

---

# Datenbanksysteme

Modellierung, Anfragesprachen, Realisierung  
Eine theoretische Einführung mit Praktikum

Peter Bartke, Oliver Schäfer und Sebastian Herker  
FU-Berlin – WiSe 2022/2023 und SoSe 2023

---

## **Datenbanksysteme – Modellierung, Sprachen, Realisierung**

© Peter Bartke, Oliver Schäfer und Sebastian Herker  
FU Berlin – WiSe 2022/23 und SoSe 2023  
Version vom 20. Februar 2023

## Inhaltsverzeichnis

<b>1. Einführung</b>	<b>1-1</b>
1.1. Datenspeicherung im Dateisystem . . . . .	1-1
1.2. Datenbanksysteme . . . . .	1-3
1.2.1. Historische Entwicklung . . . . .	1-4
1.2.2. Die Drei-Schichten-Architektur . . . . .	1-5
1.2.3. Aufgaben eines DBMS . . . . .	1-8
1.3. Vergleich Dateisystemansatz – Datenbanksystemansatz . . . . .	1-8
1.4. Übungszettel A-1 . . . . .	1-10
<b>2. Datenbankentwurf</b>	<b>2-1</b>
2.1. Das ER-Modell . . . . .	2-2
2.1.1. Entities . . . . .	2-2
2.1.2. Relationships . . . . .	2-5
2.1.3. Weitere Konzepte . . . . .	2-7
2.1.4. Ein schneller Weg in ein ER-Modell . . . . .	2-9
2.2. Übungszettel A-2 . . . . .	2-10
2.3. Von der Anforderungsdefinition zum ER-Modell . . . . .	2-12
2.3.1. Ausführliches Beispiel . . . . .	2-14
2.3.2. Zusammenfassung . . . . .	2-16
2.4. Übungszettel A-3 . . . . .	2-19
<b>3. Datenmodelle und Datenbanksprachen</b>	<b>3-1</b>
3.1. Das relationale Datenmodell . . . . .	3-1
3.1.1. Grundbegriffe . . . . .	3-1
3.1.2. Relationenalgebra: Operationen auf Relationen . . . . .	3-4
3.1.3. Abgeleitete relationale Operationen . . . . .	3-6
3.2. Software für Relationenalgebra im Unterricht . . . . .	3-9
3.2.1. Installation und erste Schritte . . . . .	3-10
3.3. Übungszettel A-4 . . . . .	3-13
3.4. Vom ER- zum Relationenmodell . . . . .	3-16
3.4.1. Transformation am Beispiel der Miniwelt <i>Weine</i> . . . . .	3-20
3.5. Übungszettel A-5 . . . . .	3-24
3.6. Relationenalgebra . . . . .	3-25
3.6.1. Grundoperationen der Relationenalgebra . . . . .	3-26
3.6.2. Abgeleitete Operationen der Relationenalgebra . . . . .	3-26
3.6.3. Gesetze der Relationenalgebra . . . . .	3-27
3.6.4. Relationenalgebraische Ausdrücke am Beispiel . . . . .	3-28
3.7. Übungszettel A-6 . . . . .	3-33
3.8. Relationale Anfragesprachen . . . . .	3-34
3.8.1. Das Tupelkalkül . . . . .	3-34
3.8.2. Das Domänenkalkül . . . . .	3-36
3.8.3. Sichere Ausdrücke . . . . .	3-37
3.8.4. TRC und DRC in DES . . . . .	3-38
3.9. Übungszettel A-7 . . . . .	3-40
3.10. Datalog . . . . .	3-42
3.10.1. Prolog . . . . .	3-42
3.10.2. Datalog . . . . .	3-43

3.10.3. Datalog und DES . . . . .	3–46
3.11. Übungszettel A-8 . . . . .	3–49
<b>4. Funktionale Abhängigkeiten und Normalisierung</b>	<b>4–1</b>
4.1. Anomalien . . . . .	4–1
4.2. Funktionale Abhängigkeiten . . . . .	4–2
4.3. Schlüssel . . . . .	4–6
4.4. Hülle und minimale Überdeckung . . . . .	4–7
4.4.1. Hüllenbildung . . . . .	4–7
4.4.2. Minimale Überdeckung . . . . .	4–10
4.4.3. Software-gestützte Berechnung . . . . .	4–11
4.5. Übungszettel A-9 . . . . .	4–13
4.6. Normalformenlehre . . . . .	4–14
4.6.1. Erste Normalform . . . . .	4–14
4.6.2. Zweite Normalform . . . . .	4–14
4.6.3. Dritte Normalform . . . . .	4–16
4.6.4. Boyce-Codd-Normalform . . . . .	4–18
4.6.5. Verlustlose Zerlegung . . . . .	4–19
4.6.6. Zerlegung in BCNF . . . . .	4–20
4.7. Übungszettel A-10 . . . . .	4–23
<b>5. SQL</b>	<b>5–1</b>
5.1. Übersicht zum Sprachkern von SQL . . . . .	5–2
5.1.1. Datendefinition (DDL) . . . . .	5–2
5.1.2. Datenmanipulation (DML) . . . . .	5–4
5.1.3. Datenkontrolle (DCL) – die SELECT-Anweisung . . . . .	5–6
5.2. Übungszettel A-11 . . . . .	5–9
5.3. Erweiterte SQL-Konzepte . . . . .	5–13
5.3.1. WHERE-Ausdrücke . . . . .	5–13
5.3.2. Sortieren . . . . .	5–14
5.3.3. Aliase und temporäre Tabellen . . . . .	5–14
5.3.4. Sichten . . . . .	5–15
5.3.5. Gruppenfunktionen, GROUP BY . . . . .	5–16
5.3.6. HAVING-Klausel . . . . .	5–17
5.3.7. Unteranfragen . . . . .	5–18
5.4. Integritätsbedingungen . . . . .	5–21
5.4.1. Statische Integritätsbedingungen . . . . .	5–21
5.4.2. Dynamische Integritätsbedingungen . . . . .	5–24
5.5. Übungszettel A-12 . . . . .	5–26
5.6. Jenseits der Standard-Typen . . . . .	5–27
5.6.1. Aufzähltypen . . . . .	5–27
5.6.2. Zusammengesetzte Typen (Verbundtypen) . . . . .	5–28
5.6.3. Felder (ARRAY-Datentypen) . . . . .	5–28
5.6.4. Definition neuer Wertebereiche mit CREATE DOMAIN . . . . .	5–30
5.7. Übungszettel A-13 . . . . .	5–31
<b>6. Transaktionen, Serialisierbarkeit und Recovery</b>	<b>6–1</b>
6.1. Transaktionen . . . . .	6–1
6.1.1. Einfaches Transaktions-Modell . . . . .	6–2

