ist: der ursprüngliche RID bleibt bestehen und verweist auf einen „Umzugs-RID“ (displacement RID), der die neue Plattenadresse darstellt. Muß nochmals umgezogen werden, ändert man den RID am Ursprungsort, d.h. man stellt sicher, daß jedes Objekt mit **maximal zwei** Seitenzugriffen gefunden und geladen wird.

Andere Details sind nicht bekannt und nach einer Aussage eines der Implementierer [Lindsay92] auch nie publiziert worden. Zu diesen Details gehört die Fragen:

- Wann wird innerhalb einer Seite geschoben (garnicht, immer, bei einer Reorganisation)?
- Wie stellt man die Länge eines Objekts fest (Abspeicherung von Längenangaben)?
- Was ist die kleinste Objektgröße?

- Wie kann man ein Umzugs-RID von einen „normalen“ Objekt unterscheiden?
- Weiß ein umgezogenes Objekt wie sein ursprüngliches RID heißt (Rückwärts-RIDs)?
- Wie groß ist ein Eintrag in der Tabelle?

Übung 7–9

Was ist die maximale Dateigröße bei 4 Byte RIDs mit 3 Byte Seitennummer und 4 KB (8 KB) Seitengröße?

Übung 7–10

Wenn, wie in der Kasseler Implementierung, 2 Bytes je Sloteintrag (Tabelleneintrag) vorgesehen sind, davon 3 Bit für Flags reserviert sind, wie groß kann dann eine Seite werden? Wie ändert sich die Seitengröße, wenn alle Bytestrings auf Wortgrenzen (Bytegrenze mod 4 = 0) anfangen?

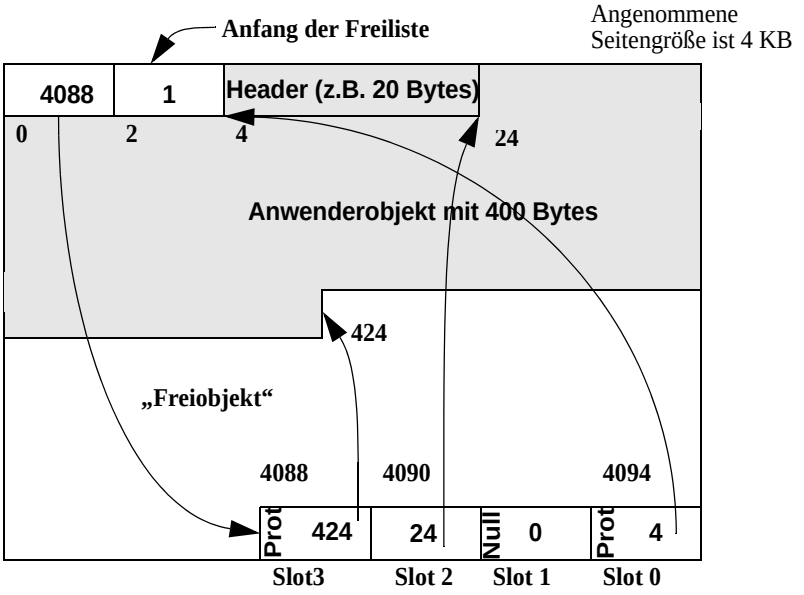
Wie in der Übung oben angedeutet, arbeitet die Kasseler Implementierung mit drei Bits für Flags, deren Belegung sich aus der Tabelle 7–3 auf S. 161 ergibt. Im einzelnen bedeutet „protected“, daß es sich um ein internes Objekt handelt. Dazu zählen Freispeicherlisten, der Eintrag im Seitenkopf jeder Seite (z.B. mit Zugriffsdatum), der freie Platz in einer Seite. Damit können „falsche“ RIDs abgefangen werden, die sonst entweder unsinnige Bytefolgen abliefern würden oder die bei einem Schreibzugriff durch unbefugte Anwender das Dateisystem korrumpieren würden.

Daneben werden Umzugs-RIDs markiert und es werden übergroße Bytestrings (solche die nicht in eine Seite passen) bis zu einer Gesamtgröße von 2 GB verwaltet. Gelöschte Objekte führen ferner zu einem leeren Slot, die untereinander verkettet sind, um schneller an freie Slots zu kommen. Datenobjekte werden sequentiell in der Reihenfolge der Sloteinträge abgelegt und führen bei Größenänderungen zum Verschieben aller Objekte (und Änderung der Sloteinträge) vor dem sich ändernden Objekt. Die Länge eines Objekt s wird nicht gespeichert und ergibt sich aus der Differenz zwischen je zwei (nichtleeren) Sloteinträgen.

Bitposition			Interpretation	Flag-No.
15	14	13		
0	0	0	Normal object (unprotected)	0
0	0	1	Protected normal object	1
0	1	0	Oversize object (unprotected)	2
0	1	1	Protected oversize object	3
1	0	0	Displacement RID (unprotected)	4
1	0	1	Protected displacement RID	5
1	1	0	Empty slot	6
1	1	1	Reserved (treat as empty slot)	7

Tab. 7–3 *Flags der Kasseler RID-Implementierung*

Beispiel 7–1



Übung 7–11

Welches RID hat das Freiobjekt in Beispiel 7.1? Wie groß ist es?

Die Umsetzung eines RIDs *R* in eine Hauptspeicheradresse *A* ergibt sich aus dem folgenden Programmfragment (in PASCAL).

```
PA := PageMap[R shr 8]
if PA = nil then ... page fault ... exit
SlotContent := Slots(PA^.Page)[2047 - (R and $000000FF)]
if SlotContent > $3FFF
then ...   displaced object or large object
           or empty slot ...
else A := Addr(PA^.Page[SlotContent and $1FFF])
```

Auch wenn heute die Trennung von logischen OIDs und physischen RIDs propagiert wird und damit RIDs nicht notwendigerweise invariant bleiben müssen, ist das RID-Schema aufgrund des fast direkten Plattenzugriffs weiterhin attraktiv. In einer jüngeren Arbeit von Wegner/Paul/Thamm/Thelemann [TWPT] wird ferner gezeigt, wie sich das neue Pointer Swizzling mit der alten RID-Technik verbinden läßt, ohne auf memory mapped files zurückgreifen zu müssen.

7.6 Database Keys

Die oben angesprochene dritte Möglichkeit, die Beziehung zwischen Objektidentifier (OID) und Plattenadresse herzustellen, war eine Tabelle. Dieses Adressierungsschema heißt in der Literatur [GR93, S. 763ff] Database Keys und ist Teil des DBTG Datenmodells und wird auch z. B. in ADABAS verwendet.

Die große Tabelle dient praktisch als permanenter Umzugsverweis, wobei man davon ausgehen muß, daß sie nicht vollständig in den Hauptspeicher paßt. Grey und Reuter machen hierzu die folgende Rechnung auf:

Größe der Datei 10^{10} Bytes, durchschnittliche Tupellänge 100 Bytes, jeder Eintrag in der Tabelle 4 Bytes. Damit ist die Tabellengröße $4*10^8$ Bytes oder $5*10^4$ Seiten für 8 KB Seiten. Hat der Pufferpool Platz für

3000 Seiten (24 MB), dann ist die Trefferrate für das Anfinden einer Tabellenseite klein bei zufälligem Zugriff.

Übung 7–12

Wie groß ist die Wahrscheinlichkeit für einen Treffer genau? Wie ändert sich die Situation, wenn die durchschnittliche Tupelgröße 2 KB ist? Wie sieht das Adressierungsschema dann im Vergleich zum RID-Ansatz aus, wenn man davon ausgeht, daß 10% der Tupel umgezogen sind.

7.7 Physische Tupeldarstellungen

Einer der Vorteile des relationalen Modells ist, daß sich ein Tupel mit n Attributen 1:1 in einen Satz mit n Feldern abbilden läßt. Im einfachsten Fall enthält dann der DB-Katalog, d.h. die Tabelle, die alle Relationenschemata aufnimmt, Typinformationen für jede Relation, aus der für das interne Modell die Versetzadresse für jedes Attribut hervorgeht. Diese Information existiert genau einmal für das Schema, nicht für jede Tupelinstanz! Restriktionen der Art „max. 255 Attribute, max. 64 KB Tupelgröße“ lassen auf den Aufbau dieser Schemainformation in einfacheren DBMS schließen.

Tupel mit einer Größe von mehr als einer Seitengröße (typisch 4 KB) benötigen allerdings einen Mechanismus zur Aufteilung auf $k > 1$ Seiten. In diesem Fall und auch wenn Tupel eigentlich immer in eine Seite passen würden, man aber zur Vermeidung von Lücken Tupel auf zwei oder mehr Seiten aufteilt, spricht man von *überspannender Speicherung* (spanning storage), sonst von *nicht überspannender Speicherung*.

Sind die Tupel eher so kurz, daß mehrere in eine Seite passen und auch so gespeichert werden, spricht man von *Blockung*. In diesem Fall geht man auch von Tupeln *fester Länge* aus.

Daneben stehen Tupel mit *variabler Länge*, die fast immer eine überspannende Speicherung verlangen. Tupel variabler Länge können entstehen durch Attribute variabler Länge, z.B. variabel lange Textfelder, durch Tupel mit Variantenstruktur und durch nicht-normalisierte Tupel mit Wie-

derholungsgruppen, z. B. ein Büchertupel mit einer variablen Anzahl von Autoren.

Klassische Datenbanksysteme speichern solche Tupel in zwei Teilen ab. Zuerst wie gehabt die fixen Anteile mit Versetzinformation im Schema, danach die variablen Teile, eingeleitet mit einem Feld von *i* Versetzadressen für bis zu *i* Attribute, wobei dieses Feld bei jeder Tupelausprägung (Instanz) mitgespeichert wird! Eine weitere Möglichkeit ist, daß alle variablen Teile eines Tupels, auch z. B. Multimedia-Daten wie etwa ein Pixelbild, zu einem *binary large object* (BLOB) zusammengefaßt werden, die einfach eine beliebig lange, zunächst uninterpretierte Bytefolge bilden.

Mit der Zunahme der Bedeutung sog. komplexer Objekte, bei denen ein Objekt eine variable Anzahl an Verweisen auf andere Objekte besitzen kann, wird man allerdings Strukturen den Vorzug geben, die auf einheitliche Weise Aggregationen von Bytestrings variabler Länge verarbeiten können. Ein solcher Ansatz wird z. B. in dem Kasseler NF^2-Editor ESCHER verfolgt, indem alle Tupel und alle Relationen nur Felder von Verweisen (RIDs) sind und selbst atomare Werte wie `Integer` oder `Char` nicht direkt im Tupel gespeichert werden.

Noch einen Schritt weiter geht man, wenn in jedem Tupel zu jedem Feld die Typinformation und ggf. sogar der Attributname abgespeichert wird.

Beispiel 7–2

Ein Büchertupel hätte dann z .B. die Form

```
[ISBN:CHAR(13)="0-8362-0415-8",
{[AUTHOR:CHAR(*)="SCOTT ADAMS"]},
TITLE:CHAR(*)="It's Obvious You Won't Survive by Your
Wits Alone", ..., YEAR:INT=1995]
```

Für kommerzielle System ist diese Form der Redundanz allerdings nicht vertretbar.

In jedem Fall sollte die physische Tupeldarstellung das Anfügen von Attributen unterstützen ohne daß die gesamte Relationenabspeicherung reorganisiert werden muß.

Weitere Freiheitsgrade bei der Realisierung der Tupeldarstellung betreffen

- die interne Repräsentation der Attributwerte, die mindestens die von SQL geforderte Präzision haben muß, DB-NULL anzeigen soll und ggf. variabel lange Felder komprimiert
- die Speicherung von Werten im Maschinenformat (binary image) oder in einem externen Zeichenkettenformat, wobei unabhängig vom verwendeten Maschinentyp und der Programmiersprache des DB-Anwendungsprogramms, Vergleiche, arithmetische Operationen, Sortierungen, usw. auf allen Anlagen gleich ausgehen sollten.

Beispiel 7–3

Gray/Reuter weisen z. B. darauf hin, daß Oracle für alle externen numerischen Datentypen nur eine interne Zeichenkettendarstellung in einem etwas seltsamen, „selbstgestrickten“ Gleitpunktformat zur Basis 100 kennt.

Zuletzt sei nochmals auf das Problem der *langen Felder* (long fields) eingegangen.

Große Tupel mit ggf. variabel langen Feldern lassen sich auf verschiedene Art und Weise abspeichern, z. B. durch

- generelleVergrößerung der Seiten auf z.B. 64 KB für alle oder einen Teil der Datenseiten (wie in DB2);
- Aufspaltung der Tupel in den kleinen fixen Teil und den langen variablen und getrennte „Spezialspeicherung“ der variablen Anteile (z.B. Patientendaten mit digitalisierten Röntgenbildern, auf die nur selten zugegriffen wird);
- überspannende Speicherung, wobei der Überlaufteil in einer **Kette** von Überlaufseiten abgelegt wird;

- ~ Überlaufteil in mehreren Seiten, die von einem Mini-Verzeichnis (**Baum**) verwaltet werden, abgelegt wird.

Beispiel 7–4

Der oben genannte DB-Editor ESCHER verwendet die letztgenannte Speicherungsform für große Objekte. Die Abb. 7–7 zeigt ein solches (einstufiges) Mini-Verzeichnis für ein großes Objekt mit 8 000 Bytes aufgeteilt in 3 000, 3 500 und 1 500 Bytes.

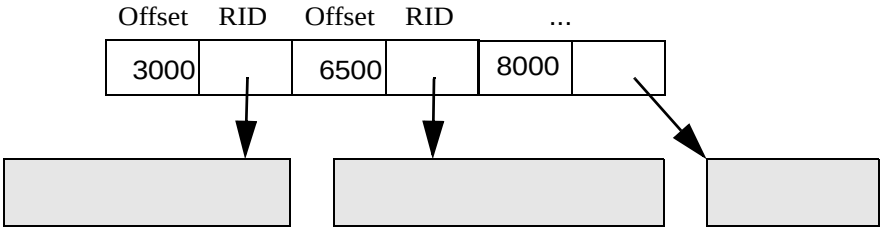


Abb. 7–7 *Mini-Verzeichnis zur Speicherung großer Objekte*

Übung 7–13

Diskutieren Sie das Suchen einer Bytefolge ab Position *x*, Einfügen und Entfernen von Teilstrings und die Generalisierung auf mehrstufige Verweisbäume.

Was sind die Vor- und Nachteile der oben gezeigten Offset-Speicherung gegenüber der Größenspeicherung (3000, 3500, 1500)?

8 Netzwerk-, hierarchisches und objekt-orientiertes Modell

8.1 Das Netzwerkmodell

Das *Netzwerkmodell*, auch *CODASYL-Modell* genannt (nach dem Komitee Conference on Data Systems Languages [COD71]), läßt sich durch die folgenden Eigenschaften beschreiben:

- orientiert an allgemeinem *Graphenmodell*,
- praktisch eingeschränkt auf ER-Modell mit binären 1:n-Beziehungen,
- Entity-Mengen sind sog. (logical) *record types*, dargestellt durch Knoten,
- einzelne Entities sind sog. (logical) *records*,
- Namen der Felder und ihre Typen bestimmen sog. (logical) *record format*,
- Beziehungen dargestellt als gerichtete Kanten,
- realisiert durch (physische) Zeiger (Satzadressen), sog. Links, dem Datenbankäquivalent zum Pointer in Programmiersprachen,
- Zeiger meist abgespeichert mit den Sätzen,
- *navigierender Zugriff*.

Damit gilt ungefähr die Entsprechung

Logisches Satzformat:	Relationenschema
Logischer Satz:	Tupel
Logischer Satztyp:	Relationenname

Übung 8–1

Es gibt mindestens drei Arten, ein Objekt zu identifizieren: über seinen Wert, über seine Adresse (Link), über einen eindeutigen Bezeichner (OID). Diskutieren Sie Vor- und Nachteile der drei Arten!

Links (binäre 1:*n*-Beziehungen) zwischen T_1 und T_2 werden durch einen Pfeil von T_1 nach T_2 dargestellt. In der Abbildung 8–1 geht der Pfeil daher von den Kunden zu den Bestellungen (ein Kunde kann viele Bestellungen haben, jede Bestellung muß genau einem Kunden gehören). Bei 1:1-Beziehungen ist der Pfeil beliebig.

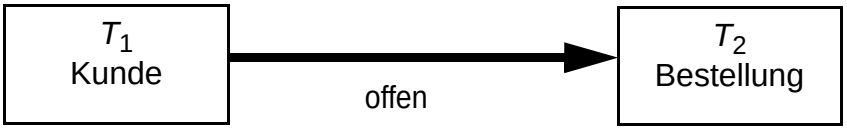


Abb. 8–1 *Darstellung eines Links als Pfeil*

Hat in einer *is-a-Beziehung* die Untermenge keine extra Attribute, z.B. MANAGER und ANG, kann die Untermenge weggelassen werden. Beziehungen anderer *Entity-Mengen*, die MANAGER benötigen, werden durch Links nach ANG realisiert.

In ANG selbst kann, falls notwendig, die Tatsache, daß ein Angestellter Manager ist, durch ein 1-Bit Feld repräsentiert werden.[1]

1. Die männliche Schreibweise „Manager“, „Angestellter“ schließt immer die Tatsache mit ein, daß Manager und Angestellte sowohl Frauen als auch Männer sind.

Übung 8–2

Verdeutlichen Sie die obige Aussage durch ein Beispiel mit Angestellten, Abteilungen (die einen Manager haben) und den entsprechenden Links.

Zur Behandlung allgemeiner *n*:*m*-Beziehungen werden diese in mehrere 1:*n*-Beziehungen aufgebrochen: man führt eine „künstliche“ Entity-Klasse ein, im Extremfall ohne jegliche echte Felder und stellt dann die many-to-one-links zwischen den neuen Sätzen und den gegebenen her.

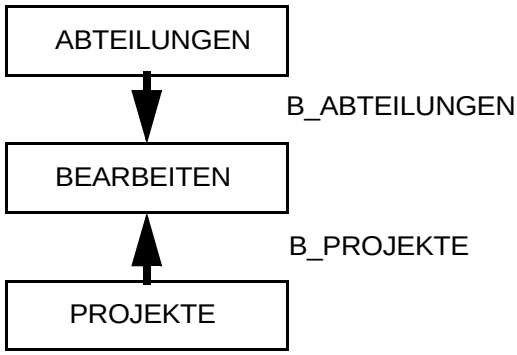


Abb. 8–2 *n:m-Beziehung aufgebrochen in zwei 1:n-Beziehungen*

Beispiel 8–1 Projekte und Abteilungen im Netzwerkmodell

Eine Abteilung kann mehrere Projekte bearbeiten, ein Projekt wird ggf. von mehreren Abteilungen bearbeitet.