6.1.2.	Zustandsübergänge von Transaktionen . . . . .	6–3
6.1.3.	Transaktionsmanagement in SQL . . . . .	6–4
6.1.4.	Lesende Transaktionen und Isolationsstufen . . . . .	6–6
6.2.	Schedules . . . . .	6–9
6.2.1.	Serielle Schedules . . . . .	6–10
6.2.2.	Serialisierbare Schedules . . . . .	6–11
6.2.3.	Das RW-Modell für Schedules . . . . .	6–13
6.3.	Übungszettel A-14 . . . . .	6–14
6.4.	Konfliktserialisierbarkeit . . . . .	6–16
6.4.1.	Konflikte . . . . .	6–17
6.4.2.	Präzedenzgraphen . . . . .	6–20
6.4.3.	Topologische Sortierung . . . . .	6–22
6.4.4.	Topologisches Sortieren mit Miranda . . . . .	6–24
6.5.	Übungszettel A-15 . . . . .	6–26
6.6.	Übungszettel A-16 – Querschnittsübung . . . . .	6–28
6.7.	2-Phasen-Sperrprotokoll (2PL) . . . . .	6–31
6.7.1.	Sperrbasierter Scheduler . . . . .	6–31
6.7.2.	Verklemmungen . . . . .	6–34
6.7.3.	Erweiterte Konzepte im 2PL . . . . .	6–36
6.8.	Übungszettel A-17 . . . . .	6–36
6.9.	Logging und Recovery . . . . .	6–37
6.9.1.	Fehlerklassifikation . . . . .	6–37
6.9.2.	Speicherhierarchie und Einbringstrategien . . . . .	6–39
6.9.3.	UNDO-Logging . . . . .	6–41
6.9.4.	REDO-Logging . . . . .	6–44
6.9.5.	UNDO/REDO-Logging . . . . .	6–45
6.9.6.	Sicherungspunkte . . . . .	6–46
6.10.	Übungszettel A-18 . . . . .	6–49
<b>7.</b>	<b>Anwendungsprogramme</b>	<b>7–1</b>
7.1.	postgreSQL und GO . . . . .	7–1
7.1.1.	Installation . . . . .	7–1
7.1.2.	Erste Schritte . . . . .	7–2
7.1.3.	Arbeiten mit Ergebnismengen . . . . .	7–4
7.2.	mySQL und GO . . . . .	7–4
7.3.	postgreSQL und PHP . . . . .	7–6
7.3.1.	Installation unter Linux . . . . .	7–6
7.3.2.	Installation unter Windows . . . . .	7–10
7.3.3.	Konfiguration für den Fernzugriff . . . . .	7–10
7.4.	Grundlagen zu HTML und CSS . . . . .	7–12
7.4.1.	Ein Minimalbeispiel . . . . .	7–14
7.4.2.	Tabellen . . . . .	7–15
7.4.3.	Gestaltung mit CSS . . . . .	7–15
7.5.	Grundlagen zu PHP . . . . .	7–17
7.5.1.	Ein Minimalbeispiel . . . . .	7–17
7.5.2.	Datenbankzugriff mit PHP . . . . .	7–18
7.5.3.	Formulare und Formularfelder . . . . .	7–21

<b>A. Planung</b>	<b>A–1</b>
-------------------	------------

<b>B. Literaturverzeichnis</b>	<b>B-1</b>
<b>C. psql – das PostgreSQL-Terminal</b>	<b>C-1</b>
<b>D. Miniwelten</b>	<b>D-1</b>
D.1. Uni-Miniwelt . . . . .	D-1
D.2. Bibliotheks-Miniwelt . . . . .	D-2
<b>E. Zugriff mit Go auf SQL-Datenbanken</b>	<b>E-1</b>
E.1. Beschreibung der Schnittstelle . . . . .	E-1
E.2. postgresQL-Spezifika . . . . .	E-2
E.3. mySQL-Spezifika . . . . .	E-4
<b>F. Unterrichtsbezogenes Datenbankpraktikum</b>	<b>F-1</b>
F.1. Einordnung . . . . .	F-1
F.2. Teilaufgaben . . . . .	F-2
F.3. Schriftliche Ausarbeitung . . . . .	F-4
F.4. Bewertung des UDBP . . . . .	F-9
F.5. Praktische Hinweise zum UDBP . . . . .	F-11

## 1. Einführung

Datenbanksysteme sind aus der heutigen Zeit kaum noch wegzudenken. Sie spielen in Unternehmen, Behörden und anderen Organisationen eine zentrale Rolle. Dabei geht es nicht nur um das Vorhalten von Daten zur Prozesssteuerung, sondern auch um die Analyse von Abläufen mit dem Zweck der Optimierung selbiger. Als Beispiel denke man an den Onlinehandel, Banken und Versicherungen, Buchungssysteme wie bei Flügen oder Konzerttickets, Personalverwaltung in Unternehmen oder die Verwaltung von Studenten an Universitäten. Auch in Schulen weicht eine Schülerdatenverwaltung mit Hilfe einer Tabellenkalkulation oder gar handschriftlich(!) einer zentralen Datenbankverwaltung mit definierten Zugriffsrechten für jede Benutzergruppe auf einem einheitlichen Datenbestand. Hinzu kommt die mittlerweile allgegenwärtige Vernetzung von Systemen<sup>1</sup>, die zu einer durchdachten Datenhaltung zwingt, die Redundanz vermeidet, Performanz ermöglicht und gleichzeitig Verlässlichkeit liefert. Dabei ist die steigende Informationsmenge<sup>2</sup> ein nicht zu vernachlässigendes Problem, dem sich heutzutage Datenbanksystementwicklungen stellen müssen. Während früher noch viele Informationen auf Papier niedergelegt wurden, findet man heute die meisten Informationen zumindest auch in digitaler Form vor, oft sogar nur noch so. Symbolträchtig soll hier der 'Brockhaus' genannt werden, der seine 22. Auflage in gedruckter Form nicht mehr erlebt hat; die 21. Auflage aus dem Jahr 2007 war die letzte gedruckte Version, der Verlag stellte die Produktion ein und wendete sich ganz der digitalen Welt zu.<sup>3</sup>

### 1.1. Datenspeicherung im Dateisystem

Daten zu speichern ist eine Domäne des Dateisystems eines jeden Betriebssystems. So ist in vielen Fällen der naive Ansatz erste Wahl, Daten(-sätze) in Form einfacher Tabellen abzulegen. Das Beispiel in Abbildung 1 möge dies illustrieren.

Sollten nun mehrere Programme/Programmkomponenten mit diesen Tabellen arbeiten, so kann ein – ebenfalls naiver – Ansatz darin bestehen, diese Dateien für den jeweils eigenen Gebrauch zu kopieren (s. Abb. 2). In nicht-vernetzten Systemen kann dies z. B. eine Kopie durch den Benutzer mittels USB-Stick von einem Rechner auf den anderen bedeuten. Innerhalb eines Betriebssystems können dies z. B. Kopien durch die Programme in die jeweilige Library bedeuten.

Eine im laufenden Betriebssystem dagegen typische Situation ist der versuchte Zugriff von unterschiedlichen Programmen/Programmkomponenten auf gleiche Dateien zu gleicher Zeit (Abb. 3). Diese Art des Dateizugriffs – zumindest im Fall mehrerer schreibender Prozesse – wird von allen Betriebssystemen konsequenterweise unterbunden; typische Fehlermeldungen wie *'Die Datei wird bereits von einem anderen Prozess verwendet.'* sind allseits bekannt.<sup>4</sup>

<sup>1</sup>An Berliner Schulen z. B. sowohl innerhalb der Schule, als auch durch die 2017 eingeführte zentrale "Berliner Lehrkräfte-Unterrichts-Schul-Datenbank"(LUSD).

<sup>2</sup>Schätzungen zufolge verdoppelt sich die Informationsmenge derzeit immer schneller. Galt 2009 noch, dass sich ca. alle 5 Jahre die Menge an Informationen in der Welt etwa verdoppelt ([KE09, S. 19]), war 2013 bereits in einer viel zitierten Studie von etwa 2-3 Jahren die Rede. <https://www.welt.de/wirtschaft/webwelt/article118099520/Datenvolumen-verdoppelt-sich-alle-zwei-Jahre.html>. Das 'Internet der Dinge' und die zunehmende Digitalisierung sog. Schwellen- und Entwicklungsländer dürften dies noch weiter vorantreiben. Die zunehmende Digitalisierung der Schulen und die Verwendung von Online-Plattformen für den Unterricht tragen ebenfalls ihren Teil bei.

<sup>3</sup><http://www.tagesspiegel.de/kultur/der-digitale-brockhaus-entsteht-access-for-all/10373022.html>

<sup>4</sup>Die Problematik ist unter dem Namen *Leser-Schreiber-Problem* aus der NSP bekannt.

Name	Adresse	PLZ	Telefon	...
Hans Müller	Allee 45, Musterstadt	12345	04321-12345	...
Lisa Meier	Weg 23, Musterdorf	21345	03241-21453	...

...	Konto	Bank	Datum	Artikel	Preis
...	213456015	1007000	20090211	Netbook	400
...	435260022	10010010	20081213	Notebook	1000

Abbildung 1: Beispiel für Daten einer Kundenverwaltung, abgespeichert als Tabelle in einer Datei

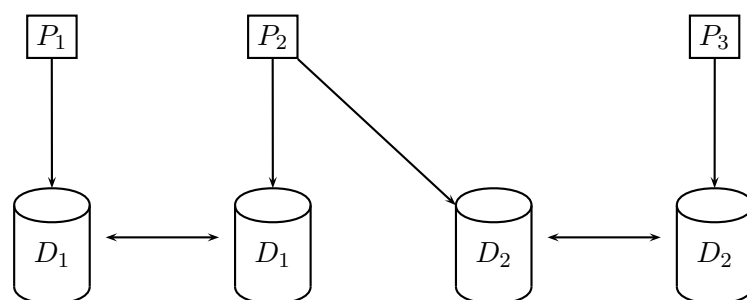


Abbildung 2: Ungünstige Architektur I: Programme bzw. Programmkomponenten greifen auf kopierte Dateien zu

Es ergeben sich durch diese Ansätze eine große Reihe von Schwierigkeiten, die zum Teil auf konzeptioneller, zum Teil auf praktischer Ebene liegen und sich nicht ohne Weiteres durch ausgeklügelte Implementierung lösen lassen:

- (a) Änderungen am Dateiformat wirken sich auf alle benutzenden Programm(teil)e aus
- (b) Direkte Abbildung der physischen Dateistruktur in der Programmlogik
- (c) Datenredundanz: gleiche Information in verschiedenen Dateien.  
Folge:
  - Platzverbrauch
  - „Vergessen“ synchron notwendiger Änderungen
- (d) Probleme mit Mehrbenutzerbetrieb:
  - Inkonsistenz
  - Ineffizienz
- (e) Eingeschränkte Portabilität der Daten

Zur Vermeidung dieser Probleme wird eine zusätzliche Schicht, eine *Abstraktionsschicht* eingebracht, die den Datenbestand absichert und organisiert. Sie ermöglicht es, die Daten in einem *gemeinsamen Datenbestand* zu integrieren und dabei ...



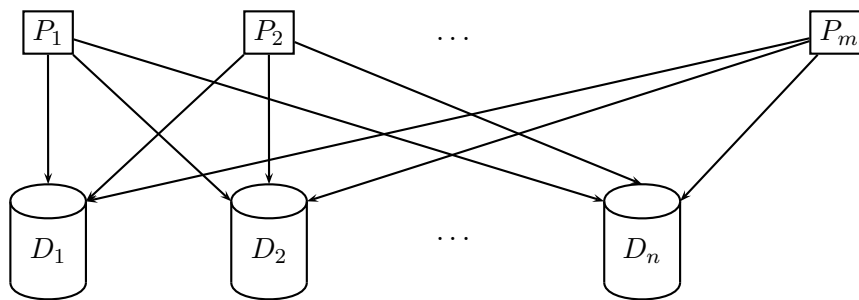


Abbildung 3: Ungünstige Architektur II:  $m$  Programme bzw. Programmkomponenten benutzen  $n$  Dateien.

- ... *Datenunabhängigkeit* zu ermöglichen,
- ... *Datenschutz* durch Autorisierungsmaßnahmen zu gewährleisten,
- ... *Datenintegrität* durch korrekte Mehrbenutzersynchronisation sicher zu stellen und
- ... *Datenhandhabung* effizient zu gestalten.

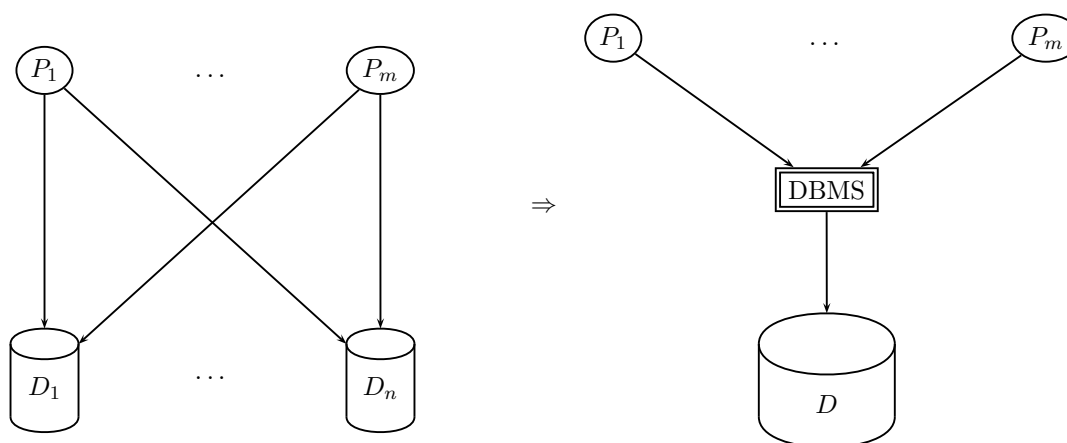


Abbildung 4: Dateisystem: Problemlösung durch Datenabstraktion

## 1.2. Datenbanksysteme

Die heilsbringende Abstraktionsschicht wird landläufig als Datenbank oder Datenbanksystem bezeichnet. Nur wenige Bücher pflegen hier eine klare, begriffliche Unterscheidung der Begriffe.<sup>5</sup> Um Klarheit in den oft wenig unterschiedenen Begriffen zu schaffen, hier eine kurze Übersicht zu Begriffen und ihren Abkürzungen.

### Datenbank, DB

Eine Datenbank ist eine zusammengehörige Menge von Daten, versehen mit einer definierten (Daten-)Struktur. In ihr gibt es Daten, die sich zeitlich verändern und

<sup>5</sup>Zu den rühmlichen Ausnahmen gehört das Standardwerk [EN09], aus dem die in diesem Skript verwendeten Begriffsdefinitionen übernommen wurden.

andere, die zeitlich stabil bleiben.<sup>6</sup> Eine Datenbank repräsentiert einen Lebenswelt-Ausschnitt<sup>7</sup> mit i. d. R. reduzierter Komplexität und wird von einer Software verwaltet. Es handelt sich insgesamt 'nur' um das Datenmaterial bzw. die Datenbasis.