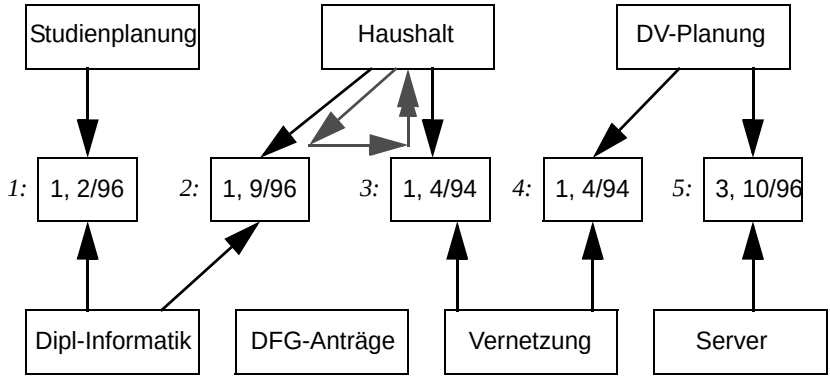


Abb. 8–3 *Abteilungen, Projekte und Verbindungssätze mit Links*

In Abbildung 8–3 haben die Bearbeitungssätze zwei Attribute:

- Anzahl der Mitarbeiter der Abteilung an dem Projekt und
- Datum der Übertragung der Projektaufgabe an die Abteilung

Man beachte, daß zwei Bearbeitungssätze identische Werte haben können. Sie sind durch ihre Adressen und Links verschieden.

Die entsprechenden Satzformate würden z. B. lauten:

```
ABTEILUNGEN(ANR, ANAME, LEITER)
PROJEKTE(PID, PNAME, BUDGET)
BEARBEITEN(ANZAHL_MITARBEITER, SEIT_WANN)
```

Übung 8–3

Wie würden entsprechende Relationen lauten?

Übung 8–4

Geben Sie ein Beispiel für die *n*:*m*-Beziehung „unterrichtet-in“ zwischen Schulklassen und Lehrern. Welches wären sinnvolle Attribute für die Verbindungssätze?

Das Netzwerkmodell führt den Begriff des *Besitzes*, bzw. *Besitzers* ein (engl. *ownership* und *owner*); im Beispiel oben etwa besitzt „Vernetzung“die Bearbeitungssätze 3 und 4, „Haushalt“ besitzt die Bearbeitungssätze 2 und 3.

Die folgende Abb. 8–4 zeigt das Beispiel 2–2 mit Kunden-Artikel-Lieferanten. Alle Beziehungen waren binär, Bestellung-Artikel aber *n*:*m*.

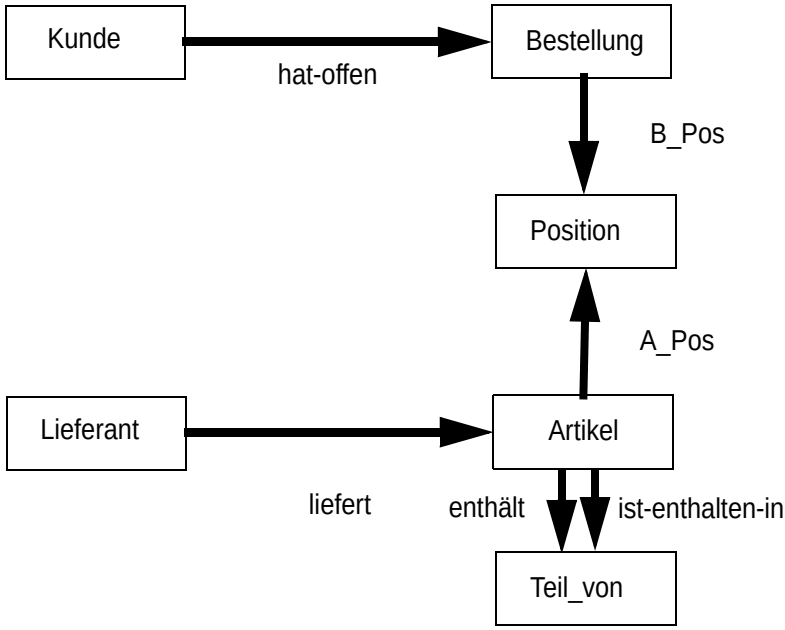


Abb. 8–4 *Netzwerk für Kunden-Artikel-Lieferanten*

Übung 8–5

Die logischen Satzformate für Kunden, Bestellungen und den neuen Satztyp POSITION sind

```
KUNDE(KDNR, KNAME, ORT)
BESTELLUNG(BESTNR, DATUM)
POSITION(POSNR, MENGE)
```

Wie lauten die Formate für Artikel, Lieferant und den neuen Satztyp TEIL_VON?

Das Netzwerkmodell ist auch als DBTG (Data Base Task Group) Modell bekannt, nach der DBTG CODASYL Sprache, einem Normvorschlag, mit Komponenten

- *Data Definition Language* (Datendefinitionssprache, DDL)
- *Subschema Data Definition Language* (Subschema DDL) zur Definition von Sichten
- *Data Manipulation Language* (Datenmanipulationssprache, DML) zum Schreiben von Anwendungen

Die Beziehungen heißen dort „Sets“ (also Mengen) - ein sehr unglücklicher Begriff - die sogar geordnet sein können [sic]. Er wird verständlich, wenn man ringförmige Realisierungsmöglichkeit betrachtet (vgl. Abb. 8–3- grauschattierte Pfeile): vom owner gibt es einen *Verweis* (link instance) auf den 1. Mitgliedssatz und von dort zum 2. Satz (Ordnung!), ..., zum letzten Mitgliedssatz der DBTG-Menge und von dort noch einen speziellen Verweis (->>) zurück zum owner.

Damit ist die *Navigation* vom Besitzer zu den Mitgliedssätzen und von einem Mitgliedssatz zu seinem Besitzer möglich. Der Ring dient allerdings oft nur als Gedankenmodell und die Verweise werden z.B. über Hash-Tabelle realisiert.

Die DDL-Vereinbarungen für die Satztypen ABTEILUNGEN, PROJEKTE und BEARBEITEN lauten (vereinfacht):

```
RECORD NAME IS ABTEILUNGEN;
  WITHIN BASIC-DATA-AREA;
  02 ANR; TYPE IS CHARACTER 6.
  02 ANAME; TYPE IS CHARACTER 30.
  02 LEITER; TYPE IS CHARACTER 30.

RECORD NAME IS PROJEKTE;
  WITHIN BASIC-DATA-AREA;
  02 PID; TYPE IS CHARACTER 6.
  02 PNAME; TYPE IS CHARACTER 30.
  02 BUDGET; TYPE IS FIXED DECIMAL 9.
```

```
RECORD NAME IS BEARBEITEN;
  WITHIN LINK-DATA-AREA;
  02 ANZAHL-MITARBEITER;TYPE IS FIXED DECIMAL 4.
  02 SEIT-WANN;TYPE IS CHARACTER 12.

SET NAME IS A-BEARBEITEN;
  OWNER IS ABTEILUNGEN;
  ORDER IS PERMANENT SORTED BY ...
  MEMBER IS BEARBEITEN

SET NAME IS P-BEARBEITEN;
  OWNER IS PROJEKTE;
  ORDER IS ...
  MEMBER IS BEARBEITEN.
```

Zusätzlich besteht die Möglichkeit, *virtuelle Felder* einzuführen, z.B. ANAME aus ABTEILUNGEN in den BEARBEITEN-Sätzen. Virtuelle Felder vermeiden *Redundanz*, erhöhen aber die Zugriffszeit.

```
RECORD NAME IS BEARBEITEN;
  WITHIN LINK-DATA-AREA;
  02 ANZAHL-MITARBEITER;TYPE IS FIXED DECIMAL 4.
  02 SEIT-WANN;TYPE IS CHARACTER 12.
  02 ABT-BEZEICHNUNG IS VIRTUAL
  SOURCE IS ANAME OF OWNER OF A-BEARBEITEN
```

Übung 8–6

Wie würde die Definition des virtuellen Felds für die Projektbezeichnung lauten?

Übung 8–7

Geben Sie einen Ausschnitt des Lieferanten/Artikel-Beispiels als DBTG-DDL Vereinbarung an!

Zur Datenmanipulation werden Navigationsbefehle

```
FIND
GET
STORE
```

eingebettet in eine *Gastsprache* (COBOL im DBTG-Vorschlag). Die Befehle verwenden einen Zeiger auf den zuletzt zugegriffenen Satz

(innerhalb des Programms, eines Record-Typs, einer DBTG-Menge), den sog. *currency pointer.*

Weitere (eher unglückliche) Begriffe sind

- *CALC-Key*
  Menge von Feldern für Zugriffsindex (nicht notwendigerweise ein eindeutiger Schlüssel)
- *Database Key*
  Zeiger, physische Adresse eines Satzes, Resultat der Dereferenzierung eines currency pointers
- *Scans*
  Durchlaufen von Ketten
- *Workspace*
  Arbeitsbereich für Sätze, Zeiger auf Sätze, Programmvariable

Bedeutendste Vertreter der Netzwerk-Datenbanken sind (bzw. waren) ADABAS (Software AG, Darmstadt) und UDS (Siemens), vgl. auch die Kurzbeschreibung in [Zehn]. Daneben sind häufig Mischformen „Netzwerk-relational“ anzutreffen (Links als Felder).

> **Problem des Netzwerk-Modells:** mangelnde Trennung von logischer und physischer Sicht – „Pointer-Philosophie“

Übung 8–8

Ein Satz ist Besitzer von zwei DBTG-Mengen und selbst Mitglied in drei DBTG-Mengen. Wieviele Link-Felder braucht man bei einer Realisierung mittels Ringstruktur?

8.2 Das hierarchische Datenmodell

Das *hierarchische Datenmodell* ähnelt dem Netzwerkmodell, erlaubt aber keine allgemeinen Graphen, sondern nur *Mengen von Bäumen* (Wälder), d.h. Mengen von gerichteten Graphen, bei denen jeder Knoten – außer der Wurzel – genau einen Vorgänger hat.

Die wichtigsten Merkmale des hierarchischen Datenmodells lassen sich wie folgt zusammenfassen:

- entstanden aus Dateisystemen mit Sätzen variabler Länge (sog. *Wiederholungsgruppen*) – vgl. Beispiel unten,
- starke Verwandtschaft zu physischer Speicherorganisation, z. B. *ISAM (*indexsequentielle Zugriffsmethode) und *VSAM* (*virtual storage access method*, IBM 1972),
- IBMs *IMS* (ab frühe sechziger Jahre) als bedeutendster Vertreter mit Datenmanipulationssprache *DL/I* (data language one),
- ungenügende Trennung des internen vom *konzeptionellen Schema*.

Beispiel 8–2 Abteilungen-Mitarbeiter-Projekte im hierarchischen Modell

Gegeben seien eine Datei mit Abteilungssätzen, je Abteilung mehrere Mitarbeiter und mehrere Geräte. Jeder Mitarbeiter nehme an mehreren Projekten teil.

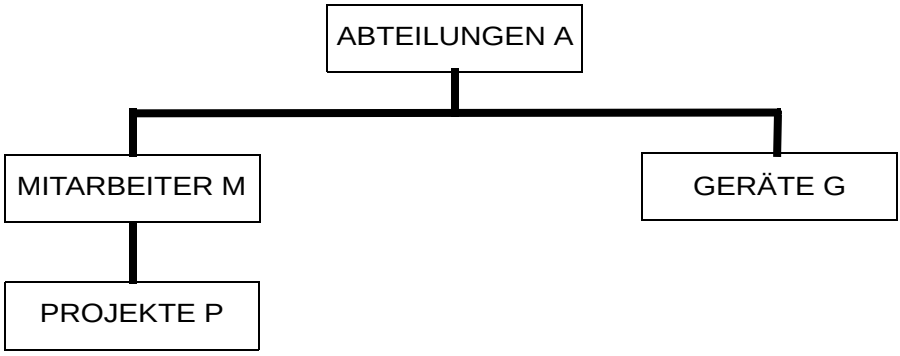


Abb. 8–5 *Hierarchie ABTEILUNGEN-MITARBEITER-GERÄTE*

Übliche Sprechweisen im Beispiel in Abbildung 8–5 sind: ABTEILUNGEN ist Vater von MITARBEITER und von GERÄTE; MITARBEITER ist z.B. Sohn von ABTEILUNGEN; MITARBEITER ist ABTEILUNGEN unmittelbar hierarchisch untergeordnet, PROJEKTE ist ABTEILUNGEN mittelbar untergeordnet. Ferner bezeichnet man den Entity-Typ ABTEILUNGEN als *Wurzel-Typ*.

Geht man zur Ebene der Instanzen über (vgl. Abbildung 8–6 unten), dann bezeichnet man z.B. die Mitarbeiter m_{11} und m_{12} der Abteilung a_1 als *Geschwister*. Auf dieser Ebene kann ein übergeordneter Knoten (Vater) auch keine Nachfolger (Söhne) haben, z.B. besitzt Mitarbeiter m_{22} die leere Projektmenge.

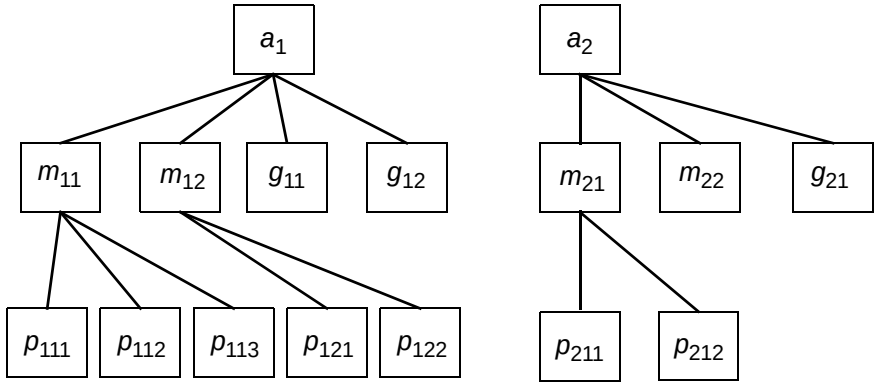


Abb. 8–6 *Instanzen ABTEILUNGEN-MITARBEITER-PROJEKTE*

Zur schnellen (Preorder-)Traversierung der Instanzen wird der Baum ggf. gefädelt (vgl. Abb. 8–7), d. h. es werden Querverweise und Rückverweise mit abgespeichert, bzw. es werden Multilist-Strukturen eingeführt, um Wiederholungsgruppen effizient an den Vaterknoten zu binden.

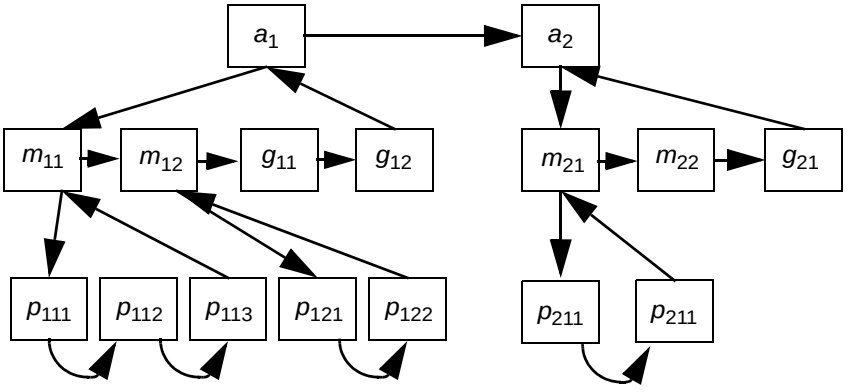


Abb. 8–7 *Verkettung der Instanzen*

Übung 8–9

Inwieweit ähnelt die Verkettung im Beispiel oben den *Netzwerk-Member-sets*?

Übung 8–10

Wieviele Felder für Zeiger werden an jedem Knoten in Abbildung 8–6 benötigt? Welche andere Verkettung(en) könnte man sich vorstellen? Hinweis: wie lautet die Knotenreihenfolge in Preorder?

Die Suche mittels *Schlüssel* innerhalb des *Wurzeltyps* wird in hierarchischen DB-Systemen immer unterstützt, z. B. durch Hashing oder einen baumartigen Index. Nicht notwendigerweise unterstützt das Modell den Zugriff vom Sohn auf den Vater. Modelliert man aber *n*:*m*-Beziehungen, z.B. wenn Mitarbeiter verschiedener Abteilungen an den selben Projekten arbeiten, müssen entweder

- Projektsätze mehrfach aufgeführt (*Redundanz*) oder
- virtuelle Sätze eingeführt werden.

Im Beispiel ist die *n*:*m*-Beziehung Projekte-Abteilungen gelöst durch zwei Bäume mit virtuellem Satztyp (vgl. Abb. 8–9). Häufig deutet man den virtuellen Satztyp mit *T an, wenn sein Zieltyp T ist, z. B. *ARTIKEL als virtueller Satztyp, der zur Darstellung der *n*:*m*-Beziehung BESTELLUNG-ARTIKEL benötigt wird (Abb. 8–8 unten).

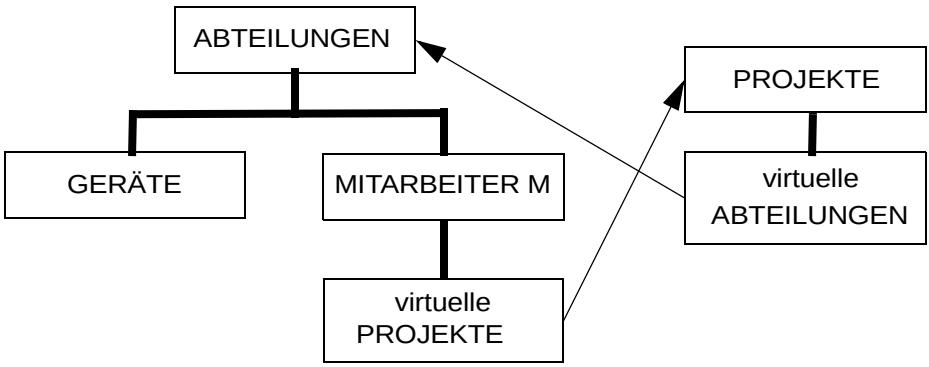


Abb. 8–8 *ABTEILUNGEN-PROJEKTE und virtuelle Sätze*

Häufig treten auch Kombinationen echter Datenfelder mit virtuellen Feldern auf, d.h. mit Feldern, die Links (Verweise) auf den gewünschten Satz eines anderen Typs enthalten.