### Datenbankmanagementsystem, DBMS

Ein Datenbankmanagementsystem (auch Datenbankverwaltungssystem) ist ein komplexes Softwaresystem mit vielen Teilkomponenten, die das Lesen, Suchen, Erzeugen, Ändern und Löschen von Daten aus einer oder mehreren Datenbanken regeln. Es enthält darüber hinaus Werkzeuge zur Definition der Datenbank (um z. B. die Struktur festzulegen) sowie Komponenten, die den sicheren Betrieb gewährleisten (Concurrency, Persistenz, ...).

### Datenbanksystem, DBS

Als Datenbanksystem bezeichnet man den Verbund aus Datenbank und Datenbankmanagementsystem sowie (manchmal als Teil eines DBMS aufgefasster) Softwarekomponenten, die eine Schnittstelle zwischen Anwender und DBMS bilden, in der Regel als GUI realisiert. Man denke z. B. an den Bankangestellten, der über ein vorbereitetes Formular den Kontostand eines Kontos erfragen kann.

DBS=DB+DBMS

### 1.2.1. Historische Entwicklung

Sowohl die Datenbankmanagementsysteme als auch die Modelle zur Repräsentation der gespeicherten Daten haben sich in den vergangenen 60 Jahren immer wieder verändert. Trotzdem gibt es Konstanten in der Entwicklung wie z. B. der relationale Ansatz. Die Entwicklung hat sich einerseits auf Seiten der DBMS und andererseits auf Seiten der Datenmodelle vollzogen.

### Historische Entwicklung der Systeme

Auch die Entwicklung der DBMS hat einmal klein angefangen. Zu Beginn wurden die Probleme einer dateibasierten Datenverwaltung nicht so ernst genommen, was vielleicht auch daran lag, dass man keine Alternative sah.

- Erste Dateisysteme wurden in den 50er Jahren entworfen.
- Erste Datenbanksysteme für große Applikationen entstanden in den 60er Jahren, sie waren dateibasiert, ihre Implementierung bestimmte die Anwendungsprogrammierung übergeordneter Zugriffs-Komponenten.
- Hierarchische DBMS: IBM lieferte IMS (Information Management System) mit Data Language/I (DL/I) in den 60er Jahren, was z. B. zur Stücklistenverwaltung des Apollo-Mondprogramms verwendet wurde und noch heute von IBM weiterentwickelt wird.
- Netzwerk DBMS: UDS Siemens (Universelles Datenbank-System) und DMS Univac waren in den 70er Jahren die großen Schlager. Auf der Hannover-Messe 1976 wurde 'Systems UDS' als große Neuerung beworben.<sup>8</sup>

<sup>6</sup>In der Datenbank werden zuerst einmal keine Prozesse auf den Daten (Methoden) gespeichert. Dies ist objektorientierten Datenbanksystemen vorbehalten.

<sup>7</sup>Das beinhaltet natürlich auch fiktive Lebenswelten, wie sie z. B. für Computerspiele generiert werden.

<sup>8</sup><http://www.computerwoche.de/a/golem-im-code-asy1,1198387>

- Erste relationale DBMSe kamen in den 70er Jahren groß raus: System R von IBM benutzte SEQUEL (=Structured English Query Language) und war der Vorläufer des relationalen Datenbanksystems DB2.
- OODBMS: ORION  $O_2$  (80er Jahre) spielen heute noch immer keine große Rolle
- Deduktionssysteme basieren auf der Arbeit mit Prädikaten, die ausgewertet werden. ConceptBase (frühe 90er Jahre) als Pionier basiert auf Datalog als Anfragesprache und ist seit 2009 freie Software.<sup>9</sup>
- Objektrelationale Systeme: DB2, Oracle, Postgres etc. entstanden in den 90er Jahren
- Weiterentwicklungen, integriert in große DBMSe: XML, Semantic Web, OLAP (Data Mining)

## Historische Entwicklung der Datenmodelle

Parallel zu den DBMSe haben sich auch die Datenmodelle gewandelt. Getreu dem Motto 'Objects first' stehen OODBMS derzeit ganz hoch im Kurs.

- Hierarchisches Datenmodell: Navigieren in Baumstrukturen IBM späte 60er
- Netzwerkmodell: Navigieren in Graphen CODASYL 1971
- Relationenmodell: Edgar Frank Codd 1970
- Entity-Relationship-Modell: Peter Chen 1976
- Deduktives Modell: Prädikatenlogik als Datenmodell ca. 1980
- Objektorientierte Datenmodelle: 80er Jahre, Object Management Group OMG

Aktuell gibt es eine Reihe von Datenbankmanagementsystemen, von denen viele aber eher unbekannt sind, zumindest für Endanwender. Das liegt unter anderem daran, dass man über Anwendungssoftware auf den Datenbestand eines Datenbanksystems zugreift und man die konkrete Datenbanksoftware gar nicht zu sehen bekommt (s. Abb. 5).

### 1.2.2. Die Drei-Schichten-Architektur

DBMSe stellen dem Benutzer Daten zur Verfügung, auf die dieser schnell und verlässlich, vor allem aber implementierungsunabhängig zugreifen kann. Um das zu gewährleisten, wurde im Jahr 1975 die sogenannte Drei-Schichten-Architektur (auch ANSI-SPARK-Architektur) entwickelt. Sie beschreibt die grundlegende Trennung verschiedener Beschreibungsebenen eines DBS.<sup>10</sup> Es sind dies ...

DB-Schemata

#### ...die externe Ebene/das externe Schema

Die externe Ebene (externe Schicht) beschreibt die Sicht des Endanwenders auf die Daten (Formulare, Masken, ...). Korrespondierend damit sind sogenannte Benutzersichten (VIEWS im SQL-Jargon).

#### ...die konzeptuelle Ebene/das konzeptuelle Schema

Auf der konzeptuellen (oft auch *konzeptionellen*) Ebene wird beschrieben, welche Daten in der Datenbank gespeichert werden und wie sie logisch voneinander abhängen. Hier

<sup>9</sup>Die Auswertung von Prädikaten auf einer (großen) Variablenmenge gehört in die Klasse NP (siehe Vorlesung zur theoretischen Informatik).

<sup>10</sup>Eine gute Darstellung hierzu findet man in [Vos08, S. 26 ff].

Name	Hersteller	Kurzcharakteristik/URL
Access	Microsoft	Einbenutzer-DBS <a href="http://www.microsoft.com/de-de/microsoft-365/access">www.microsoft.com/de-de/microsoft-365/access</a>
Adabas	Software AG	NF <sup>2</sup> -DBS, Großsysteme <a href="http://www.softwareag.com/de/products">www.softwareag.com/de/products</a>
BerkeleyDB	Oracle	embedded DBS, Leichtgewicht <a href="http://www.oracle.com/database/berkeley-db/">http://www.oracle.com/database/berkeley-db/</a>
ConceptBase	RWTH Aachen	Deduktives DBS, Abfragen in Datalog <a href="http://www-i5.informatik.rwth-aachen.de/CBdoc/">http://www-i5.informatik.rwth-aachen.de/CBdoc/</a>
Db2	IBM	RelDBMS, abgeleitet von System R <a href="http://www.ibm.com/software/data/db2/">http://www.ibm.com/software/data/db2/</a>
Derby	Apache, frei	leichtes RelDBMS, reines Java, im JDK <a href="http://db.apache.org/derby/">db.apache.org/derby/</a>
Filemaker	Claris	Einbenutzer-DBS, MacOS/iOS <a href="http://www.claris.com/de/filemaker/">http://www.claris.com/de/filemaker/</a>
Firebird	Borland, frei	ObjRelDBMS, künftig in LibreOffice <a href="http://www.firebirdsql.org">http://www.firebirdsql.org</a>
HSQLDB	Open Source	RelDBS, reines Java, Standard in LibreOffice <a href="http://www.hsqldb.org">http://www.hsqldb.org</a>
Informix	IBM, HCL	RelDBMS, OLTP <a href="http://www.ibm.com/products/informix">www.ibm.com/products/informix</a>
Ingres	CA, Open Source	RelDBMS <a href="http://www.ingres.com">www.ingres.com</a>
MariaDB	Frei	Fork aus MySQL <a href="http://www.mariadb.org">www.mariadb.org</a>
MaxDB	SAP	ehem. Projekt von TU Berlin und Nixdorf <a href="https://maxdb.sap.com/">https://maxdb.sap.com/</a>
Microsoft SQL Server	Microsoft	RelDBMS <a href="http://www.microsoft.com">www.microsoft.com</a>
MongoDB	MongoDB Inc.	noSQL, für unstrukturierte Datenmengen <a href="http://www.mongodb.com">www.mongodb.com</a>
MySQL	Oracle	RelDBMS + MyISAM, InnoDB <a href="http://www.mysql.com">http://www.mysql.com</a>
Oracle	Oracle	Aus System R entstanden, Marktführer <a href="http://www.oracle.com">http://www.oracle.com</a>
PostgreSQL	Frei	Objektrelationales DBMS <a href="http://www.postgresql.org">http://www.postgresql.org</a>
SQLite	Frei	embedded RelDBS, DB in nur einer Datei <a href="http://www.sqlite.org">www.sqlite.org</a>
Sybase	Sybase (SAP)	Großes RelDBMS <a href="http://www.sybase.com">www.sybase.com</a>

wird auf möglichst vollständige und redundanzfreie Darstellung aller Daten (Normalisierung<sup>11</sup>) Wert gelegt. Das konzeptuelle Schema ist idealerweise hard- und software-unabhängig. Die konzeptuelle Ebene korrespondiert mit der Sicht des Datenbank-Designers auf das DBMS, z. T. auch die des Datenbank-Administrators.

### ... die interne Ebene/das interne Schema

Auf interner (auch physischer) Ebene wird beschrieben, wie die Daten physikalisch im Rechner (auf der Platte) gespeichert werden. Hier spielt Effizienz eine wichtige Rolle, zu deren Gunsten oft auf Redundanz – zumindest auf dieser Ebene – verzichtet wird. Beispielsweise werden Informationen auch in Form von Indexlisten (doppelt) gespeichert, um den Zugriff auf die Daten zu beschleunigen. Diese Sicht ist die des Implementierers, z. T. auch die des Datenbank-Administrators auf das DBMS.

In der Regel sind in einem DBMS die drei Ebenen nicht konsequent voneinander getrennt vorzufinden.

Im Zusammenhang mit der Drei-Schichten-Architektur gibt es auch unterschiedliche Personengruppen, die einen unterschiedlichen Blick auf ein DBS haben und an verschiedenen Stellen auf das Programmsystem/die Daten zugreifen müssen.

Akteure im  
DB-Kontext

#### (a) DBS-Verwalter: Datenbankadministrator DBA

- Zugriffsrechte
- ggf. Software/Hardware-Ressourcen
- Sicherheit
- Performanz (DB-Tuning)

#### (b) DBS-Designer

- konzeptueller Entwurf von Weltausschnitten
- Kommunikation mit Fachgebietsexperten für Designentscheidungen
- Entwicklung von Datensichten
- Planung von Data Mining

#### (c) Endbenutzer

- ephemere Nutzer mit Standardtransaktionen
- Professionelle Endbenutzer mit Spezialanforderungen
- erwarten funktionierendes, konsistentes, gut handhabbares, performantes Datenbanksystem

#### (d) Systemanalytiker

- Bedarfsermittlung
- Systemauswahl (auch Standardsoftware für Anwendungsschicht)
- ökonomische und technische Analyse der Sollkonzeption
- Definition von Benutzerschnittstellen, Einführung neuer Systeme

#### (e) Anwendungsprogrammierer

- Realisierung von Sichten für Endbenutzer
- Programmierung von Schnittstellen zum DBMS
- Entwicklung von Programmen zum automatisierten Datenaustausch von DBMS und externen Komponenten

---

<sup>11</sup>S. Kap. 4.6

### 1.2.3. Aufgaben eines DBMS

Ein Datenbank-Managementsystem sorgt für den sicheren, effizienten und implementierungsunabhängigen Zugriff auf eine (große) Menge Daten. Im Detail hat ein DBMS folgende Dinge zu tun:<sup>12</sup>

- *Integration* durch kontrollierte Beschreibung der Gesamtstruktur der Daten  $\leadsto$  nichtredundante Speicherung mit *Katalog*
- Bereitstellung aller Operationen zum *Lesen, Aufsuchen, Einfügen, Ändern, Löschen*
- Verschiedene *Sichten* auf die Daten
- Überwachung der *Integrität* der Daten (Konsistenz)
- Datenschutz durch *Authentifizierung*
- Abwicklung von Operationsfolgen als Paket (*Transaktionen*)
- *Mehrbenutzerbetrieb* ohne Konflikte
- *Datensicherheit* bei Systemfehlern

### 1.3. Vergleich Dateisystemansatz – Datenbanksystemansatz

Der DBS-Ansatz bietet viele Vorteile gegenüber einem dateibasierten Zugriff auf Daten. Man erkennt dies an den typischen Merkmalen eines DBS-Ansatzes:

- Die Anwender sehen keine Dateien mehr und können nicht direkt auf sie zugreifen
- Jeder Anwender sieht die selben Daten: keine Datenredundanz
- Strukturbeschreibung der Daten liegt im Systemkatalog (Metadaten), nicht im Anwendungsprogramm.
- Programm- und Datenunabhängigkeit: Effekt der Hinzunahme eines Attributs ist minimal.
- Verwaltung abstrakter Datentypen (ADTs) in objektrelationalen DBS  $\Rightarrow$  Datenisolation und Datenabstraktion.
- Information hiding: Keine Repräsentation von Datensatzpositionen in Dateien im Anwenderprogramm.
- Definition von Sichten auf die Daten.
- Mehrbenutzerfähigkeit mit kontrollierter Nebenläufigkeitssteuerung.
- OLTP – Online Transaction Processing (Echtzeit-Transaktionsverarbeitung)

Die Liste der Merkmale zeigt deutlich, dass die Datenorganisation in einem DBS eine Reihe von Vorteilen aufweist, die diesen Ansatz einem dateiorientierten überlegen scheinen:

- Kontrolle redundanter Datenhaltung und damit verbundener Inkonsistenzen.
- Feingranulierte Zugriffskontrolle, Datenschutz
- Implementierung von deklarativen Regeln auf dem Datenbestand.
- Integritätsbedingungen (Prädikate)
- Recovery: Wiederherstellen eines Datenbestandes
- Portierbarkeit
- Kurze Entwicklungszeit von Applikationen

Dennoch kann es Gelegenheiten geben, in denen der Einsatz regulärer Dateien einem DBS-Einsatz vorzuziehen ist. Sind beispielsweise die Daten keinen oder nur geringen zeitlichen Änderungen unterworfen, werden Echtzeitforderungen an die Datenverarbeitung gestellt oder

<sup>12</sup>In [SSH10] findet man eine wesentlich feingranuliertere Auflistung der Aufgaben eines DBMS.

greifen nicht mehrere Benutzer auf die Daten zu, ist die Verwendung eines DBMS u. U. nicht anzuraten, der finanzielle und personelle Aufwand dafür mglw. zu groß.