> **Problem des hierarchischen Modells**: keine geeignete Realisierung von *n*:*m*-Beziehungen.

Wie können in Abb. 8–9 Kunden benachrichtigt werden, falls ein Lieferant für einen bestellten Artikel ausfällt?

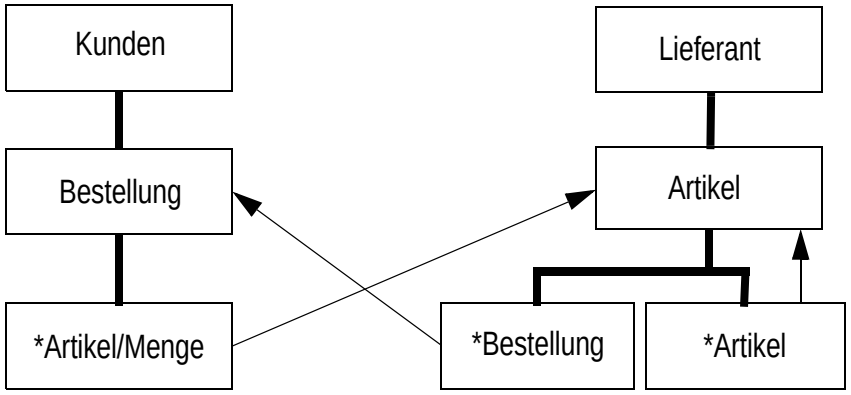


Abb. 8–9 *n:m-Beziehung im hierarchischen Modell*

Übung 8–11

Geben Sie geeignete Entity-Typen und Beziehungen für das Lehrer-Schüler-Klassen-Beispiel im hierarchischen Modell an.

8.3 Das objekt-orientierte Datenmodell

Bei den obigen Modellen, bzw. deren kommerziellen Realisierungen, fallen die folgenden Schwachstellen auf:

- Es sind keine *aktiven Elemente* vorgesehen, d. h. Aktionen, etwa in Form von Prozeduren, die ggf. automatisch ausgelöst werden bei einem Zugriff oder einer Modifikation. Ein trivialer Fall (heute meist möglich): für ein Feld „Summe“ die Summe der Bestellung errechnen aus Summe der Posten; schon schwieriger wäre es, die

Fläche für ein beliebiges Polygon zu errechnen im Zusammenhang mit Geo-Daten.

- Die Abspeicherung *nicht-textuelle Werte* (Bilder, Graphiken) ist häufig unmöglich.
- Obwohl im Netzwerkmodell z. T. möglich, erlaubt das relationale Modell keine Wiederholungsgruppen als Datenwert, allgemein keine sog. *komplexen Objekte* zur Modellierung zusammengesetzter Strukturen, z. B. einer CAD-Zeichnung bestehend aus Ecken, Kanten, Flächen.
- Problematisch sind auch alle Anwendungen, die Rekursion und transitiven Abschluss erordern, z. B. die Menge aller Städte, die, ausgehend von einer gegebenen Stadt, per Zug bei Abfahrt nach 20 Uhr noch am selben Tag erreicht werden.
- Schon die Verwirklichung *varianter Strukturen*, z. B. ein Kunde mit 3 Anschriften, ist oft unmöglich.
- Neben *existentiellen Abhängigkeiten* („keine Bestellung ohne zugehörigen Kunden“) lassen sich in der Miniwelt meist noch strengere Vorschriften aufstellen, etwa der Art „wenn Storno, dann immer für Gesamtmenge des ursprünglichen Bestellpostens“, „kein Artikel kann direkt oder indirekt in sich selbst enthalten sein“, „Lieferanten, deren Artikel für 12 Monate nicht nachgefragt wurden, sind zu streichen“, usw. Diese Integritätsbedingungen müssen zur Zeit fast immer von Hand einprogrammiert werden, d.h. es mangelt an *automatisierten Integritätsüberprüfungen*.

Anfang der achtziger Jahre begann man Datenmodelle zu entwerfen, in denen von der Forderung nach atomaren Datenwerten im relationalen Datenmodell (sog. 1. *Normalform*, vgl. Abschnitt 5.2) abgewichen wurde. Die Vorstellung war, daß sich mittels dieser geschachtelte Relationen im sog. NF^2-Datenmodell (***N**on-**F**irst **N**ormal **F**orm-Datenmodel*) die komplexen Objekte der sog. Nicht-Standard-Anwendungen in Wissenschaft und Technik leichter und effizienter abbilden ließen.

Prototypen für dieses Datenmodell sind z. B. DASDB [Scheck], AIM-P [Dad86] und ESCHER [KTW90]

{ } KUNDEN						
[ ] KUNDE			{ } BESTELLUNGEN			
#KDNR	#KNAME	ORT	#BESTNR	DATUM	{ } POSTEN	
					#ARTNR	#MNG
4710	Schmidt	Hamburg	20996	96-11-12	2100	1
			20998	96-11-16	{ }	
4712	Müller	Mittelstadt	20995	96-11-12	1002 1102	4 1
			20997	96-11-13	1001 1202 1102	6 1 1
4711	Meier	Kleinhausen	{ }			

Abb. 8–10 *Bestellung als geschachtelte Relation*

Abbildung 8–10 zeigt eine solche geschachtelte Relation in der ESCHER-Darstellung mit den Werten aus einer klassischen betriebswirtschaftlichen Anwendung. Man beachte, daß der Kunde „Schmidt“ eine Bestellung mit einer leeren Menge von Posten, der Kunde „Meier“ eine leere Menge von Bestellungen hat.

Gegenwärtig ist die Diskussion über die Vor- und Nachteile dieses Datenmodells noch nicht abgeschlossen. Fraglich ist

- ob die *Schachtelung nur als Sicht* existieren soll und generell auf das flache relationale Modell intern abgebildet wird (XNF-Sprachvorschlag [Lindsay]),
- wie die *Rekursion* befriedigend gelöst werden kann,
- wie *n*:*m*-Beziehungen optimal darstellbar sind,
- ob die Komplexität der Abfragesprache den Anwender nicht überfordert.

Von vielleicht größerer kommerzieller Bedeutung sind deshalb die sog. *objekt-orientierten Datenbanken*, die neben strukturierten Werten zur Modellierung komplexer Objekte noch weitergehende Konzepte, wie *Vererbung*, *Kapselung*, *Botschaftenaustausch*, usw. vorsehen.

Objekt-orientierte Datenbanksysteme (OODBS) sind entstanden in der Folge der stärkeren Verbreitung objekt-orientierter Programmiersprachen - z.B. Smalltalk, Eiffel, C++ - und bauen größtenteils auf den persistenten Varianten dieser Sprachen auf.

Entsprechend fordern die Vertreter dieser Richtung, z.B. im OODBS Manifest [Atk89], daß sowohl die Elemente der Datenbanken, also Persistenz, Externspeicherorganisation, Mehrbenutzerbetrieb, Sicherheit vor Systemabstürzen und ad-hoc-Abfragen, möglich sein müssen, als auch die der OO-Programmiersprachen, also *komplexe Objekte*, *Objektidentität*, *Kapselung*, *Typen* und/oder *Klassen*, *Vererbung*, *Polymorphismus* mit später Bindung, *Erweiterbarkeit* und *operationale Abgeschlossenheit*. Die einzelnen Punkte sollen im folgenden erläutert werden.

Komplexe Objekte

Wie im NF^2-Datenmodell sollen Mengen, Tupel, *Listen* (arrays), ggf. auch *Multimengen* (bags) und deren orthogonale Kombination unterstützt werden.

Objektidentität

Es wird unterschieden zwischen identischen und (wert-)gleichen Objekten. Damit können sich zwei Objekte ein drittes teilen (object sharing), z.B. wenn man festhalten will, welche Institutsmitglieder sich welche Rechner teilen (vgl. auch die unterschiedlichen Zuweisungen $x := y$ (copy) und $x \leftarrow y$ (own)).

Kapselung

Diese Idee geht auf die *abstrakten Datentypen* (ADTs) der siebziger Jahre zurück. Datenstrukturen und die Realisierung von Operationen werden

versteckt im *Implementierungsteil*, nach außen werden nur wohldefinierte, abstrakte Zugriffs- und Manipulationsoperationen angeboten.

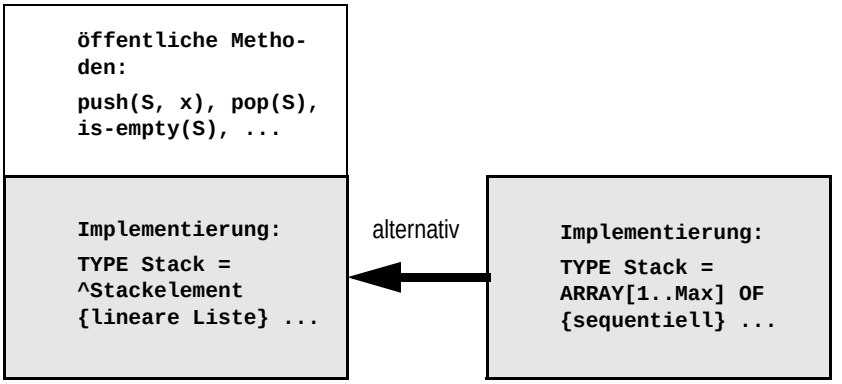


Abb. 8–11 *ADT-Darstellung eines Stapels*

Das klassische Beispiel dazu ist der Stapel (vgl. Abb. 8–11 mit privatem Implementierungsteil grau unterlegt).

Im Fall der OODBS erscheint die völlige Abschirmung der Werte unnatürlich: ein Datenbanksystem dient schließlich der Abspeicherung und Manipulation von Daten durch den Anwender, gerade auch durch nichtvorhergesehene, spontane Abfragen. Entsprechend ist die Ersetzung der Attribute durch Funktionen gleichen Namens, welche die gewünschten Werte liefern, bzw. setzen, eine eher gekünstelte Angelegenheit.

Typen und Klassen

Starkgetypte Programmiersprachen (Pascal und Nachfolger) bieten Schutz vor Programmierfehlern, weil bereits zur Übersetzungszeit die Typverträglichkeit der Operanden in einem Programm überprüft werden kann. In OO-Sprachen wird Typ häufiger im Sinne eines ADTs (vgl. oben) gebraucht, d. h. mit einem Satz öffentlich gemachter Methoden und einer versteckten Implementierung. So würde ein Typ BESTELLUNG z. B. die *Methoden* (Funktionen)

- `Lesen-Datum`, `Setzen-Datum`,
- `Lesen-Besteller`, `Setzen-Besteller`,

- `Lesen-Bestellsumme`, `Streichen-Bestellposten`,

usw. kennen, ein Typ KUNDE vielleicht die Methode `Anzahl-offener-Bestellungen`.

Unter einer Klasse versteht man dann meist die *Extension eines Typs*, d.h. die *Kollektion* (Menge, Liste, usw.) von Exemplaren (*Instanzen*) eines Typs. So könnte man sich Klassen KUNDE und KUNDE-MIT-OFFENER-BESTELLUNG vorstellen, wobei hier letztere Klasse eine Teilmenge der ersteren wäre. Klassen in einem OODBS dienen also als Sammelstätten für Objekte. Sie erlauben die Schaffung neuer Objekte (durch Kopieren oder wie im Fall der beiden KUNDEN-Klassen durch „sharing“ der gemeinsamen Teile eines Kundenexemplars) und bieten generische Funktionen für die Kollektion, z. B. die Suche nach Objekten mit bestimmten Merkmalen, an.

Für andere Arten von Klassen macht die automatische Verwaltung aller Objekte eines Typs, etwa die Menge aller Rechtecke, durch ein OODBS allerdings wenig Sinn.

Klassen- oder Typhierarchie (Vererbung)

Wie oben bereits angedeutet, lassen sich zu einer Klasse durch *Spezialisierung*, *Klassifizierung* und *Teilmengenbildung Unterklassen* bilden, die Merkmale und Methoden ihrer *Oberklasse* erben. Im Fall der „*single inheritance*“, d. h. wenn jede Klasse außer der obersten nur genau eine Oberklasse hat, werden die Hierarchien durch Bäume dargestellt.

Beispiel 8–3

In Abbildung 8–12 erbt `Klassenlehrer` alle Eigenschaften von `Lehrer` (z. B. Unterrichtsfächer, Jahr des Dienstantritts, etc.) und indirekt von `Person` (z. B. Name, Anschrift, Alter).

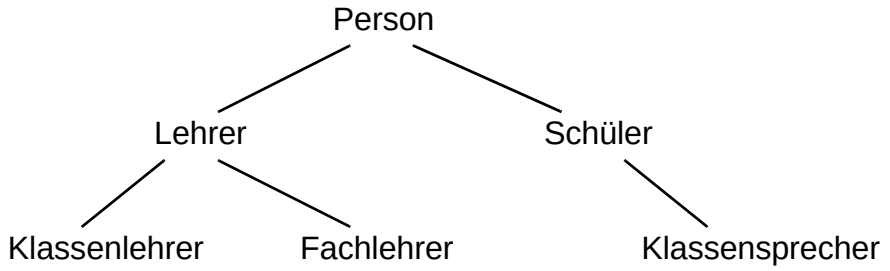


Abb. 8–12 *Eine Vererbungshierarchie*

Zusätzlich gibt es die für `Klassenlehrer` einmaligen Merkmale, z. B. Klasse, für die man Klassenlehrer ist.

Polymorphie

Um gewisse Operationen auf unterschiedlichen Objektklassen auszuführen, braucht man den Objektklassen(-typen) angepaßte Operationen.

Man denke etwa an eine Operation `berechne-Fläche`, die als generische Operation für geometrische Objekte vorgesehen ist und die für alle Unterklassen, wie z. B. Rechteck, Quadrat, Kreis, usw., erstellt werden muß.

Die Idee ist nun, daß der Anwender in sein Programm Aufrufe der Operation `berechne-Fläche` hineinschreiben darf, die dann zur Laufzeit (sog. *spätes Binden*!) durch die korrekten Aufrufe der speziellen passenden Operation ersetzt werden. Wird eine existierende Operation durch eine speziellere einer Unterklasse ersetzt, spricht man auch vom *Überladen des Operators*.

Sonstige Forderungen

Von SQL weiß man, daß es nicht *allgemeine Berechenbarkeit* bietet, d. h. daß es nicht die gleiche Mächtigkeit wie eine Programmiersprache (Typ 0 Mächtigkeit) hat. Für die DML eines OODBS wird dies gewünscht, zusammen mit der Erweiterbarkeit, d.h. neue Typen und Methoden sollen

vom Anwender eingebracht werden können und zwar ununterscheidbar von vordefinierten Typen. Die anderen Forderungen nach Persistenz, Externspeichermanagement, Mehrprogrammbetrieb (Nebenläufigkeit), Sicherung und Wiederaufsetzen im Fehlerfall, sowie Spontanabfragen sind selbstverständlich aus Sicht der klassischen, kommerziellen Datenbanksysteme.

Optionale Eigenschaften

Zu den wünschenswerten bzw. unklaren Eigenschaften zählen

- *Mehrfachvererbung*, engl. *multiple inheritance*, z. B. ein Lehrer, der an einer Weiterbildungsmaßnahme teilnimmt erbt Attribute und Operationen von Lehrer und Schüler;
- *Typprüfung*: möglichst vollständig zur Übersetzungszeit;
- *Verteilung*: Daten und Transaktionen verteilt auf mehrere Systeme;
- *lange und geschachtelte Transaktionen*, z. B. im Ingenieurbereich bei CAD-Entwürfen;
- *Versionsverwaltung*: wie lautete der Auftragsbestand des Kunden *x* per (engl. *as of*) 1.1.1993.

Kritiker dieser Manifests werfen den „OODBS-Heilsbringern“ im wesentlichen vor, daß sie mit ihren Verknüpfungen via OIDs eine navigierende DML im Stile der Netzwerke fördern. Sie argumentieren ferner [Comm90], daß viele der genannten Eigenschaften wie Typsystem, Vererbung, komplexe Objekte, usw. Eingang in Erweiterungen relationaler Systeme finden werden, ohne die Vorteile dieses Modells aufzugeben.

Gegenwärtig existieren bereits eine ganze Reihe experimenteller und auch bereits kommerzieller Vertreter der OODBS-Richtung:

- Altair O_2, Ontologic Ontos, HP Iris, IBM Starburst, Exodus, Cactis, Berkeley POSTGRES, Object Design ObjectStore, Versant OBJECT-Base, Itaska (ex MCC ORION), Servio Corporation GemStone, DEC Trellis/OWL, Symbolics Statice, Objectivity Objectivity/DB, TI Zeitgeist, POET.

Beispiel 8–4 Personen-Lehrer-Klassenlehrer

In einem sprachlich an O_2 angenäherten Modell sei eine Klasse Personen wie folgt eingeführt.

```
class Personen
  type tuple(
         PNr: integer,
         Name: tuple(
                Vorname: string,
                Nachname: string),
         Adresse: tuple(PLZ: integer,
                    Ort: string,
                    StrasseundNr: string),
         Geburtsdatum: date)
```

Die Klassen `Lehrer` und `Schüler` erweitern wir um die Attribute `Einstelldatum` und betreute `Schulklassen`, bzw. `Schulklasse` zu der ein Schüler gehört. Man beachte die **`inherits`**-Klausel für die isa-Beziehung. Ferner verwendet Schüler das objektwertige Attribut Klassenlehrer.

Für Lehrer wird zusätzlich die Methode `UStunden` vereinbart, deren Rückgabewert die Anzahl der geleisteten Unterrichtswochenstunden ist. Hier fällt auf, daß in der Definition nur die Methodenschnittstelle angegeben wird, der Rumpf folgt weiter unten.

```
class Lehrer inherits Personen
  type tuple(Einstelldatum: date,
         Schulklassen: set(tuple(SKlasse: string,
              AnzahlStunden: integer)))
  method UStunden: integer

class Schüler inherits Personen
type tuple(Klasse: string,
     Klassenlehrer: Lehrer)

method body UStunden: integer in class Lehrer
  {int S;
  S = 0;
  for (t in self → Schulklassen)
    S = S + t.AnzahlStunden;
  return(S)
  }
```

In der Methode `UStunden` bezeichnet `self` das die Methode aufrufende Objekt. Würde man auch für Schüler ein mengenwertiges Attribut `Unterricht: set(tuple(UFach: string, AnzahlStunden: integer))` vereinbaren, könnte auch für Schüler eine Methode `UStunden` definiert werden, hier die Anzahl der zu besuchenden Unterrichtsstunden. Allgemein auf Personen angewandt, müßte zur Laufzeit die richtige Methode automatisch ausgewählt werden, je nachdem ob man einen Schüler oder einen Lehrer betrachtet.