## Literatur zu Kapitel 1

- [EN09] ELMASRI, Ramez A. ; NAVATHE, Shamkant B.: *Grundlagen von Datenbanksystemen*. 3. Auflage. München : Pearson-Studium, 2009. – ISBN 978-3-8689-4012-1
- [KE09] KEMPER, Alfons ; EICKLER, André: *Datenbanksysteme: Eine Einführung*. 7. Auflage. München : Oldenbourg Wissenschaftsverlag, 2009. – ISBN 978-3-486-59018-0
- [SSH10] SAAKE, Gunter ; SATTLER, Kai-Uwe ; HEUER, Andreas: *Datenbanken: Konzepte und Sprachen*. 4. Auflage. Heidelberg : mitp, 2010. – ISBN 978-3-8266-9057-0
- [Vos08] VOSSEN, Gottfried: *Datenmodelle, Datenbanksprachen und Datenbankmanagementsysteme*. 5. Auflage. München : Oldenbourg, 2008. – ISBN 978-3-4862-7574-2

## 1.4. Übungszettel A-1

### Aufgabe 1.1.

- (a) *Erläutern Sie die Begriffe physische und logische Datenunabhängigkeit und begründen Sie ihre Bedeutung für die Konstruktion von DBMS.*
- (b) *Welche Nachteile haben Dateisysteme gegenüber Datenbanksystemen?*
- (c) *Welche Nachteile haben Datenbanksysteme gegenüber Dateisystemen?*

**Aufgabe 1.2.** *Flugbuchungssysteme verwalten u. a. den Reservierungsstatus der einzelnen Flüge. Konstruieren Sie eine Situation, in der ein Platz (unbeabsichtigt) mehrfach vergeben wird. Überlegen Sie, von welchen Annahmen über den Verarbeitungsvorgang Platz reservieren Sie dabei ausgehen. (Eine Überbuchungspraxis von Fluggesellschaften soll hier außer acht gelassen werden.)*

**Aufgabe 1.3.** *Gegeben sei ein stark vereinfachtes Modell eines Betriebes, der in Abteilungen gegliedert ist. Diese Abteilungen führen Projekte durch. Die Mitarbeiter gehören immer genau einer Abteilung an. Sie können an einem oder mehreren Projekten mitarbeiten, müssen aber nicht.*

*Geben Sie an, in welchem Schema **primär** Änderungen vorgenommen werden müssen (externes, internes, konzeptionelles, evtl. gar keines), wenn die im folgenden aufgeführten Änderungen vorgenommen werden (knappe Begründung):*

- (a) *Das Gehalt der Mitarbeiter darf in den Projektübersichten nicht mehr erscheinen.*
- (b) *In der Gehaltsliste soll künftig der Name, nicht mehr die Personalnummer an erster Stelle stehen.*
- (c) *Der Chef des Betriebes, der keiner Abteilung angehört, soll künftig auch in der Personalliste enthalten sein.*
- (d) *Das Projekt  $p_1$  wird in die formal neuen Projekte  $p_2$  und  $p_3$  aufgeteilt.*
- (e) *In Zukunft soll zwischen Forschungs- und Entwicklungsprojekten unterschieden werden.*
- (f) *Leistung soll belohnt werden: Jede(r) Angestellte erhält bei der nächsten Gehaltsabrechnung automatisch zusätzlich 50,- Euro Bonus, wenn er (sie) an mindestens einem Projekt mitarbeitet.*
- (g) *Es geht nicht mehr ohne Projektleiter. Künftig soll es für jedes Projekt eine(n) verantwortliche(n) Mitarbeiter(in) geben.*
- (h) *Die Projektübersicht soll künftig auch die beteiligte Mitarbeiterzahl enthalten und nach dieser Zahl absteigend geordnet sein.*
- (i) *Zur Erhöhung der Betriebssicherheit wird eine zusätzliche Archivkopie der Daten gehalten und zusätzlich die in einem gewissen Zeitraum anfallenden ändernden Operationen.*
- (j) *Es wird eine neue Abteilung eingerichtet.*



**Aufgabe 1.4.** Die Pädagogische Koordinatorin einer Schule pflegt drei Dateien, eine *Schueler*-Datei, eine *Kurswahlen*-Datei und eine *Kurs*-Datei. Ein (Unter-)Programm soll diejenigen Kurse ermitteln, die ggf. geteilt werden sollten, also z. B. mehr als 20 Wahlen aufweisen. Konkret sei folgende Auswertung gefordert: Zu erstellen ist eine Liste aller Kurse, die mehr als 20 Kurswahlen aufweisen.

- (a) Benennen Sie die Daten(-typen), die in den drei Dateien vermutlich aufgehoben werden.
- (b) Formulieren Sie in Pseudocode einen Algorithmus zur Lösung dieser Aufgabe.
- (c) Überlegen Sie, welche Änderungen dieses Auswertungsprogramm erfahren muss, wenn die Kurswahldatei fehlt und die Kurswahlen direkt in der Kursdatei durch Angabe der Anzahl der wählenden Schüler vermerkt sind. Welchen Nachteil hat diese Änderung?

**Aufgabe 1.5.** In einem Unternehmen existiert eine Datei *Mitarbeiter* mit den Angaben: Name, Adresse, Alter, Position, Betriebszugehörigkeit. Aus Datenschutzgründen wird die Information über die Gehälter in einer separaten Datei *Gehälter* gespeichert, die nicht jedem Nutzer zugänglich ist. Sie ist folgendermaßen aufgebaut: Name, Alter, Gehalt. Der Geschäftsführer des Unternehmens wünscht nun, dass

- (a) zukünftig die 10 Mitarbeiter mit der längsten Betriebszugehörigkeit ein Geburtstagspräsident bekommen,
- (b) Mitarbeiter, deren Gehalt die Beitragsbemessungsgrenze übersteigt, darüber schriftlich (Brief an Privatadresse) informiert werden.

Geben Sie in Pseudocode Algorithmen zur Realisierung dieser beiden Zusatzanforderungen an.

Nach Fertigstellung dieser Programme fällt dem Geschäftsführer ein, dass für eine sozialgerechte Gehaltsabrechnung zusätzlich Familienstand und Anzahl der Kinder gespeichert werden sollen.

Welche Lösungsmöglichkeiten gibt es dafür? Welche Probleme ergeben sich mit den jeweiligen Lösungen?