Übung 8–12 UStunden für Schüler

Wie sähe die Methode `UStunden` für eine derartig erweiterte Definition von Schüler aus?

Der ODMG-93 Standard

Die Standardisierungsgruppe ODMG (Object Data Management Group) ist Teil der OMG ist; letztere ist bekannt durch den CORBA (Common Request Broker Architecture) Standard. Die unabhängige ODMG unter Leitung ihres Vorsitzenden Rick Cattell bemüht sich um die Einführung eines Standards und hat dazu den Vorschlag ODMG-93 unterbreitet [Catt94]. Der Vorschlag enthält vier Komponenten (nach [HS95]):

- ein an C++ orientiertes *Objektmodell*
- die Datenbanksprachen ODL (Object Definition Language) und OQL (Object Query Language)
- *Spracheinbettungen* (engl. *bindings*) für C++ und SMALLTALK
- einen Bezug zur OMG, zu CORBA und ANSI C++.

Das Objektmodell kennt nur Objekttypen und verzichtet auf Klassen. Für Objekttypen können Attribute, Operationen und Beziehungen definiert werden.

Für Details zu OODBMS verweisen wir auf die Literatur. Gute Bücher über objekt-orientierten Datenbanken sind Heuer [Heuer92] und – etwas kürzer gehalten – Lausen/Vossen [LV96]. Übersichten finden sich in

Heuer/Saake [HS95] und z. B. im Themenheft Next-Generation Database Systems, Comm. ACM, Vol. 34, No. 10 (Oktober 1991).

Übung 8–13

Entwerfen Sie eine Typhierarchie für Beschaffungen einer Hochschule (Bücher, Geräte, Rechner, etc.).

Übung 8–14

Geben Sie für ein geometrische Objekt, z. B. einen Polyeder, eine NF^2-Darstellung an.

Übung 8–15

Wie sähe eine objekt-orientierte Darstellung aus? Welche Methoden würde man sich wünschen? Hinweis: Man denke etwa an eine Methode (Funktion) `Grad(v)`, die zu einem Knoten `v` die Anzahl der verbundenen Kanten liefert.

Anhang Lösungen

Lösungen zu den Übungsaufgaben

Übung 1–1

Bei Programmen wird über den Compiler/Linker eine Verbindung von einer Variablen zum Speicherort des zugehörigen Werts (Hauptspeicheradresse) hergestellt. Man spricht von *Ortsadressierung*, die Adresse ist bekannt, der dort gespeicherte Wert ggf. nicht. Bei externer Speicherung ist oft ein Wert (der Schlüssel, also ein Teil des Inhalts des Satzes) bekannt, die Adresse muss aber gesucht werden, um an den weiteren Inhalt zu gelangen. Daher spricht man von *Inhaltsadressierung*.

Übung 2–1

Bei Kunden natürlich z. B. Bonität, letzte Bestellung, Privatkunde, abweichende Lieferanschrift, Kunde seit wann. Generell kann man an dieser Stelle auch diskutieren, welche Angaben man aus unterschiedlichsten Gründen wohl eher nicht in einer Datenbank speichern würde, z. B. persönliche, nicht objektivierbare Ansichten, schlecht formalisierbare Fakten.

Übung 2–2

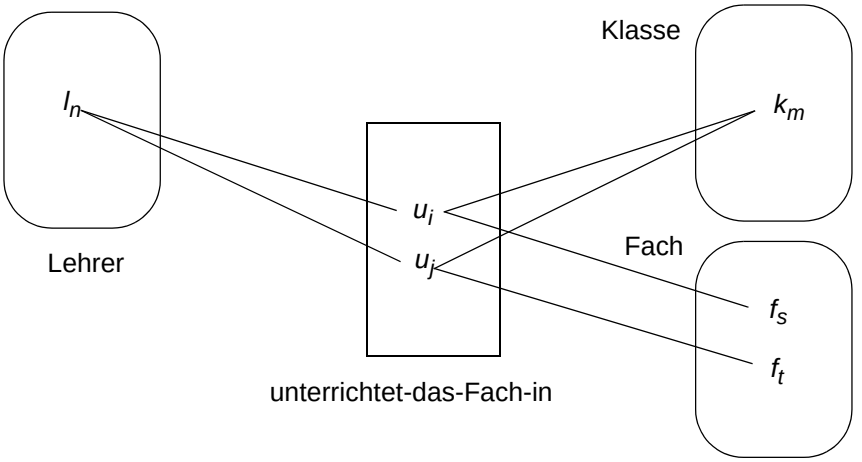
Invariante Objektidentifier (vgl. Kapitel 8 - Objektidentität).

Übung 2–3

Vgl. Kapitel 8. Entity-Klassen ähneln den Objektklassen, Extension den Objektinstanzen (Entities), Vererbung ist als is-a-Beziehung modellierbar.

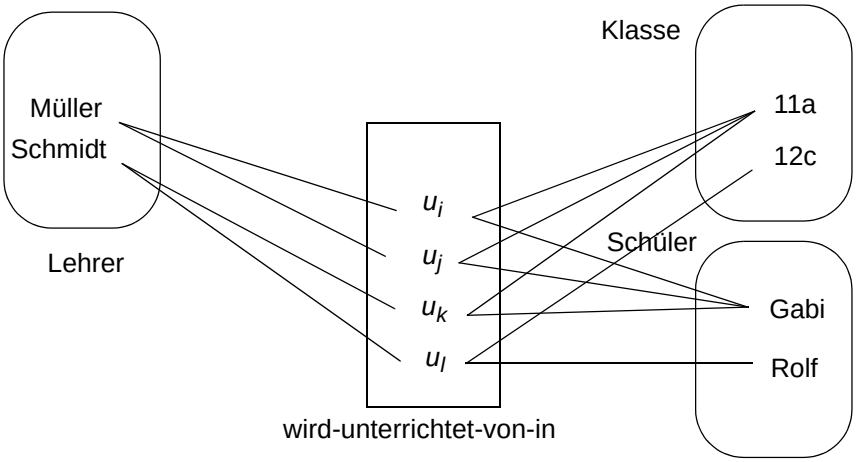
Übung 2–4

Die Abbildung 2-2 drückt nicht aus, wie viele oder welche Fächer ein Lehrer in einer Klasse unterrichtet. Die Beziehung ließe sich aber erweitern zu „unterrichtet-das-Fach-in“ und das Fach als Attribut dort unterbringen, oder die Beziehung ternär machen, wie unten angedeutet.



Übung 2–5

Lehrer Müller unterrichtet Schülerin Gabi aus der Klasse 11a z. B. in Che-



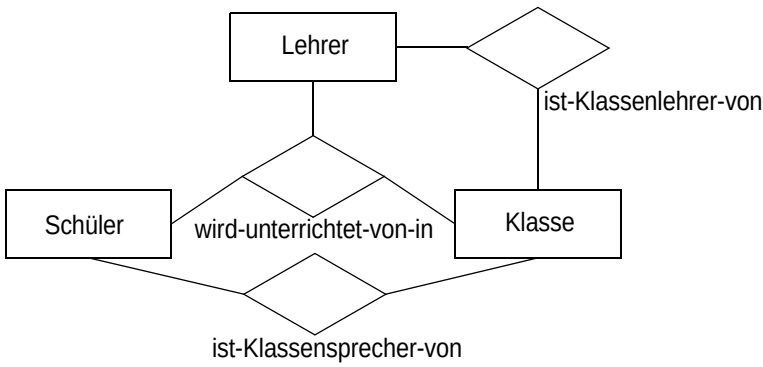
mie (u_i) und in Physik (u_j), Gabi hat zudem noch Frau Schmidt in Mathematik (u_k). Rolf aus der 12c hat auch Frau Schmidt, z.B. ebenfalls in Mathematik (u_l). Generell wird das Fach nicht explizit durch die Beziehung festgelegt, sondern hier nur beispielhaft erwähnt.

Übung 2–6

Bei der ternären Beziehung ist klargestellt, dass bei Beteiligung von (Gabi, Müller, 11a) an dieser Beziehung wie oben gezeigt, Lehrer Müller Schülerin Gabi tatsächlich in einem Fach in der 11a unterrichtet. Die drei Ausprägungen (Müller, Gabi), (Müller, 11a), (Gabi, 11a) stellen das nicht unbedingt sicher, vielleicht weil Gabi in der 11a kein Fach belegt, das Müller dort unterrichtet und die Verknüpfung (Müller, Gabi) vielleicht aus dem Unterricht in der Theater-AG stammt.

Ein ggf. besseres Beispiel wären Buchempfehlungen eines Dozenten für seine Vorlesung. Nehmen wir an, es gibt Dozenten *A* und *B*, Datenbankbücher *X* und *Y* und Vorlesungen *DB I* und *DB II*. Bei (A, DB I, X), (A, DB II, Y), (B, DB I, Y), (B, DB II, X) ist klar, wer welches Buch für welche Datenbankvorlesung empfiehlt (die Empfehlungen der Dozenten *A* und *B* sind gerade gegenseitig vertauscht). Aus (A, DB I), (A, DB II), (B, DB I), (B, DB II), (A, X), (A, Y), (B, X), (B, Y) und (DB I, X), (DB I, Y), (DB II, X), (DB II, Y) geht diese Präferenz nicht mehr hervor, alle Kombinationen sind möglich, es tritt ein Informationsverlust ein.

Übung 2–7

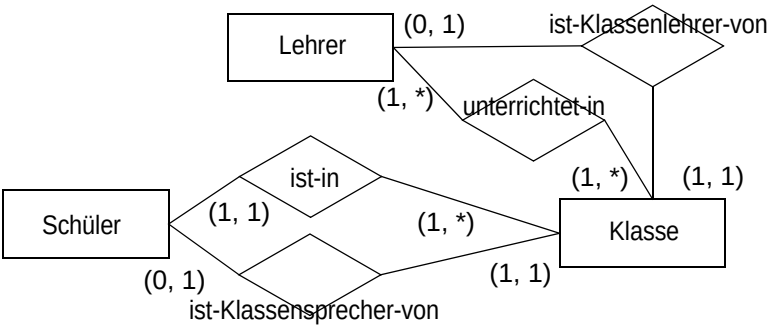


Übung 2–8

Die Linien von Projekt zu „arbeitet-an“ und zu „leitet“ könnte man doppelt ziehen, um die existentielle Abhängigkeit anzudeuten. Ein Projekt stirbt, wenn es keine Angestellten gibt, die daran arbeiten oder es keiner leiten will.

Übung 2–9

Die Annahmen hier lauten: Lehrer unterrichten in mindestens einer Klasse, ggf. in vielen. Jeder Schüler sei in genau einer Klasse, jede Klasse habe mindestens einen, in der Regel viele Schüler. Nicht jeder Lehrer ist Klassenlehrer, wenn dann nur in einer Klasse. Genauso ist nicht jeder Schüler Klassensprecher und natürlich nur für eine Klasse. Jede Klasse hat genau einen Klassenlehrer und einen Klassensprecher.



Übung 2–10

Artikel hängen existentiell vom Lieferanten, Bestellungen von Artikel und Kunde ab. Damit wird ausgedrückt, dass ein Artikel, für den der Lieferant nicht bekannt ist, nicht im Sortiment existieren sollte. Genauso sollen leere Bestellungen oder solche mit unbekanntem Auftraggeber verhindert werden.

Übung 2–11

Die klassische hierarchische Beziehung im Beispiel Lehrer-Schüler-Klasse ist zwischen Klasse und Schüler, weil eine Klasse viele Schüler hat, jeder Schüler aber genau einer Klasse angehört. Lehrer unterrichten in vielen Klassen und Klassen haben viele Lehrer (Lehrer-Schüler analog), daher sind dies *n*:*m*-Beziehungen und damit nicht hierarchisch.

Klassenlehrer, Oberstufenlehrer, Fachgebietsleiter wären Spezialisierungen, Schüler ggf. eine Generalisierung von Unter-/Oberstufenschüler.

Übung 2–12

Das ER-Diagramm findet man in Abb. 2-1 mit MITARBEITER statt ANGESTELLTE. Geeignete Tabellen lassen sich analog zu denen in Beispiel 2-3 konstruieren. In der Projekttabelle tauchen genau die Abteilungen nicht auf, die keine Projekte bearbeiten, etwa eine Personalabteilung. Wegen der geforderten hierarchischen Beziehung zwischen ABTEILUNG und PROJEKT müsste die Kardinalität bei PROJEKT (1, 1) sein.

Übung 2–13

AbtNr	AbtName	{Angestellte}
		PersNr
1	FuE	101
		103
2	Marketing	102
		201
		202

Der Vorteil dieser Speicherform ist, dass man mit dem Zugriff auf eine Abteilung auch sofort alle Angestellten (zumindestens deren Personalnummer) kennt. Ein Test, wieviele Angestellte eine Abteilung hat, wäre sehr leicht und schnell durchzuführen. Speichert man wie in Beispiel 2-3 weiterhin bei den Angestellten die Abteilungsnummer ab, dann kann es zu Inkonsistenzen kommen. Hingen an den Abteilungstabellen auch noch die mengenwertigen Attribute für bearbeitete Projekte, vorhandene Ausstattung, belegte Standorte, ... würde die Abteilungstabelle sehr groß und unübersichtlich.

Übung 3–1

Eine neue Spalte kann zunächst mit Nullwerten gefüllt werden. Ein Tupel ebenfalls, außer bei dem/den Attribut/en für den Schlüssel.

Übung 3–2

Sicher kann man einen Garantiefall nur anmelden, wenn das Alter stimmt und die Garantiekarte (Rechnung) vorhanden ist. Ist dagegen beides nicht gegeben, kann man die Garantieleistung definitiv nicht geltend machen. Im Fall, dass eines von beiden sicher wahr ist und der zweite Faktor – oder auch beide Faktoren – noch unbekannt ist bzw. sind, kann man vielleicht noch einen Garantiefall anmelden.

Die Eignung für einen Job steht definitiv fest wenn ein Diplomzeugnis vorgelegt oder der Nachweis von mindestens 3 Jahren Berufserfahrung geführt wurde. Nur wenn beides ausgeschlossen ist, ist die Eignung sicher nicht gegeben. In allen anderen Fällen, bei denen mindestens ein Sachverhalt unklar ist, ist vielleicht die Eignung doch gegeben.

Übung 3–3

Geschäftsführer/-innen und Lehrlinge hätten ggf. ein ?I, weil sie keinen Abteilungswert bekommen. Normale Angestellte hätten bei fehlendem Abteilungswert ein ?A, weil dort eigentlich ein Wert eingetragen sein sollte, der aber momentan unbekannt ist. Wäre der Status (Geschäftsführer, Lehrling, Angestellter) unbekannt (?A), dann stünde bei Abteilung ?U (unbekannt ob anwendbar).

Übung 3–4

Die Null und die leere Zeichenkette sind als interne oder externe Repräsentanten einer DB-Null ungeeignet, weil sie mit echten Werten verwechselt werden könnten.

Übung 3–5

Von Universitätsbibliotheken kennt man die Praxis, dass bei einer Dauerleihe für das Rückgabedatum, also *A*, der Wert x = Ausleihdatum + 100 Jahre ausgegeben wird.

Übung 3–6

Eine mögliche externe Repräsentation für DB-Null bei Zeichenketten ist „?“ oder „n.a.“, was im Deutschen *nicht anwendbar* und im Englischen

not applicable bedeutet. Einen eindeutigen internenen Repräsentanten zu finden und festzulegen, dürfte schwierig sein, da z.B. auch ASCII-Null als Terminator von Zeichenketten auftritt und damit die Unterscheidung zur leeren Zeichenkette nicht gegeben ist. Daher speichern viele Datenbanksysteme bei allen Attributwerten ein ganzes Extrabyte mit ab, mit dem signalisiert wird, ob ein DB-Nullwert vorliegt oder nicht.

Übung 3–7

Der SQL-Standard legt nicht fest, wohin Nullwerte in der Sortierordnung gehören. In Oracle kommt die DB-Null am Ende bei aufsteigender Ordnung, möglich ist eine Klausel NULLS FIRST bzw. NULLS LAST.

Ein Nulltupel hätte für alle Attrbutwerte DB-Null eingetragen. Davon könnte es höchstens eines geben wegen der Mengeneigenschaft von Relationen. Meist verbindet man aber mit dem Schlüsselattribut eine NOT NULL-Klausel, die das Auftreten von Nullwerten in diesem Attribut verbietet.

Übung 3–8

Man betrachte eine geschachtelte Personalrelation mit einem mengenwertigen Attribut KINDER. Die leere Menge bedeutet, dass die Person keine Kinder hat. Eine Menge mit einem Nulltupel bedeutet, es gibt ein oder mehrere Kinder über die aber keine Daten, etwa weder Name, Alter noch Geschlecht, bekannt sind. Alternativ, wenn bekannt wäre, dass die Person, sagen wir, drei Kinder hätte, über die aber nichts weiter bekannt wäre, könnte man drei Tupel mit Nullwerten und den fiktiven Kindernamen A, B, C einfügen. Eine mengenwertige DB-Null würde man ablegen, wenn nicht bekannt wäre, ob die Person Kinder hat. Gäbe es ein explizites Attribut ANZAHL_KINDER, dann wäre im ersten Fall dort 0 einzutragen, im zweiten eine Zahl n > 0, die die bekannte Zahl an Kindern wiedergibt, im letzten Fall DB-Null, wenn über deren mögliche Existenz nichts bekannt ist.

Übung 3–9

Will man die Mengeneigenschaft (kein Element doppelt) einhalten, wäre {{A, 4711}, {A, 4712}, {B, 4711}, {A, ?}, {?, 4712}, {?, ?}} eine sinn-

volle Ausgabe, solange für das zweite Attribut auch andere Werte als 4711 und 4712 zulässig wären. Wenn dem nicht so ist, wäre {A, ?} zwangsläufig ein Duplikat. Ähnliche Überlegungen kann man anstellen, wenn das erste Attribut nur A oder B sein kann. Multimengen lösen solche Probleme nicht wirklich.