**Aufgabe 1.6.** *In einer 'Frischemarkt'-Kette wird die Verwaltung mit einem großen DBMS erledigt. Unter anderem sind folgende Aufgaben zu erledigen:*

- (a) Aktualisieren der Anzahl der Bestandsartikel nach einer erfolgten Lieferung.*
- (b) Löschen des Accounts eines Mitarbeiters, der gerade das Unternehmen verlassen hat.*
- (c) Erstellen eines automatischen Berichts zum Bestandswert der Artikel am Monatsende.*
- (d) Telefonauskunft nach einem Sortimentsartikel.*
- (e) Neuaufnahme der Zusatzinformation 'Verfallsdatum' ins System.*
- (f) Umstellung der Sicherung aller Daten von wöchentlich Sonntag auf täglich nachts.*
- (g) Abfrage aller Artikel einer Filiale, in der der Bestand unter 5 gesunken ist.*

*Ordnen Sie die Tätigkeiten – mit stichwortartiger Begründung – der richtigen Benutzergruppe (s. Skript 1.2.2) im DBMS-Umfeld zu.*



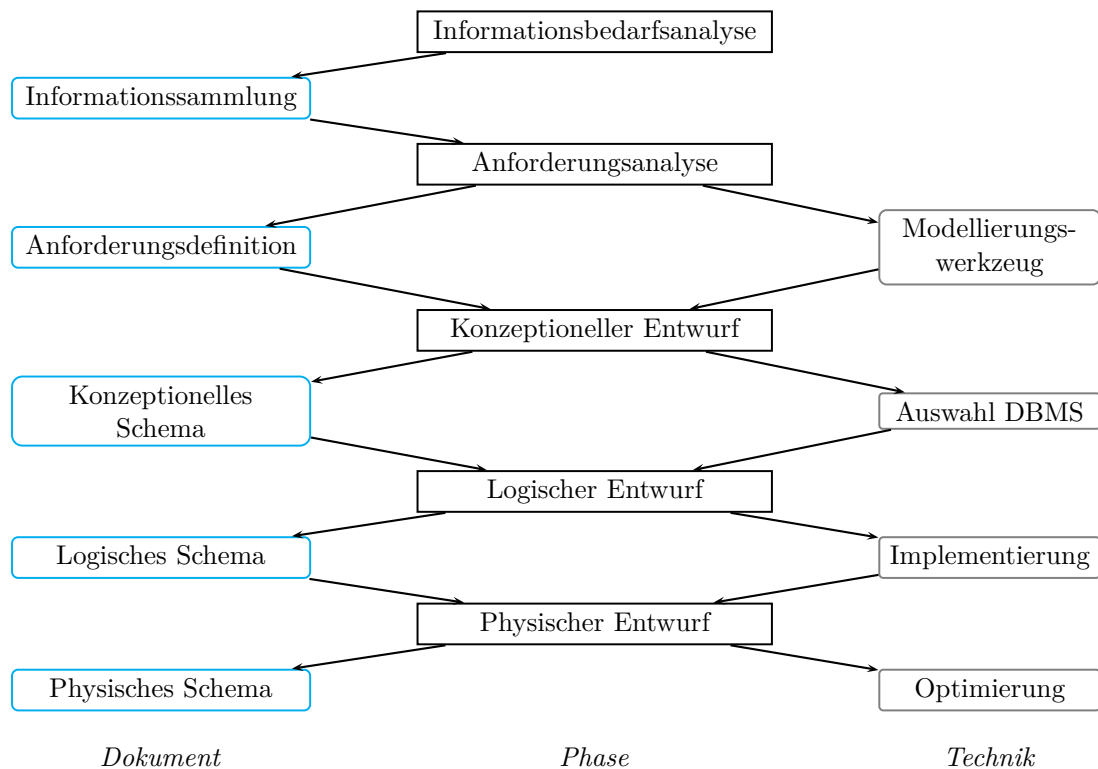


Abbildung 7: Der Entwurfsprozess

## 2.1. Das ER-Modell

Der Erfolg der Entwurfsphase hängt entscheidend davon ab, ob die Fraktion der Datenbank-Implementierer mit der Fraktion der Datenbank-Anwender gut kommunizieren kann. Hilfreich für diese Modellierungsphase ist eine sie beschreibende Sprache, die beide Parteien verstehen können. Im Jahre 1976 hat Peter Chen das sogenannte ER-Modell (*Entity-Relationship-Modell*) vorgeschlagen, das sich einer anschaulichen Darstellung bedient, die dieser Notwendigkeit Rechnung trägt. Das ER-Modell (auch *ERM*) war so erfolgreich, dass es sich bis heute als De-Facto-Standard gehalten hat. Die deutschsprachige Bezeichnung *Gegenstands-Beziehung-Modell* wird auch im deutschsprachigen Raum kaum verwendet. Abbildung 8 zeigt eine Übersicht.<sup>14</sup>

### 2.1.1. Entities

Kern des ER-Modells sind *Entities* und *Relationships*. Als Entity (im Deutschen auch als Entität bezeichnet) bezeichnet man wohlunterscheidbare Dinge der realen Welt ([Vos08, S. 61]). Beispiele sind Autos, Personen, Städte, Konten, Firmen usw. Einzelne Entitäten, die vergleichbar sind, werden zu einem *Entity-Set* oder *Entitätsklasse* zusammengefasst, beispielsweise alle Angestellten eines Betriebes. Entitäten haben Eigenschaften (Attribute) wie z. B. die Farbe des Autos, das Geburtsdatum der Person usw., deren konkrete Ausprägungen als

<sup>14</sup>Es existieren verschiedene „Dialekte“, sprich leicht unterschiedliche Darstellungsformen eines ER-Modells. Die hier angegebene Darstellung ist eine übliche Standardform.

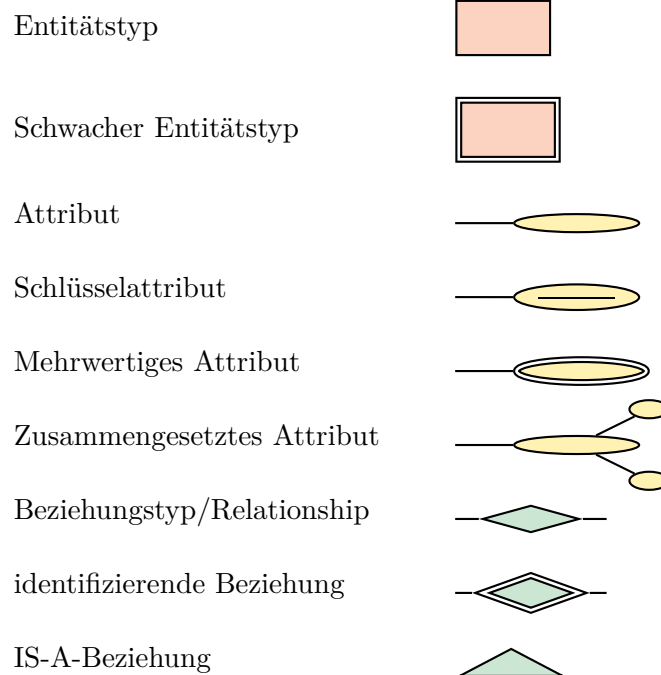


Abbildung 8: Das ER-Modell bedient sich der graphischen Darstellung seiner Komponenten.

Werte bezeichnet werden. Die Menge aller Werte für eine Eigenschaft wird als Wertebereich (engl. *domain*) bezeichnet.