Übung 3–10

Der Entwurf der Schemata hängt vom Einsatzzweck ab. Lehrer werden in der Personaltabelle des Kultusministeriums sicherlich per Schlüssel PERSONALNR unterschieden. Würde die Stundenplanung innerhalb einer Schule datenbankgestützt ablaufen, wäre als Schlüssel für eine Lehrertabelle sicherlich der Nachname oder das Paar (Nachname, Vorname) geeignet, bei Schülern u.U. auch das Triple (Nachname, Vorname, Geburtsjahr). Geeignete Relationenschemata lassen sich daraus entwickeln.

Übung 3–11

Gemeint ist das ER-Diagramm aus Abb. 2-10 und die angedeutete Fremdschlüsselbeziehung in Beispiel 3-4.

- Die Einfügung einer Bestellung mit unbekannter Kundennummer muss zurückgewiesen werden, was sich in SQL erzwingen lässt.
- Genauso eine Änderung der Kundennummer zu einer unbekannten Nummer für eine schon bestehende Bestellung.
- Die Änderung der Kundennummer in der Kundenrelation (Änderung des Primärschlüsselwerts) lässt die Bestellungen mit der alten Kundennummer in der Luft hängen (dangling tuples), ggf. werden diese automatisch gelöscht oder es wird ein Nullwert als Kundennummer eingetragen.
- Gibt es einen Kunden 4710, werden dessen Bestellungen gelöscht. Hatte er keine Bestellungen oder gibt es Kunden 4710 gar nicht, passiert nichts und der Aufruf gibt auch keine Fehlermeldung aus.
- Hatte 4710 Bestellungen, sollten auch diese gelöscht werden (kaskadierendes Löschen, vgl. Übung 3-12). Gibt es 4710 nicht, dann passiert nichts, auch keine Fehlermeldung.

Übung 3–12

Fällt ein Lieferant weg, müssen alle exklusiv von ihm gelieferten Artikel aus dem Sortiment gelöscht werden. Im Lehrer-Schüler-Klassen-Beispiel hat man ja eine hierarchisch-existentielle Beziehung von Schüler und Klasse (jeder Schüler ist in genau einer Klasse). Damit ist KlassenNr Fremdschlüssel in der Schülerrelation. Beschließt man jetzt z. B. für einen Jahrgang nur drei statt ursprünglich vier Klassen einzurichten und die Schüler der d-Klasse auf die Klassen a, b und c zu verteilen, dann wird man natürlich verhindern wollen, dass bei einer Löschung der Klasse *X*d in der Klassentabelle alle abhängigen Schülertupel in der Schülertabelle verschwinden. Stattdessen wird man gezielte UPDATE-Befehle auf die Schülertupel loslassen, deren KlassenNr *X*d ist und dort den neuen Wert *X*a, *X*b oder *X*c eintragen.

Übung 3–13

LNAME	BEZEICHNUNG
Müller	Mutter
Müller	Unterlegscheibe
Schmidt	Bolzen

Übung 3–14

In MITARBEITER wäre NAME oder MID als Schlüssel möglich. Wenn man aber schon einen MitarbeiterID einführt, wird man diesen auch als Primärschlüssel wählen, zumal noch Mitarbeiter mit bereits vorhandenen Namen hinzukommen könnten.

In ABTEILUNGEN ist ABTID bereits als Schlüssel unterstrichen. Dieses Attribut ist auch Fremdschlüssel in MITARBEITER und die referentielle Integrität erfordert, dass dort nur AbteilungsIds eingetragen werden, die auch in ABTEILUNGEN existieren.

Übung 3–15

Das Ergebnis im Fall der kurzen TLIST-Tabelle lautet:

NAME	FB	BEZ
Hans	17	Math-Inf
Emil	19	E-Tech

Übung 3–16

Der Equi-Join unter Beibehaltung beider Joinattribute lautet:

HNAME	STADT	KNAME	ORT
Schmidt KG	Hamburg	Meier	Hamburg
Schmidt KG	Hamburg	Wagner	Hamburg
Müller GmbH	München	Bauer	München

Übung 3–17

Das Ergebnis des zweifachen Joins über PROJEKT, ARBEITET_AN und MITARBEITER lautet wie folgt, wobei das Attribut ZEIT für eine übersichtlichere Darstellung nach rechts verschoben wurde.

PID	NAME	MNR	MNAME	ZEIT
4711	RZ-Ausbau	100	Hans	50
4711	RZ-Ausbau	250	Rolf	100
4712	PC-Versorgung	100	Hans	50
4712	PC-Versorgung	150	Gabi	100
4712	PC-Versorgung	200	Inge	100

Die Abarbeitungsreihenfolge ist beliebig, die Operation Join ist kommutativ. In der Praxis wird man versuchen, Zwischenergebnisse klein zu halten, um möglichst wenige Vergleiche über den Joinattributen machen zu müssen.

Übung 3–18

Analog zu oben.

Übung 3–19

Man wird zunächst eine Selektion mit PID=‘4711‘ auf ARBEITET_AN machen und die zwei resultierenden Tupel dann per Join mit MITARBEITER verknüpfen. Wäre auch der Projektname gefragt, wäre eine Selektion auf PROJEKT mit PID 4711 besser als ein allgemeiner natürlicher Join zuerst mit erst dann anschließender Selektion.

Übung 3–20

Bei einer Realisierung mittels äußerer und innerer Schleife durchläuft man für jedes der n Tupel der einen Tabelle alle m Tupel der anderen. Der Aufwand ist demnach $O(n \times m)$, also quadratisch. Im Falle einer Sort-Merge-Realisierung sortiert man beide Tabellen nach dem Joinattribut, was $O(n \log n)$ bzw. $O(m \log m)$ Schritte benötigt. Der Abgleich mittels Merge (Verschmelzen) benötigt dann zusätzlich $O(n + m)$ Vergleiche, d. h. asymptotisch dominiert das Sortieren. Liegt für beide Tabellen ein sequentiell durchlaufbarer Index über das Joinattribut vor, fällt nur der lineare Aufwand $O(n + m)$ für den Abgleich an.

Übung 3–21

Zu zeigen ist am Beispiel, dass $R \ltimes S = R \ltimes \pi_{R \cap S}(S)$.

R

A	B	C
a	b	c
d	b	c
b	b	f
c	a	d

S

B	C	D
b	c	d
b	c	e
a	d	b

S‘

B	C
b	c
a	d

S' ist die Projektion von S auf $R \cap S = \{B, C\}$. Der natürliche Join von S' mit R bringt das gewünschte Ergebnis.

R ⋈ S‘ = R ⋉ S

A	B	C
a	b	c
d	b	c
c	a	d

Übung 3–22

TLIST

NAME	FB	TEL
Hans	17	4477
Emil	19	3443
Gabi	71	4478

FBNAMEN

BEZ	FB
Math-Inf	17
E-Tech	19
Esoterik	99

TLIST * ⋉ * FBNAMEN

NAME	TEL	FB	BEZ
Hans	4477	17	Math-Inf
Emil	3443	19	E-Tech
Gabi	4478	71	?
?	?	99	Esoterik

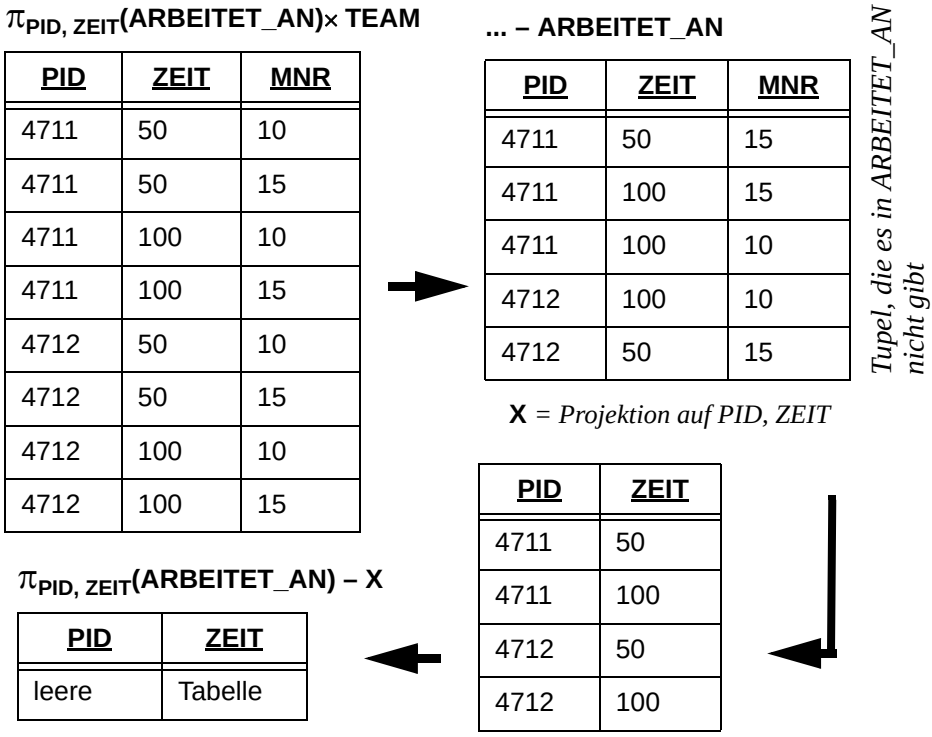
Übung 3–23

Siehe Übungsblatt.

Übung 3–24

Zur besseren Lesbarkeit des Beispiels wurden die Mitarbeiternummern um die letzte Ziffer gekürzt. Das Divisionsattribut ist weiterhin MNR

Das Beispiel ist leider nicht sehr intuitiv. Das Ergebnis wären Projekte und Zeitanteile der Mitarbeiter, die für alle Mitarbeiter in TEAM gelten. Da an 4711 Mitarbeiter 15 nicht teilnimmt, entfällt 4711 sowieso. Bei 4712 arbeitet Mitarbeiter 10 mit 50% und Mitarbeiter 15 mit 100%, dadurch gehören sie nicht beide entweder der Äquivalenzklasse (4712, 50) oder der Klasse (4712, 100) an. Somit ist das Divisionsergebnis leer.



Anders wäre es, wenn Mitarbeiter 15 nur mit 50% in 4712 arbeiten würde und die restlichen 50% garnicht, weil mit halber Stelle. In der Differenz ... – ARBEITET_AN (vgl. oben) fiele (4712, 50,15) raus, (4712, 100, 15) bliebe drin. Damit wäre in der Projektion *X* auch (4712, 50) nicht vertreten, somit bliebe nach Abzug von *X* noch (4712, 50) aus der Projektion von ARBEITET_AN übrig und ist das Ergebnis der Division. Nur für Projekt 4712 gilt, alle Mitarbeiter in TEAM arbeiten mit 50% Zeit daran.

Übung 3–25

Bei der Vereinigung bleibt der Grad gleich, die Kardinalität steigt bis hin zur Summe der Kardinalitäten der beteiligten Relationen bei Disjunktheit.

Bei der Produktbildung ist der Grad die Summe der Einzelgrade, die Kardinalität ist das Produkt der Kardinalitäten der Ausgangsrelationen.

Beim Theta-Join addiert sich auch der Grad, ggf. abzüglich der Anzahl identischer Joinattribute beim natürlichen Join. Der Grad kann von Null

(keinerlei Treffer für das Prädikat Θ) bis zum Produkt der Kardinalitäten reichen, wenn das Prädikat für alle Tupelkombinationen wahr ist.

Bei der Differenz $R - S$ bleibt der Grad gleich, die Kardinalität ist höchstens so groß wie die von *R*. Bei der Division $R \div S$ ist der Grad bestimmt durch die Nichtdivisionsattribute in *R*, demnach kleiner als der von *R*. Die Kardinalität ist kleiner oder gleich der von *R*.

Übung 3–26

$$\{t \mid t \in \text{FBNAMEN} \wedge \neg\exists\, s\ (s \in \text{TLIST} \wedge s[\text{FB}] = t[\text{FB}])\}$$

$$\{t \mid t \in \text{FBNAMEN} \wedge \forall\, s\ (s \in \text{TLIST} \wedge s[\text{FB}] \neq t[\text{FB}])\}$$

Übung 3–27

Namen von Mitarbeitern, die mit weniger als 100% ihrer Arbeitszeit an der PC-Versorgung beteiligt sind.

$$\{t \mid \exists\, s\ (s \in \text{MITARBEITER} \wedge \exists\, u\ (u \in \text{ARBEITET\_AN}) \wedge \exists\, v\ (v \in \text{PROJEKT} \wedge v[\text{NAME}] = \text{'PC-Versorgung'} \wedge v[\text{PID}] = u[\text{PID}] \wedge u[\text{ZEIT}] < 100 \wedge u[\text{MNR}] = s[\text{MNR}]) \wedge s[\text{MNAME}] = t[\text{MNAME}])\}$$

Übung 3–28

$$\{t \mid t \in \text{LIEFERANTEN} \wedge \forall\, r\ (r \in \text{ARTIKEL} \wedge r[\text{LNR}] = t[\text{LNR}] \wedge \neg\exists\, s\ (s \in \text{BESTELLUNG} \wedge s[\text{ARTNR}] = r[\text{ARTNR}])\}$$

Übung 3–29

R* (ohne Tupel mit identischen X und Y Werten)

X	Y
1	2
1	3
2	3
2	1
3	1
3	2

X	Y
4	1
4	2
4	3

Nein, geht über Prädikatenlogik 1. Stufe hinaus im allgemeinen Fall, da sich keine obere Schranke für die Anzahl der Schritte, d.h. für die Einfügungen ... $\wedge\ \exists\ t_i\ (t_i \in R \wedge t_i[X] = t_j[Y]\ \ldots)$, angeben läßt.

Übung 3–30

Geht nicht, da eine Arithmetik im Kalkül nicht vorgesehen ist. Ist in SQL über Erweiterungen mittels COUNT-, SUM- und AVG-Funktionen und GROUP BY-Befehlen möglich.

Übung 4–1

Relation statt Tabelle, Tupel für Zeile, Attribut für Spalte, Attributwert für Feld.

Übung 4–2

```
CREATE TABLE TLIST(NAME VARCHAR2(30) NOT NULL,
 FB NUMERIC(2,0),
 TEL NUMERIC(4,0));
```

Anmerkung: In dieser speziellen Telefonliste waren die Telefonnummern offensichtlich 4-stellige Nebenstellennummern. Allgemein kann man Telefonnummern als numerische Werte oder als Zeichenketten abspeichern, was Vor- und Nachteile bzgl. Klammern, Bindestriche, Leerstellen, führender Nullen bei der Vorwahl und der Formulierung gültiger Abfragen hat.

```
CREATE TABLE FBNAMEN(FB NUMERIC(2,0) NOT NULL,
 BEZ VARCHAR2(30));
```

Lehrer, Schüler, Klassen analog.

Übung 4–3

```
CREATE VIEW Kundensicht(KNUMMER, KNAMEORT) AS
 SELECT KDNR, KNAME + KDADRESSE
 FROM Kunden;
```

Üblicher ist allerdings seit SQL99 der Verkettungsoperator „||“.

Übung 4–4

```
CREATE VIEW Preise (ARTNR, ARTBEZ, LNR, NETTOPREIS,
    BRUTTOPREIS, FUENETTOPREIS) AS
 SELECT ARTNR, ARTBEZ, LNR, NETTOPREIS,
    NETTOPREIS*1.19, NETTOPREIS*0.6*1.19
 FROM Artikel;
```

Übung 4–5

Nein, der SQL-Updatebefehl wird mit der Viewdefinition verbunden, wodurch aus logischer Sicht eine WHERE-Klausel mit

```
WHERE NAME = 'Mueller' AND ALTER > 40
```

entsteht, die Updates für Angestellte mit Alter ≤ 40 verhindert.

Übung 4–6

Wird nicht auch das UPDATE-Recht entzogen, kann ein Anwender das Tupel mit Fantasiewerten oder der DB-Null füllen, was einem DELETE fast gleichkommt.

Übung 4–7

COUNT (DISTINCT ...) verhindert die Zählung von Duplikaten. Gefragt ist ja die Anzahl der vorhandenen Telefone pro Fachbereich, nicht die Anzahl der über diese Telefone erreichbaren Personen, die höher ist, wenn sich mehrere ein Telefon über eine Nummer teilen.

Übung 4–8

Im ersten Fall soviele Pärchen von Fachbereichsnummern und zugehörigen Fachbereichsbezeichnungen, wie es Teilnehmer in TLIST gibt, also mit sehr vielen Duplikaten. Im letzteren Fall für jeden Fachbereich nur ein Tupel.

Übung 4–9

```
INSERT INTO FBNAMEN VALUES (25, 'Informatik');
```

Übung 4–10

SQL2 wird von allen Herstellern inzwischen fast ganz abgedeckt. Die Online-Dokumentationen der Hersteller führen daher meist nur noch die Unterschiede zum Nachfolgestandard SQL3 auf. Kleinere Abweichungen gibt es aber immer, so akzeptiert Oracle z. B. weiterhin kein Schlüsselwort AS zwischen Tabellennamen und Alias.

Übung 4–11

TLIST	NAME	FB	TEL
P.	_X		_V

TLIST	NAME	FB	TEL
	_Y		_W

Bedingungen
_V = _W _X <> _Y

Übung 5–1

TITEL	VORNAME	NACHNAME	STICHWORT
Algorithmen und Datenstrukturen	Thomas	Ottmann	Sortieren
Algorithmen und Datenstrukturen	Thomas	Ottmann	Suchen
Algorithmen und Datenstrukturen	Peter	Widmayer	Sortieren
Algorithmen und Datenstrukturen	Peter	Widmayer	Suchen
Tcl und Tk	John K.	Ousterhout	Skriptsprachen

TITEL	VORNAME	NACHNAME	STICHWORT
Tcl und Tk	John K.	Ousterhout	grafische Benutzer-schnittstellen
Tcl und Tk	John K.	Ousterhout	Widgets
Oracle Database 11g Release 2 Doc.	-	-	Online documenta-tion
Oracle Database 11g Release 2 Doc.	-	-	SQL

vgl. Übung 5-2 unten.

Übung 5–2

{}BÜCHER			
TITEL	{}STICHWÖRTER	{}AUTOREN	
	STICHWORT	VORNAME	NACHNAME
Algorithmen und Datenstrukturen	Sortieren Suchen	Thomas Peter	Ottmann Widmayer
Tcl und Tk	Skriptsprachen grafische Benutzer-schnittstellen Widgets	John K.	Ousterhout
Oracle Database 11g Release 2 Doc.	Online documenta-tion SQL	{ }	

Übung 5–3

Vorausgesetzt NAME ist alleiniger Schlüsselkandidat, folgt daraus, dass M2 in 2NF ist, weil nur bei zusammengesetzten Schlüsseln die 2. NF verletzt werden kann.

Übung 5–4

Die einzige nichttriviale FA ist {NAME, FB} $\rightarrow$ MITGLIED und geht damit vom Schlüssel aus. Also ist F-M-2 in 3NF.

Übung 5–5

In M' galt ORT bestimmt ADRESSE und umgekehrt, NAME und STATUS waren nicht abhängig von ORT, NAME war alleiniger Schlüsselkandidat. Damit ist die transitive Abhängigkeit

$$\text{NAME} \quad \not\rightleftarrows \quad \text{ORT} \rightarrow \text{ADRESSE}$$

gegeben.

Übung 5–6

F-M-2 ist auch in BCNF, da in der einzigen FA der Schlüssel {NAME, FB} das Attribut MITGLIED bestimmt. Im Fall, dass man nur in einem weiteren FB Zweitmitglied werden kann (was üblicherweise nicht gilt), gäbe es auch die FA {NAME, MITGLIED} $\rightarrow$ FB. Allerdings wäre dann auch {NAME, MITGLIED} ein Schlüsselkandidat und somit auch wieder BCNF gegeben.

Übung 5–7

Jedes der drei Attribute in FNT ist prim (Teil eines Schlüssels), damit ist die Tabelle in 3NF. TEL alleine ist jedoch kein Schlüssel, damit verletzt die Abhängigkeit TEL $\rightarrow$ FB die Bedingung für BCNF.

Übung 5–8

Man kann FNT in FN(FB, NAME) und NT(NAME, TEL) teilen. Man sieht aber, dass NT „unpraktisch“ ist, weil einem bei doppelten Einträgen, etwa für „Hans“, die Fachbereichszugehörigkeit fehlt, aus der klar würde, welche der beiden Telefonnummer die gesuchte ist. In Beispiel 5-10 unten wird formal gezeigt, dass auch der nachträgliche Join von FN mit NT diese Information aus FNT nicht zurückbringt, d. h. die Aufteilung ist nicht verlustfrei. Zusätzlich ist die Aufteilung auch nicht abhängigkeitserhaltend.

Wie weiter unten in Kapitel 5 gezeigt, ist die Aufteilung FT(FB, TEL) und NT(NAME, TEL) auch möglich und verlustfrei, aber ebenso nicht abhängigkeitserhaltend.

Übung 5–9

Das Postleitzahlensystem ist so aufgebaut, dass PLZ → STADT und {STADT, STRASSE (ggf. mit Hausnummernbereich)} → PLZ.

STADT	STRASSE	PLZ
Kassel	Oberzwehrener Str.	34132
Kassel	Heinrich-Plett-Straße	34132
Kassel	Opernplatz	34117
Hannover	Opernplatz	30159

Alle drei Attribute sind prim (Teil eines Schlüssels), damit ist 3NF gegeben. Die FA {STRASSE, PLZ} → STADT ist eine partielle Abhängigkeit, weil schon PLZ → STADT gilt. Damit hängt STADT nicht voll vom Schlüssel ab und somit ist die Tabelle nicht in BCNF.

Übung 5–10

BESTPOSTEN(BESTNR, DATUM, KDNR, POS, ARTNR, MENGE)

Schlüsselkandidaten: {BESTNR, POS}, auch {BESTNR, ARTNR}, wenn alle gleichen Artikel einer Bestellung unter einer Position zusammengefasst sind.

BESTNR → DATUM, BESTNR → KDNR und die Schlüsselabhängigkeiten, z. B. {BESTNR, POS} → DATUM.

Dann ist BESTPOSTEN nicht in 2NF, weil DATUM nicht voll funktional von jedem Schlüssel abhängt, sondern partiell bereits von BESTNR.

Übung 5–11

Aufteilung in BESTELLUNGEN(<u>BESTNR</u>, DATUM, KDNR) und POSTEN(<u>BESTNR</u>, <u>POS</u>, ARTNR, MENGE). Schlüssel sind unterstrichen. In BESTELLUNGEN gibt es nur Schlüsselabhängigkeiten, DATUM zusammen mit KDNR bestimmen **nicht** BESTNR, weil ein Kunde zweimal am selben Tag bestellen kann. Damit ist BESTELLUNGEN sogar in BCNF.

Auch in BESTELLUNGEN gibt es nur voll funktionale Schlüsselabhängigkeiten und auch wenn zu {BESTNR, POS} man noch {BESTNR, ARTNR} als Schlüsselkandidaten hinzunimmt (vgl. Begründung in Übung 5-10), bleibt dies so. Demnach ist auch POSTEN in BCNF.

Übung 5–12

Gegeben ist R= {FB, NAME, TEL} mit den funktionalen Abhängigkeiten F = {{FB, NAME} → {TEL}, {TEL} → {FB}}. Zu zeigen ist, mit welchen Axiomen und Regeln man F^+ im Beispiel errechnet. Bezeichne **R**, **E**, **T** die Axiome für Reflexivität, Erweiterung und Transitivität. **V**, **P**, **A** stehe für die Regeln Vereinigung, Pseudotransitivität und Aufspaltung. An die erzeugten FA haben wir unten die zutreffende Kennungen angefügt.

- {TEL} → {TEL}, {FB} → {FB}, {NAME} → {NAME} (alle **R** trivial)
  {TEL} → {FB} und {TEL} → {FB, TEL} (**V** mit TEL → TEL)
- {FB, NAME} → {FB, NAME, TEL} (**V**)
  {FB, NAME} → {FB, NAME} (**R** trivial)
  {FB, NAME} → {NAME, TEL} (**V** mit {FB, NAME}→NAME)
  {FB, NAME} → {FB, TEL} (**V** analog)
  {FB, NAME} → {TEL} (so gegeben)
  {FB, NAME} → {FB} und {FB, NAME} → {NAME} (beide **R**)
- {TEL, NAME}→{FB, NAME, TEL} (aus {TEL}→{FB} mit **E** und **V**)
  {TEL, NAME} → {TEL, NAME} (**R** trivial)
  {TEL, NAME} → {FB, NAME} (mit **E**)
  {TEL, NAME} → {FB} (aus oben mit **A**)
  {TEL, NAME} → {TEL} und {TEL, NAME} → {NAME} (beide **R** trivial)
- {FB, TEL} → *Y*, für alle $Y \subseteq$ {FB, TEL} (mit **R** trivial)
- {FB, NAME, TEL} → *Y*, für alle $Y \subseteq R$ (mit **R** trivial)

Übung 5–13

Betrachtet wird die Aufteilung von FNT in FN(FB, NAME) und NT(NAME, TEL) wie in Übung 5-8 vorgeschlagen.

$F_1 = F_{FN}$ = {{NAME, FB} → NAME, {NAME, FB} → FB} (alle trivial)
$F_2 = F_{NT}$ = {{NAME, TEL} → TEL, {NAME, TEL} → NAME} (dito)

Aus $(F_1 \cup F_2)^+$ läßt sich nicht {NAME, FB} → TEL, also die Annahme, es gebe keine zwei Personen mit gleichem Namen in einem Fachbereich und jeder habe höchstens ein Telefon, ableiten. Zwar sieht die Pseudotransitivität $X \rightarrow Y$ und $WY \rightarrow Z$ impliziert $WX \rightarrow Z$ verlockend aus, aus {NAME, FB} → NAME und {NAME, TEL} → TEL, folgt damit aber nur {NAME, FB, TEL} → TEL und das Attribut TEL links darf man natürlich nicht einfach streichen.

Übung 5–14

In Übung 5-11 haben wir BESTPOSTEN aufgeteilt in R_1 = BESTELLUNGEN(BESTNR, DATUM, KDNR) und R_2 = POSTEN(BESTNR, POS, ARTNR, MENGE). Die Schlüssel sind unterstrichen. Zunächst ist $R_1 \cap R_2$ = {BESTNR} und {BESTNR} → R_1, weil BESTNR Schlüssel ist. Demnach ist die Aufteilung verlustfrei.

Die in BESTPOSTEN zunächst geltenden FA waren BESTNR → DATUM, BESTNR → KDNR und die von {BESTNR, POS}, bzw. {BESTNR, ARTNR} ausgehenden Schlüsselabhängigkeiten. Die ersten beiden FA lassen sich in R_1 lokal überprüfen, weil BESTNR dort Schlüssel ist.

Bleibt die Frage, ob auch die alten Schlüsselabhängigkeiten, z.B. {BESTNR, POS} → DATUM, {BESTNR, ARTNR} → KDNR lokal überprüfbar sind. In R_2 gilt dies sicher für alle Attribute, weil die Determinanten Schlüsselkandidaten sind. In R_1 haben wir aber schon BESTNR → DATUM und BESTNR → KDNR, also ist auch über das Erweiterungsaxiom {BESTNR, POS} → DATUM und {BESTNR, ARTNR} → KDNR in $(F_1 \cup F_2)^+$. Insgesamt wurde so die Äquivalenz zu F^+ gezeigt, d. h. die Aufteilung ist auch abhängigkeitserhaltend. Aus gutem Grund ist dies ja

auch eine Form der Tabellengestaltung, wie man sie im Bestellwesen oft findet.

Übung 5–15

Die Büchertabelle mit Schema R = {TITEL, AUTOR, STICHWORT} wird man aufteilen in R_1 = {TITEL, AUTOR} und R_2 = {TITEL, STICHWORT}, wobei angenommen wird, dass TITEL in R Schlüsseleigenschaft hat. Besser wäre es allerdings einen kurzen, künstlichen Schlüssel, etwa eine ISBN, einzuführen. Ein Join über den TITEL (oder eben besser die ISBN) bringt dann alle Autoren eines Buchs mit allen Stichworten zusammen.

Übung 6–1

Härder und Rahm geben in ihrem Klassiker „Datenbanksysteme - Konzepte und Techniken der Implementierung, 2. überarb. Aufl. Springer 2001, auf S. 471 für den Fall des Gruppen-Commit an, „dass sich durch das verzögerte Schreiben der Log-Daten i. allg. eine Erhöhung der Sperrdauer und Antwortzeit ergibt.“ Der wesentliche Vorteil des verzögerten Updates ist das entfallende Undo im Fall eines Abbruchs vor dem Commit. Die bis dahin ins Log geschriebenen Änderungen werden in diesem Fall einfach ignoriert.

Übung 6–2

Häufig arbeiten Texteditoren auf Kopien des Dokuments. Ggf. werden periodisch Sicherungskopien erstellt (*xyz*.backup). Bei einem expliziten Speichern wird ggf. zuerst nochmals eine Sicherungskopie geschrieben und bei Erfolg vertauschen im Verzeichnis Originaleintrag und Kopieeintrag im Idealfall atomar den Zeiger auf die Datenblöcke, also ähnlich einem Schattenseitentausch.

Übung 6–3

Bei sehr großen Multimediadaten wird man bei kleinen Änderungen gerne ein Vorher-/Nachher-Image vermeiden und nur die reversiblen Änderungsaktionen aufzeichnen und erst mit Abschluss der Bearbeitung

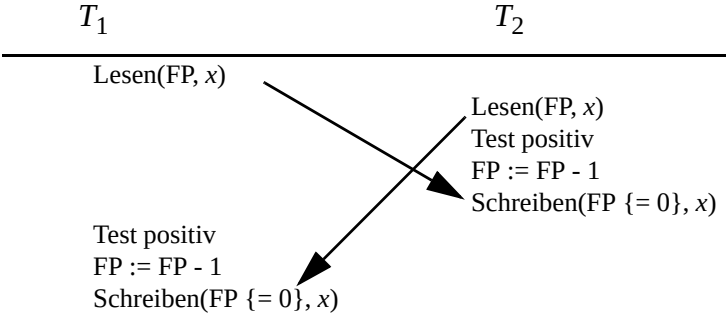
ein komplettes neues Abbild anlegen. Allerdings ist Speicher heute billig, so dass sich der Aufwand oft nicht lohnt.

Übung 6–4

Grundsätzlich entsprechen Log-Dateien einem Buchungspfad. Noch besser wären versionierte Datenbanksysteme, bei denen jeder einmal eingetragene Datenwert mit einem Gültigkeitsdatum (gültig ab ..., valid as-of ...) versehen ist und Daten nicht überschrieben werden.

Übung 6–5

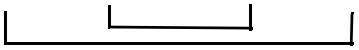
Die folgende Ablauffolge ist nicht serialisierbar, weil durch die verschränkten Lese-/Schreibzugriffe auf Datensatz *x* ein Zyklus von T_1 zu T_2 und zurück entsteht.



Übung 6–6

In F_a ist $R_2(B) < W_2(B) < R_1(B) < W_1(B)$, d. h. bei allen konfliktären Paaren (zwei unterschiedliche Transaktionen, selber Datensatz, mindestens eine Schreiboperation) ist T_2 vor T_1, damit kein Zyklus und F_a ist serialisierbar.

In F_b ist $R_2(B) < R_1(B) < W_2(B) < W_1(B)$.



Wie angedeutet, ist einmal T_2 vor T_1 und im anderen Paar umgekehrt T_1 vor T_2. Demnach gibt es einen Zyklus und F_b ist nicht serialisierbar. Führt man das Beispiel mit Kontenständen von 100, 200 und 300 € für Konten

(Datensätze) A, B und C durch, dann ist der Anfangsbestand 600 €, nach Ausführung der Bestand aber 620 €, obwohl nur Umbuchungen stattfanden. Die wundersame Geldvermehrung beruht also darauf, dass durch die Verschränkung eine Transaktion Werte sieht und verändert zurückschreibt, die sie bei serieller Ausführung so nicht gesehen hätte.

Übung 6–7

F_1 enthält die folgenden vier konfliktären Paare: ① $R_1(A) < W_2(A)$ ② $W_1(A) < R_2(A)$ ③ $R_1(B) < W_2(B)$ ④ $W_1(B) < R_2(B)$. Im Abhängigkeitsgraph gehen demnach nur Kanten von T_1 nach T_2, womit er keine Zyklen hat und F_1 serialisierbar ist. Die in F_1 gezeigte Verschränkung wäre mit 2PL allerdings nicht herstellbar, da T_1 zunächst eine Lesesperre auf A erlangt, diese zu einer Schreibsperre ausweitet (eskaliert), wonach T_2 nicht mehr lesend auf A zugreifen kann. T_1 hält diese Sperre bis auch B gelesen und geschrieben wurde. Immerhin kommt es in F_1 zu keiner gegenseitigen Blockade, denn T_2 bricht entweder ab oder wartet, bis A und B frei werden. Danach läuft T_2 durch, sofern nicht andere Transaktionen wieder dazwischen kommen.

Übung 6–8

F_2 ist nicht serialisierbar. $R_1(A) < W_2(A)$ und $R_2(B) < W_1(B)$ ergeben einen Zyklus. Die Folge ist trotzdem logisch korrekt, weil sie eigentlich aus zwei getrennten, abgeschlossenen Unterfolgen besteht, die jede für sich serialisierbar sind, die erste mit $T_1 < T_2$ und die zweite umgekehrt. Vor der scheinbar konfliktären Schreiboperation wird der Wert in dieser Unterfolge nochmals gelesen, so dass die von der anderen Transaktion ausgeführte Schreiboperation schon berücksichtigt ist.

Übung 6–9

Wir nehmen an TS(T_1) = 10:30 Uhr < TS(T_2) = 10:45 Uhr und F_1 = (1) $R_1(A)$, (2) $W_1(A)$, (3) $R_2(A)$, (4) $W_2(A)$, (5) $R_1(B)$, (6) $W_1(B)$, (7) $R_2(B)$, (8) W_2 (B) wie oben angegeben. Dann erfolgen die Einträge der Lese-/Schreibmarken an den Sätzen wie folgt und F_1 geht durch.

RTM(A)	WTM(A)	RTM(B)	RTM(B)
10:...	10:...	10:...	10:...
10:30 (1)	10:30 (2)	10:30 (5)	10:30 (6)
10:45 (3)	10:45 (4)	10:45 (7)	10:45 (8)

Übung 7–1

Heutige RAM-Speicher sind deutlich schneller (ca. 10 ns, Stand Herbst 2011), Festplatten sind nur z. T. schneller geworden, sehr flotte Exemplare schaffen 8 ms. Damit hat sich die Geschwindigkeitsschere eher weiter geöffnet gegenüber den Zahlen im Skript. Die konkreten Zahlen sind Teil einer Übungsaufgabe.[1]

Übung 7–2

Siehe Übungsblatt.

Übung 7–3

Die zufällige Drehwartezeit entspricht einer halben Umdrehung.

Übung 7–4

Ein System mit N unabhängigen Platten, über die hinweg bit-striping erfolgt, arbeitet nur dann fehlerfrei, wenn keine der Platten ausfällt. Ist die Ausfallwahrscheinlichkeit p einer Platte innerhalb eines Zeitintervalls X bekannt, dann überlebt das System die Zeit X mit Wahrscheinlichkeit $(1 - p)^N$. Für $1- p = 0.999$ ergibt sich bei $N = 8$ nur noch 0.992.

1. Jon Bentley hat in seinem Buch *Perlen der Programmierkunst - Programming Pearls* (Addison-Wesley 2000) diese Art der überschlägigen Kalkulation (back-of-the-envelope calculation) berühmt gemacht.

Übung 7–5

Bei dem Verteilungsschema mit einfacher Replikation (b) kann nach Verlust von Platte Nr. 1 nur noch der Verlust von Nr. 3 toleriert werden. Im Fall von (c) kann kein weiterer Ausfall ausgeglichen werden. Die Vorteile dieser Replikation liegen daher mehr in der Lastverteilung beim Lesen.

Übung 7–6

XOR-Verknüpfung als Replikation

Platte 1	Platte 2	Platte 3
0	0	0
0	1	1
1	0	1
1	1	0

Man sieht an der Wertetabelle links, dass die XOR-Verknüpfung symmetrisch ist. Damit kann der Ausfall einer der drei Platten verkraftet werden, der zusätzliche Speicheraufwand beträgt nur 50% des Nettodatenvolumens. Geschrieben wird entweder auf alle drei Platten gleichzeitig (je zwei Nutzbytes und ein XOR-Byte), oder ein schon existierendes Byte wird gelesen und zwei Platten beschrieben (1 Nutzbyte und ein XOR-Byte).

Übung 7–7

Besonders portabel sind sicher solche DBMS, die auf vorgegebene Dateisysteme des Betriebssystems aufsetzen, da diese in der Regel aufwärtskompatibel weiterentwickelt werden. Geschwindigkeitsvorteile bringen dagegen Implementierungen, die ohne die Zwischenschicht der Dateiverwaltung auskommen. Ob sich Bestrebungen durchsetzen werden, schon in die Dateiverwaltung der Betriebssysteme Funktionalität einzubauen, die sonst erst Datenbanksysteme bieten (z. B. Recovery), muss sich zeigen. Ein Vertreter davon wäre das Linux-Dateisystem Btrfs.

Übung 7–8

Beladys Strategie (optimale Strategie), Zugriffe fett, 9 Ladevorgänge

1	1	1	1	**3**	3	3	**3**	3	**5**	5	5	**5**	5	5	**3**
?	**2**	**2**	2	2	2	**2**	2	2	2	**1**	1	1	1	**1**	1

?	?	?	**5**	5	**4**	4	4	**4**	4	4	**4**	4	**2**	2	2

LRU (längste Zeit nicht genutzt), Zugriffe fett, 11 Ladevorgänge

1	1	1	1	**3**	3	3	**3**	3	3	**1**	1	1	**2**	2	2
?	**2**	**2**	2	2	**4**	4	4	**4**	4	4	**4**	4	4	**1**	1
?	?	?	**5**	5	5	**2**	2	2	**5**	5	5	**5**	5	5	**3**

FIFO (nach Ladealter), Zugriffe fett, 11 Ladevorgänge

1	1	1	1	**3**	3	3	**3**	3	3	**1**	1	1	**2**	2	2
?	**2**	**2**	2	2	**4**	4	4	**4**	4	4	**4**	4	4	**1**	1
?	?	?	**5**	5	5	**2**	2	2	**5**	5	5	**5**	5	5	**3**

Zähler (zirkuläre FIFO-Liste), Unterstreichung=used bit 1, 11 Ladevorg.

1	1	1	1	**3**	3	3	**3**	3	**5**	5	5	**5**	**2**	2	2
?	**2**	**2**	2	2	**4**	4	4	**4**	4	**1**	1	1	1	**1**	1
?	?	?	**5**	5	5	**2**	2	2	2	2	**4**	4	4	4	**3**

Übung 7–9

Mit 3 Byte für die Seitennummer gibt es 2^{24} Seiten. Eine 4 KB-Seite hat $4096 = 2^{12}$ Byte, damit ist die Dateigröße $2^{24} \cdot 2^{12} = 2^{36} = 64$ GB. Die Dateigröße bei 8 KB Seiten ist dann doppelt so groß, also 128 GB, für heutige Verhältnisse insgesamt zu gering. Man beachte aber, dass man mit 4 Byte für ein RID bei dieser Organisationsform mehr als $2^{32} = 4$ GB adressieren kann.

Übung 7–10

Nimmt man für die Sloteinträge in der seiteninternen Tabelle 2 Byte und sieht davon 3 Bit als Flags vor, dann lassen sich Seiten der Größe $2^{13} = 8$ KB adressieren. Interpretiert man die Sloteinträge als Wortanfänge wären Seiten der vierfachen Größe, also 32 KB adressierbar, das RID-Konzept würde dann 512 GB Dateien ermöglichen.

Übung 7–11

Das RID für das Freiobjekt in Seite *x* des Beispiels ist *x*3. Seine Größe ergibt sich aus „Start der Slotliste“ minus „Startadresse in Slot 3“, also zu 4088 - 424 = 3664 Byte.

Übung 7–12

Bei einer Tabellengröße von $5 \cdot 10^4$ Seiten und einem Anteil von $3 \cdot 10^3$ Seiten im Hauptspeicher liegt die Trefferquote für zufälligen Zugriff bei 3/50 = 0.06, also 6%. In der Praxis wird die Trefferquote durch Lokalitätsverhalten höher liegen und man wird heute deutlich größere Caches haben. Trotzdem würde eine Datenverwaltung mittels RID und 10% umgezogene Seiten (die einen zweiten Plattenzugriff erfordern) besser abschneiden.

Für Tupel mit einer durchschnittlichen Größe von 2 KB gäbe es nur 10^{10} / $2 \cdot 10^3 = 5 \cdot 10^6$ Einträge zu je 4 Byte, also eine Tabelle der Größe $2 \cdot 10^7$ Byte oder $2{,}5 \cdot 10^3$ Seiten. Das folgt auch aus der Tatsache, dass sich die Tupellänge um den Faktor 20 erhöht hat, was eine Verkleinerung der Tabelle um den selben Faktor zur Folge hat ($20 \cdot 2{,}5 \cdot 10^3$ ergibt die alte Tabellengröße $5 \cdot 10^4$ Seiten zu 8 KB). Damit passt sie vollständig in den Cache von 3000 Seiten und schlägt so das RID-Verfahren immer.

Übung 7–13

Innerhalb der (Offset, RID) Listen kann man *Binäre Suche* für das Finden einer Position *x* verwenden. Das kann sich bei einem genügend großen Verzweigungsgrad gegenüber sequentieller Suche durchaus lohnen (vgl. B*-Bäume). Auch entfällt das Aufsummieren der Teilgrößen bis zum Wert > *x* im Vergleich zur Längenspeicherung.

Beim Einfügen und Entfernen sind bei Verzweigung über Offsets auf dem Pfad von der Wurzel bis zum Blatt (der Einfügestelle) **alle Angaben** (Offsets) **rechts von der Einfügeposition** zu korrigieren (aber nur in die Knoten auf dem Pfad). Hier ist die Speicherung der Teillängen im Vorteil, weil bei Längenänderungen auf dem Pfad in jeder Verzweigungsliste **nur ein Wert** zu ändern ist.

Übung 8–1

Die klassische Sicht der relationalen Datenbanksysteme beruht auf der Identifizierung eines Objekts über seine Schlüsselwerte. Entitäten, die sich nicht in den beobachtbaren (und gespeicherten!) Werten unterscheiden, sind als identisch anzusehen. Das macht anonymisierte Statistikauswertungen schwieriger, ist aber sicherer als eine Identifikation, die mit extern vergebenen Objektidentifiern arbeitet (zwei Flugzeugen wird die zeitgleiche Landung auf der selben Landebahn zugewiesen, der Datensatz existiert zweimal mit unterschiedlichen Objektidentifiern, was beim Crash herzlich wenig nützt). Beide Lösungen erfordern eine Umsetzung in tatsächliche Speicheradressen.

Die entfällt bei einer Identifikation über Adressen, was die Verarbeitung schnell macht. Allerdings ist diese Objektidentifikation wenig flexibel und nicht portabel.

Übung 8–2

Von der Abteilung zeigt ein Link auf den ersten Angestellten dieser Abteilung. In den Angestellten-Sätzen zeigt ein Angestellten-Link auf den nächsten Angestellten, beim letzten in der Kette zeigt der Link ggf. zurück zur Abteilung oder ist nil. Einer der Angestelltensätze hat das Managerbit auf 1 gesetzt, zusätzlich kann ein Managerlink von Abteilung auf diesen Angestelltensatz zeigen.

Übung 8–3

Die BEARBEITEN-Relation, die im relationalen Modell die Verbindung einer *n*:*m*-Beziehung herstellt, wäre um die Fremdschlüssel ANR und PID zu ergänzen. ABTEILUNGEN und PROJEKTE könnten so bleiben.

Übung 8–4

Die Verbindungssätze, die im Netzwerkmodell die n:m-Beziehung Schulklassen-Lehrer herstellt, könnte als Attribut Fach und Stunden-im-Fach enthalten. Verkettungen enthalten alle von einem Lehrer zu unterrichtenden Fächer mit Stundenanzahl, sowie alle in einer Schulklasse zu unterrichtenden Fächer mit Stundenanzahl. Dafür benötigt jeder Verbindungssatz zwei Links.

Übung 8–5

Die logischen Satzformate für LIEFERANT und ARTIKEL im Netzwerkmodell wären

LIEFERANT(LID, LNAME, LORT)
ARTIKEL(ANR, ABEZ)

Für TEIL_VON gibt es mindestens zwei Lösungen. Zum Ausdruck der is-a-Beziehung fehlt der Satz ganz und wird nur durch einen Link (ist-enthalten-in) von einem Artikel zu seinem Vaterartikel realisiert. Dann findet man allerdings nicht schnell für einen Sammelartikel (Vater) alle seine enthaltenen Unterartikel (Söhne). Alternativ legt man einen Verbindungssatz je enthaltenem Unterartikel an (ohne Attribute) und zeigt mit einem Link auf den betreffenden Artikel sowie mit einem zweiten Link auf den Nachfolger in der Kette all der Artikel, die zusammen die Untermenge bilden. Alle Artikel haben einen Link auf Sätze TEIL_VON, aber nur wenn ein Artikel tatsächlich aus Unterartikeln zusammengesetzt ist, ist ein Wert im Link eingetragen und zeigt dann auf den ersten TEIL_VON-Satz (Sohn). Vorstellbar wäre auch ein Attribut MENGE in TEIL_VON, wenn ein Artikel mehrfach als Unterteil in einem anderen Artikel enthalten ist.

Übung 8–6

```
02 P-BEZEICHNUNG IS VIRTUAL
SOURCE IS PNAME OF OWNER OF P-BEARBEITEN
```

Übung 8–7

```
RECORD NAME IS ARTIKEL;
  WITHIN BASIC-DATA-AREA;
  02 ANR; TYPE IS CHARACTER 6.
  02 ANAME; TYPE IS CHARACTER 30.
  02 APREIS; TYPE IS FIXED DECIMAL 9.

RECORD NAME IS LIEFERANT;
  WITHIN BASIC-DATA-AREA;
  02 LID; TYPE IS CHARACTER 6.
  02 LNAME; TYPE IS CHARACTER 30.
  02 LORT; TYPE IS CHARACTER 30.
```

```
RECORD NAME IS LIEFERT;
  WITHIN LINK-DATA-AREA;
  02 SEIT-WANN; TYPE IS CHARACTER 12.

SET NAME IS L-LIEFERT;
  OWNER IS LIEFERANT;
  ORDER IS PERMANENT SORTED BY ...
  MEMBER IS LIEFERT
```

Es wird angenommen, jeder Artikel habe genau einen Lieferanten. Das schnelle Auffinden eines Lieferanten für einen gegebenen Artikel wird hier allerdings nicht gezielt unterstützt.

Übung 8–8

Ein Satz, der Besitzer zweier DBTG-Mengen ist und selbst in drei DBTG-Member Mitglied ist, benötigt fünf Link-Felder.

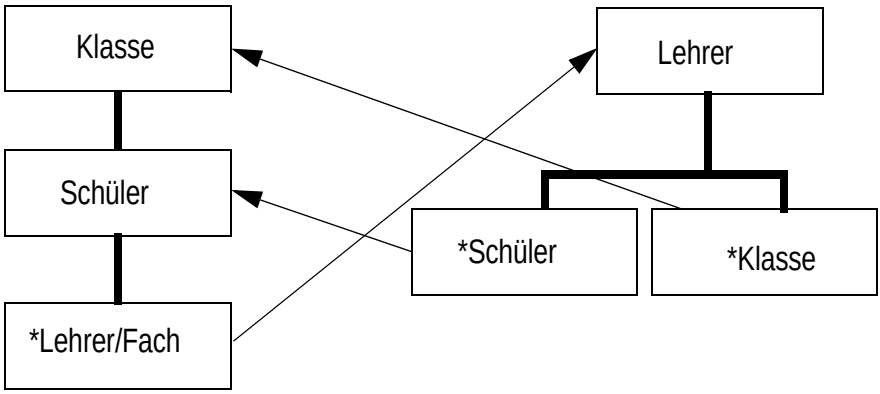
Übung 8–9

Die Fädelung der Blätter eines Baums zur schnelleren Traversierung ähnelt der Implementierung von DBTG-Mengen mittels Ringstruktur. Bei der Fädelung nutzt man die mit nil besetzten Linkfelder auf linken oder rechten Sohn und speichert dort einen Zeiger auf den Vorgänger oder Nachfolger in Inorder (Linker Teilbaum - Wurzel - Rechter Teilbaum). Das setzt voraus, dass man Fädelungspointer von Zeigern auf Söhne unterscheiden kann.

Übung 8–10

Würde man im Vaterknoten einen Zeiger je Sohnknoten unterbringen, müsste man eine variable Anzahl an Zeigern speichern können, da die Anzahl der Mitarbeiter und Geräte je Abteilung, sowie der Projekte je Mitarbeiter, beliebig groß sein kann. Man behilft sich deshalb wieder mit einer Ringstruktur, bzw. speichert die Daten in Preorder sequentiell ab (Preorder: Wurzel - Linker Teilbaum - Rechter Teilbaum). Die Preorder-auflistung des Baum in Abb. 8-6 ist a_1, m_{11}, p_{111}, p_{112}, p_{113}, m_{12}, p_{121}, p_{122}, g_{11}, g_{12}; a_2, m_{21}, ...

Übung 8–11



Zwischen Klasse und Schüler gibt es eine echte hierarchische Beziehung (jeder Schüler ist in genau einer Klasse gemäß Miniweltannahme). Lehrer unterrichten in vielen Klassen, Klassen haben viele Lehrer, genauso ist Schüler-Lehrer eine *n*:*m*-Beziehung. Diese erscheinen jetzt als virtueller Satztyp **T*. Bei *Lehrer haben wir als echtes Datenfeld das Fach, in dem ein Schüler von einem Lehrer unterrichtet wird, hinzugenommen.

Übung 8–12

Die Klasse Schüler wurde erweitert um zu hörende Unterrichtsstunden.

```
class Schüler inherits Personen
type tuple(Klasse: string,
     Klassenlehrer: Lehrer,
     Unterricht: set(tuple(UFach: string,
                    AnzahlStunden: integer))
)
```

Die Methode UStunden sieht dann sehr ähnlich zu der für Lehrer aus. Welche Ausprägung zu nehmen ist, wird zur Laufzeit des Programms vom Typ des Objekts bestimmt. Man nennt dies *late binding*.

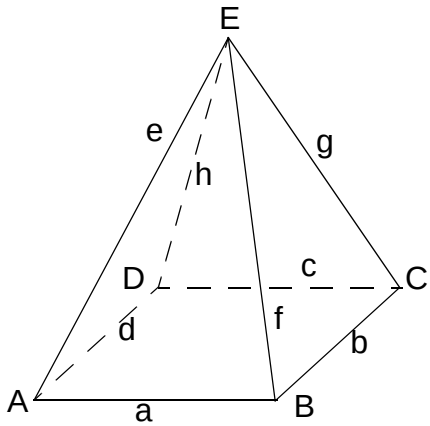
```
method body UStunden: integer in class Schüler
  {int S;
  S = 0;
  for (t in self → Unterricht)
    S = S + t.AnzahlStunden;
  return(S)
  }
```

Übung 8–13

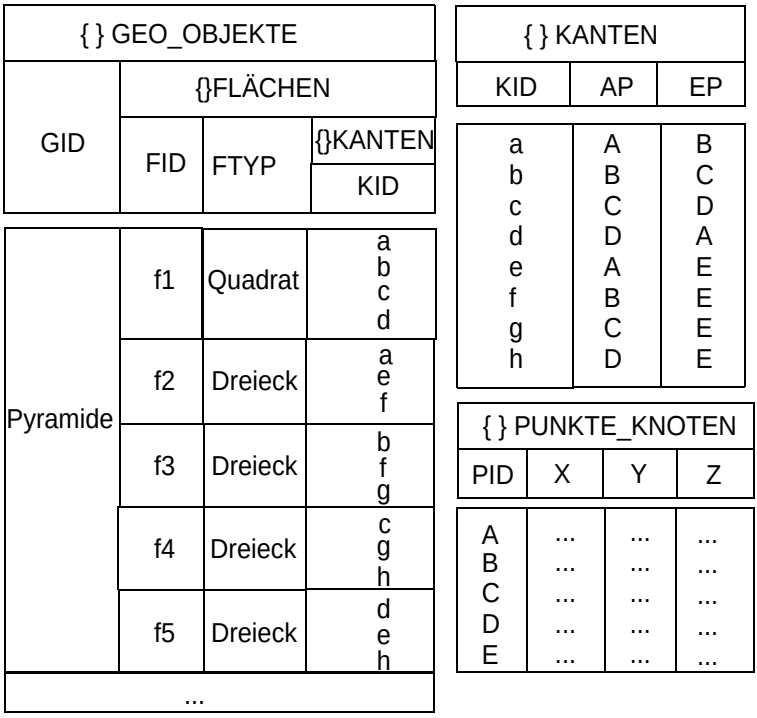
Man könnte sich eine Aufteilung der Beschaffungen in Verbrauchsmaterialien, Investitionsgüter, Bücher und Zeitschriften vorstellen. Bei den Verbrauchsmaterialien eine Trennung in Büromaterial, Werkstattbedarf, Sonstiges. Bei den Investitionsgütern könnten DV-Geräte eine Spezialisierung des Guts darstellen, diese getrennt in Rechner und Peripherie.

Übung 8–14

Der Einfachheit halber betrachten wir als Beispiel eine Pyramide, deren Geometrie durch Punkte, Kanten und Flächen bestimmt ist.



Die Geodaten haben wir recht willkürlich in eine NF^2-Tabelle für Flächen/Kanten und zwei „flache“ Tabellen für Kanten, bzw. Knoten (Punkte) aufgeteilt. Die (x, y, z)-Werte ergeben sich aus einem geeigneten Koordinatensystem.



Übung 8–15

Eine Methode grad(*v*) in einem objekt-relationalen DBMS mit obigem Datenschema würde den Grad eines Knotens *v* dadurch bestimmen, dass es die Tupel in Tabelle KANTEN zählt, deren Anfangs- oder Endpunkt den Wert *v* hat. Die entsprechende SQL-Anweisung lautet:

```
SELECT COUNT(*)
FROM KANTEN
WHERE AP = v OR EP = v
```

Alternativ könnte man z. B. Flächenberechnungen ganz in eine Programmiersprache verlegen und sich aus der Datenbank nur die Koordinatenwerte holen.

Literatur

Textbücher

[Cann99] S. J. Cannan, *lntroduction to SQL3*, in: *The SQL Files*, Array Publications. 1999.

[Catt94] R. Cattell (Hrsg.), The Object Database Standard: ODMG-93, Morgan Kaufmann, San Mateo, CA, 1994

[CP84] S. Ceri und G. Pelagatti, Distributed Databases: Principles and Systems, McGraw-Hill, New York, 1984, ISBN 0-07-Y66215-0

[Codd90] E.F.Codd, The relational model for database management: version 2, Addison-Wesley, Reading, MA, 538 S., 1990, ISBN 0-201-14192-2 (US $37,75)

[Date90] C.J. Date, An Introduction to Database Systems, Vol. 1, Addison-Wesley, Reading, MA, 854 S., 5. Auflage 1990, pb ISBN 0-201-52878-9 (US $ 38,95), hardcover ISBN 0-201-51381-1 ($46,25)

[Date85] C.J. Date, An Introduction to Database Systems, Vol. 2, Addison-Wesley, Reading, MA, 383 S., 2. Auflage 1985, ISBN 0-201-14474-3

[Date86] C. J. Date, Relational Database: Selected Writings, Addison-Wesley, Reading, MA, 497 S., 1986, ISBN 0-201-14196-5

[DD97] C. J. Date and H. Darwen, *A Guide to the SQL Standard*, Appendix F, Addison Wesley Longman, 4. Edition, 1997.

[DD98] C. J. Date and H. Darwen, *Foundation for Object/Relational Databases*, Appendix E, Addison Wesley Longman. 1998.

[DW] C. J. Date und Andrew Warden, Relational database writings (1985-1989), Addison-Wesley, MA, 528 S., 1990, ISBN 0-201-50881-8 (US $41,95)

[Diehr] George Diehr, Database Management, Scott, Foresman & Co., Glenview, IL, 412 S., 1989, ISBN 0-673-18820-5 (US $48,50)

[DU] Suzanne W. Dietrich und Susan D. Urban, An Advanced Course in Database Systems - Beyond Relational Databases, Pearson Prentice Hall, Upper Saddle River, NJ 2005

[EN89] R. Elmasri and S. B. Navathe, Fundamentals of Database Systems, Benjamin Cummings, Redwood City, CA, 802 S., 1989, ISBN 0-201-53090-2, US $39,90
5. Auflage 2007, Pearson International Edition (€ 80,-)
deutsch Ausgabe Grundstudium (3. Auflage 2005) € 29,95

[FBRB] H. Faeskorn-Woyke, B. Bertelsmeier, P. Riemer, E. Bauer, Datenbanksysteme - Theorie und Praxis mit SQL2003, Oracle und MySQL, Pearson Studium, Mün 2007, 508 S. (€ 29,95)

[Fort99] P. Fortier, *SQL3: Implementing the SQL Foundation Standard*, McGraw-Hill, 1999.

[GR93] Jim Gray and Andreas Reuter, Transaction Processing: Concepts and Techniques, Morgan Kaufmann Publishers, San Mateo, CA, 1993, ISBN 1-55860-190-2

[HR] Theo Härder und Erhard Rahm, Datenbanken - Konzepte und Techniken der Implementierung, Springer Berlin Heidelberg 1999, 582 S. (€ 39,95)

[Heuer] Andreas Heuer, Objektorientierte Datenbanken: Konzepte, Modelle, Systeme, Addison-Wesley, Bonn, 1992, ISBN 3-89319-315-4

[HHKU] M. Hein, H.H. Herrmann, G. Keereman, G. Unbescheid, Das ORACLE Handbuch - Das relationale Datenbanksystem und seine Werkzeuge, Addison-Wesley, Bonn, 640 S., 1991, ISBN 3-89319-280-8 (DM 79,-)

[HS95] Andreas Heuer und Gunter Saake, Datenbanken: Konzepte und Sprachen, Internat. Thomson Publ., Bonn, 1995, ISBN 3-929821-31-1

[KB96] Setrag Khoshafian and A. Brad Baker, MultiMedia and Imaging Databases, Morgan Kaufmann Publishers, San Francisco, CA, 1996, ISBN 1-55860-312-3

[KE] Alfons Kemper und André Eickler, Datenbanksysteme: eine Einführung, R. Oldenbourg Verlag, Mün, 1996, ISBN 3-486-23008-5
6. Auflage 2006, 672 S., € 39,80

[KW] Alfons Kemper und Martin Wimmer, Übungsbuch Datenbanksysteme (mit DVD), Oldenbourg Verlag Mün 2006 (€ 24,80)

[Koch] George Koch, ORACLE: The Complete Reference, Osborne/McGraw-Hill, Berkeley, CA., 1045 S. (Jahr ?), ISBN 0-07-881635-1

[KD88] D. Kroenke and K. Dolan (Eds), Database processing: fundamentals, design, implementation, Macmillan Publishing, New York, NY 679 S., 1988, ISBN 0-574-18642-5, (US $53,-)

[KS86] H. F. Korth and A. Silberschatz, Database System Concepts, McGraw-Hill, New York, 546 S., 1986, ISBN 0-07-044752-7

[Küsp85] Klaus Küspert, Fehlererkennung und Fehlerbehandlung in Speicherstrukturen von Datenbanksystemen, Informatik-

Fachberichte 99 (W.Brauer Ed.), Heidelberg, 1985, ISBN 3-540-15238-5

[Lausen] Georg Lausen, Datenbanken, Grundlagen und XML-Technologien, Spektrum Akademischer Verlag (Elsevier), Mün 2005

[LV96] Georg Lausen und Gottfried Vossen, Objekt-orientierte Datenbanken: Modelle und Spra, .Oldenbourg Verlag, Mün, 1996, ISBN 3-486-22370-4

[ME00] J. Melton, A. Eisenberg: *Understanding SQL and Java Together: A Guide to SQLJ, JDBC, and Related Technologies*, Morgan Kaufmann, 2000

[MS99] J. Melton, A. R. Simon: SQL:1999 - Understanding Relational Language Components, Morgan Kaufmann, 2001*Understanding the New SQL: A Complete Guide*.

[NH89] G.M. Nijssen und T.A. Halpin, Conceptual schema and relational database design: A fact oriented approach, Prentice-Hall, Inc., Englewood Cliffs, NJ, 342 S., 1989, ISBN 0-13-167263-0 (US $48,20)

[Oracle] Oracle Corp., SQL*Plus User's Guide, Version 2.0, Oracle Corporation Belmont, Cal., Part Number 3201-V2.0, Revised July 1987

[PV89] M. Papazoglou und W. Valder, Relational database management: a systems programming approach, Prentice-Hall, Englewood Cliffs, NJ 555 S., 1989, ISBN 0-13-771866-7 (US $48,20)

[SSH] Gunter Saake, Kai-Uwe Sattler und Andreas Heuer, Datenbanken, Konzepte und Spra, Mitp-Verlag; 3. Auflage 2007, 784 S., € 39,95

[Sauer] Herrmann Sauer, Relationale Datenbanken, Theorie und Praxis, Addison-Wesley, Bonn, 232 S., 1991, ISBN 3-89319-167-4 (DM 59,-)

[Türker] Can Türker, SQL:1999 & SQL:2003 - Objektrelationales SQL, SQLJ & SQL/XML, dpunkt.verlag, Heidelberg 2003(€ 29,-)

[TS05] Can Türker und Gunter Saake, Objektrelationale Datenbanken, dpunkt.verlag, Heidelberg 2006 (€ 39,-)

[Ullm88] J.D. Ullman, Principles of Database and Knowledge-Base Systems, Vol. 1, Computer Science Press, Rockville, MD, 631 S., 1988, ISBN 0-7167-8158-1

[Vossen] Gottfried Vossen, Data models, database languages and database management systems, Addison-Wesley, Reading MA, 1991, ISBN 0-201-41604-2

[Wied] Gio Wiederhold, Dateiorganisation in Datenbanken, McGraw-Hill, Maidenhead, Berkshire(UK), 404 S., 1988, ISBN 3-89028-133-8 (DM 98,-)

[Wilke] Hans D. Wilke, ORACLE - Datenbank-Management professionell: Design, Realisierung und Optimierung, Addison-Wesley, Bonn, 256 S., 2. Auflage 1991, ISBN 3-89319-325-1(DM 59,-)

[WK88] Gabrielle Wiorkowski und David Kull, DB2 Design & Development Guide, Addison-Wesley, Reading, MA, 322 S., 1988, ISBN 0-201-16949-5

[Zehn] C.A. Zehnder, Informationssysteme und Datenbanken, B.G. Teubner, Leitfäden der Angewandten Informatik, Stuttgart, 274 S., 5. Auflage 1989, ISBN 3-519-22480-1 (DM 42,-)

Artikel

[Atk89] M. Atkinson, F. Bancilhon, D. DeWitt, K.Dittrich, D. Maier, S. Zdonik, The Object-Oriented Database System Manifesto, ALTAIR TR No. 30-89, GIP ALTAIR, LeChesnay, France, Sept. 1989, also in: Deductive and Object-oriented Databases, Elsevier Science Publishers, Amsterdam, Netherlands, 1990

[AB87] M.P.Atkinson und O.P. Buneman, Types and Persistence in Database Programming Languages, ACM Computing Surveys, Vol. 19, No. 2, 1987

[AM89] M.P. Atkinson und R. Morrison, Persistent System Architectures, in: Persistent Object Systems, Proc. Third Int. Workshop, Newcastle, Australia, January 1989 (John Rosenberg and David Koch eds.), Springer-Verlag, London, 1989, pp. 73-97

[Arm74] W.W.Armstrong, Dependency Structures of Data base Relationships, Proc. 1974 IFIP Congress (1974) pp. 580-583

[AS75] ANSI/X3/SPARC (American National Standards Committee/Standard Planning and Requirements Committee) Study Group on Data Base Management Systems: Interim Report. FDT (Bulletin of ACM SIGMOD) 7, No. 2 (1975)

[Astr76] M.M.Astrahan, M.W.Blasgen, D.D.Chamberlin, K.P.Eswaran, J.N.Gray, P.P.Griffiths, W.F.King, R.A.Lorie, P.R.McJones, J.W.Mehl, G.R.Putzolu, I.L.Traiger, B.W.Wade, and V.Watson: System-R: Relational approach to database management, ACM Trans. Database Systems, Vol. 1, No. 2 (June 1976) pp. 97-137

[ANSI89] American National Standards Institute, Database language: Language Embeddings, Doc. ANSI X3.168-1989

[BCCL91] C.Batini. T.Catarci, M.F.Costabile, S.Levialdi, Visual Query Systems, Rap. 04.91, Universita degli Studi di Roma „La Sapienza“, Dipartimento di Informatico e Sistemistica, Marzo 1991

[BLN86] C. Batini, M. Lenzerini, and S.B. Navathe, A Comparative Analysis of Methodologies for Database Schema Integration, ACM Comp. Surveys, Vol. 18, No. 4, Dec. 1986, pp. 323-364

[Bee90] C. Beeri, A formal approach to object-oriented databases, Data and Knowledge Engineering, Vol. 5, No. 4, pp. 353-382, 1990

[Cha88] A.K. Chandra, Theory of Database Queries, Proc. ACM Symp. PoDS, 1988, pp. 1-9

[Chen76] P. P. S. Chen, The Entity Relationship model: Toward a unified view of data, ACM Trans. on Database Systems, 1(1):9-36

[COD71] CODASYL Data Base Task Group April 71 Report, ACM, New York.

[Codd70] E.F.Codd, A relational model of data for large shared data banks, Commun. ACM 13, 6 (June 1970), 377-387

[Codd71] E.F.Codd, A data base sublanguage founded on the relational calculus, Proc. ACM SIGFIDET Workshop on Data Description, Access and Control (San Diego, Calif.), ACM, New York, 1971, pp. 35-68

[Codd 79] E.F. Codd, Extending the Database Relational Model to Capture more Meaning, ACM TODS, Vol. 4, No. 3 (Sept. 1979), pp. 397-434

[Codd85a] E.F.Codd, Is your DBMS really relational?, Computerworld, Oct. 14, 1985

[Codd85b] E.F. Codd, Does your DBMS run by the rules? , Computerworld, Oct. 21, 1985

[Codd87] E.F. Codd, More Commentary on Missing Information in Relational Databases (Applicable and Inapplicable Information). SIGMOD RECORD 16(1), S. 42-47 (March 1987)

[Dad86] P. Dadam et al., A DBMS prototype to support extended NF2-relations: An integrated view on flat tables and hierarchies, Proc. ACM SIGMOD Conf., 376-387 (1986)

[Date82] C.J. Date, Null Values in Database Management, Proc. 2nd British National Conference on Databases, Bristol, England, July 1982, nachgedruckt in C.J.Date, Relational Database: Selected Writings, Addison Wesley, Reading, Mass., 1986

[Date84] C.J.Date, A Critique of the SQL Database Language, ACM SIGMOD Record 14, No. 3 (Nov. 1984), reprinted in [Date 86], S. 269-311

[DGK82] U. Dayal, N. Goodman, R. Katz, An Extended Relational Algebra with Control Over Duplicate Elimination, Proc. ACM Symp. PoDS, Los Angeles, CA, March 1982, pp. 117-123

[EM99] A. Eisenberg, J. Melton, *SQL:1999, formerly known as SQL3*. ACM SIGMOD Records, 28(1). März 1999.

[Gray95] Jim Gray, A Survey of Parallel Database Techniques and Systems, Tutorial Handouts, 21. VLDB, Zurich, Switzerland, Sept. 11-15, 1995

[GWHP94] G. Ganger, B. Worthington, R. Hou, and Y. Patt, Disk Arrays: High-Performance, High-Reliability Storage Subsystems, IEEE Computer, Vol. 27, No. 3 (March 1994) 30-36

[HR83] T. Härder und A. Reuter, Principles of Transaction-Oriented Database Recovery, ACM Computing Surveys, Vol. 15, No. 4, (December 1983) 287-318

[HSKun] H.S. Kunii, Graph Data Model and Its Data Language, Springer Verlag, Heidelberg, 106 S., 1990, ISBN 3-540-70058-7

[ISO87a] International Standards Organization, Database Language NDL, Doc. ISO 8907-1987(E), also ANSI X3.133-1986

[ISO87b] International Standards Organization, Database Language SQL, Doc. ISO 9075-1987(E), also ANSI X3.135-1986 - „SQL86“

[ISO89] International Standards Organization, Database Language SQL with integrity enhancement, Doc. ISO 9075-1989(E), also ANSI X3.135-1989 - „SQL89“

[ISO90] International Standards Organization, (ISO working draft) Database Language SQL2 and SQL3, internal committee document ISO/IEC JTC1/SC21 WG3 DBL SEL-3b, dated April 1990

[JTTW90] John J. Joseph, Satish M. Thatte, Craig W. Thompson, David L. Wells, Object-Oriented Databases: Design and Implementation, Proc. of the IEEE, Vol. 79, No. 1, January 1991, pp. 42-64

[JV85] M. Jarke und Yannis Vassiliou, A Framework for Choosing a Database Query Language, ACM Computing Surveys, Vol. 17, No. 3, Sept. 1985, 313-340

[KTW90] K. Küspert, J. Teuhola und L. Wegner, Design Issues and First Experiences with a Visual Database Editor for the Extended NF^2 Data Model, Proc. 23rd Hawaii Int. Conf. on System Sciences, Kona, Hawaii, 1990, B.Shriver (Ed.), IEEE CS Press (1990) 308-317

[KW07] Prabhu Shankar Kaliappan und Uwe Wloka: Investigations to the Conformance about Oracle, DB2, MS SQL Server, Sybase with respect to SQL:2003 Standard, Datenbank Spektrum 23/2007, S. 38-44

[Lufter99] J. Lufter. *Objektrelationale Datenbanksysteme*, Informatik Spektrum, 22(4), Aug. 1999

[Mattos99] N. Mattos et al., *SQL3, Objekt-Orientierung und Java als Standard: Ein Überblick über SQL3 und SQLJ,* Deutsche Informatik-Akademie, März 1999.

[MHLPS] C. Mohan, D. Haderle, B. Lindsay, H. Pirahesh and P. Schwarz, ARIES: A Transaction Recovery Method Supporting Fine-Granularity Locking and Partial Rollbacks Using Write-Ahead Logging, IBM Research Report RJ

6649R, 1990, erschienen in ACM Trans. on Database Systems, Vol 17, No. 1 (March 1992) 94-162

[RC89] Joel E. Richardson und Michael J. Carey, Implementing Persistence in E, in: Proc. Workshop Persistent Object Systems (John Rosenberg and David Koch eds.), Newcastle, Australia, 1989, Springer-Verlag, Heidelberg

[Sch77] J.W.Schmidt, Some high-level language constructs for data of type relation, ACM Trans. Database Systems, Vol. 2, No. 3 (Sept. 1977) pp. 247-261

[SH92] A.S.M. Sajeev und A. John Hurst, Programming Persistence in C, IEEE Computer, Vol. 25, No. 9, September 1992, pp. 57-66

[Shaw90] P.Shaw, Database Language Standards: Past, Present, and Future, Proc. Int. Symposium Database Systems of the 90s, A.Blaser (ed.), LNCS 466, Springer-Verlag, Heidelberg, 1990, S. 55-80

[SM93] Ernest Szeto and Viktor M. Markowitz, ERDRAW 5.3, a Graphical Editor for Extended Entity-Relationship Schemas, Reference Manual.-Technical Report LBL-PUB 3084, Lawrence Berkeley Laboratory, Berkeley CA, 1993

[SWKH] M.Stonebraker, E.Wong, P.Kreps, G.Held, The design and implementation of INGRES, ACM Trans. on Database Systems, Vol. 1, No. 3 (Sept. 1976) pp. 189-222

[St90] The Committee for Advanced DBMS Function (M. Stonebraker, L.A. Rowe, B. Lindsay, J. Cray, M. Carey, M. Brodie, P. Bernstein, D. Beech), Third Generation Database Manifesto, SIGMOD RECORD, Vol. 19, No. 3, September 1990, pp. 31-44

[TYF86] T.J. Teorey, D. Yang, J.P. Fry, A Logical Design Methodology for Relational Databases Using the Extended Entity-Relationship Model ACM Comp. Surveys, Vol. 18, No. 2, June 1986, pp. 197-222

[Todd76] S.J.P.Todd, The Peterlee Relational Test Vehicle - A system overview, IBM Syst.J. Vol. 15, No. 4, 1976, pp. 285-308

[TWPT] Jens Thamm, Lutz Wegner, Manfred Paul, Sven Thelemann, Pointer Swizzling in Non-Mapped Object Stores, Proceedings of the 7th Australasian Database Conference (ADC '96, Melbourne, Australia, 29-30 January 1996) [Australian Computer Science Communications, Vol. 18, No. 2], pp. 11-20

[Yue91] K. Yue, A More General Model for Handling Missing Information Using a 3-Valued Logic, SIGMOD RECORD 20(3), September 1991, S. 43-49

[Zloof77] M.Zloof, Query By Example: A data base language, IBM Syst. J. Vol. 16, No. 4 (1977) pp. 324-343

Index

B

C

D

E

F

L

M

N

O

P

T

U

V

W

X

Z

Ziffern