Konzept	Notation	Beispiel
Entität	$e$	Angestellter Müller
Entitätstyp	$E$	Menge aller Angestellten-Attribute
Entitätsmenge	$Ang$	alle Angestellten
Attribut (Eigenschaft)		Geburtsdatum
– einwertig	$A$	Gehalt
– mehrwertig	$[A]$	Vornamen (alle gleichen Typs)
– zusammengesetzt	$A(B_1, B_2)$	Adresse (unterschiedliche Typen mgl.)
Schlüsselattribut	$\underline{A}$	<u>Personalnummer</u>

Als Beispiel möge der Angestellte einer Firma dienen, dessen Name, Adresse, Geburtsdatum und Gehalt in einer Datenbank festgehalten werden sollen. Um einen Datensatz eindeutig kennzeichnen zu können, erhält jeder Angestellte eine (künstliche) Personalnummer. Zwar ist es unwahrscheinlich aber sehr wohl möglich, dass es zwei Angestellte der Firma gibt, die in allen Attributwerten übereinstimmen. Eine Entität oder genauer ein Entitätstyp kann textlich oder bildlich dargestellt werden.

Entitätstyp "Angestellte":

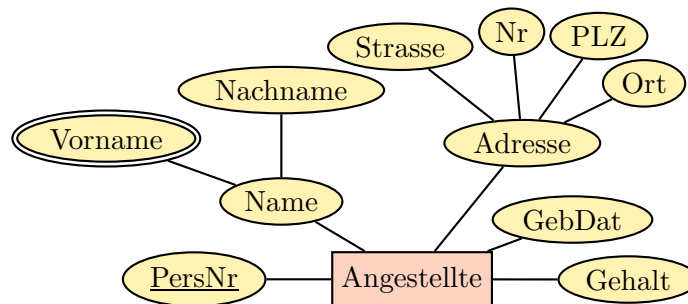
$$E = \{ \underline{\text{PersNr}}, \\ \text{Name}([\text{Vorname}], \text{Nachname}), \\ \text{Adresse}(\text{Strasse}, \text{Nr}, \text{PLZ}, \text{Ort}), \\ \text{GebDat}, \\ \text{Gehalt} \}$$


Abbildung 9: Graphische Darstellung eines Entitätstyps

Wie bereits erwähnt, liegt jedem Attribut  $A$  ein Wertebereich (engl. *domain*)  $D = \text{dom}(A)$  zugrunde. Man unterscheidet einwertige, mehrwertige und zusammengesetzte Attribute:

- Einwertige Attribute haben einfache Wertebereiche wie `int(12)`, `char(20)` oder `datum`.
- Mehrwertige Attribute haben Potenzmengen Wertebereiche. Sind dabei  $D$  Werte möglich, umfasst der Wertebereich  $2^D$  verschiedene Belegungen.
- Zusammengesetzte Attribute haben kartesische Produkte der Komponentenbereiche als Wertebereich.

Eine besondere Rolle kommt einem Attribut zu, wenn es sich um einen sogenannten Schlüssel handelt. Ein Schlüssel – wie die oben genannte Personalnummer eines Angestellten – dient dazu, eine Entität eindeutig zu kennzeichnen.

**Definition 2.1.1 (Schlüssel).** Sei  $E$  ein Entitätstyp und  $E^t$  eine Entitätsmenge zum Entitätstyp  $E$ .

**Superschlüssel:**

Gilt zu jedem Zeitpunkt für je zwei verschiedene Entitäten  $e_1, e_2 \in E^t$ , dass  $e_1 \neq e_2$  auf einer Teilmenge  $K$  der Attributmenge von  $E$  ist, so heißt  $K$  Superschlüssel von  $E$ .

**Schlüssel:**

Ist  $K$  Superschlüssel und minimal, so heißt  $K$  Schlüssel.

**Schlüsselkandidaten:**

Gibt es zu einem Entitätstyp mehrere Teilmengen der Attributmenge mit der Schlüsseleigenschaft, so heißen diese Mengen Schlüsselkandidaten.

Beispiel: Ein Entitätstyp *Kleinstadt* kann sowohl durch *Name+PLZ* als auch durch *Name+Vorwahl* identifiziert werden.

### 2.1.2. Relationships

Relationships – im Deutschen *Beziehungen* – stellen die zweite zentrale Kategorie des ER-Modells dar. Entitäten stehen üblicherweise zueinander in Beziehung. Bücher werden von Lesern ausgeliehen, Vorlesungen von Professoren gehalten. Beziehungen können – und haben in der Regel – ebenfalls Attribute. So findet der Buch-Ausleihvorgang an einem bestimmten Datum statt, Vorlesungen werden in einem bestimmten Semester gelesen.

**Definition 2.1.2** (Beziehungen – Relationships). Seien  $E_1, E_2$  zwei Entitätstypen,  $A$  eine Attributmenge.

**Binärer Beziehungstyp:**

$R = (E_1, E_2, A)$  heißt binärer Beziehungstyp (binary relationship) zwischen den Entitätstypen  $E_1$  und  $E_2$  mit der Attributmenge  $A$ . Eine Beziehung ist eine Teilmenge des kartesischen Produkts:  $R \subseteq E_1 \times E_2 \times A$ .

**Mehrstellige Beziehungstypen:**

Analog sind dreistellige (ternäre) und höhergradige Beziehungstypen definiert.

**Rekursiver Beziehungstyp:**

Ein binärer Beziehungstyp heißt rekursiv, wenn  $E_1 = E_2$ .

**Bemerkungen:**

- (a) Statt Beziehungstyp verwendet man vereinfachend oft auch den Begriff Beziehung.
- (b) Jedem Beziehungstyp ist eine zeitlich variante Beziehungsmenge zugeordnet, die sich aus den kartesischen Produkten der Entitätsmengen ergibt.
- (c) Rekursive Beziehungen bilden häufig zwei verschiedene Rollen des Entitytyps ab.

In der graphischen Darstellung einer Beziehung werden die Unterschiede sehr deutlich, wie Abbildung 10 zeigt.

Beziehungen sind zeitlich variant, da die an ihnen beteiligten Entitätsmengen sich im Laufe der Zeit ändern können. Dennoch haben die Beziehungstypen zeitinvariante Eigenschaften, die sich in der Komplexität der Beziehung widerspiegelt. So lesen z. B. Professoren mehrere Vorlesungen im Semester, eine Vorlesung wird aber in der Regel nur von einem Professor gelesen. Beim Ausleihen eines Buches zeigt sich dagegen eine 1 : 1-Beziehung: Ein Buch(-exemplar) kann zu jedem Zeitpunkt nur von genau einer Person ausgeliehen werden. Man nennt dies die Komplexität einer Beziehung.

**Definition 2.1.3** (Komplexität einer Beziehung). Sei  $R$  ein Beziehungstyp mit  $R = (E_1, \dots, E_n, X)$ , d. h.  $n$  beteiligten Entitätstypen und der zur Beziehung gehörigen Attributmenge  $X$ . Die Komplexität  $comp$  des Beziehungstyps  $R$  ist formal definiert:

$$comp(R, E_i) = (m..n) \quad : \iff \quad m \leq |Anz_e| \leq n$$

Dabei ist  $Anz_e = |\{r \in R^t \mid r[E_i] = e_i\}|$  für alle Zeiten  $t$  und alle  $e_i \in E_i$ .

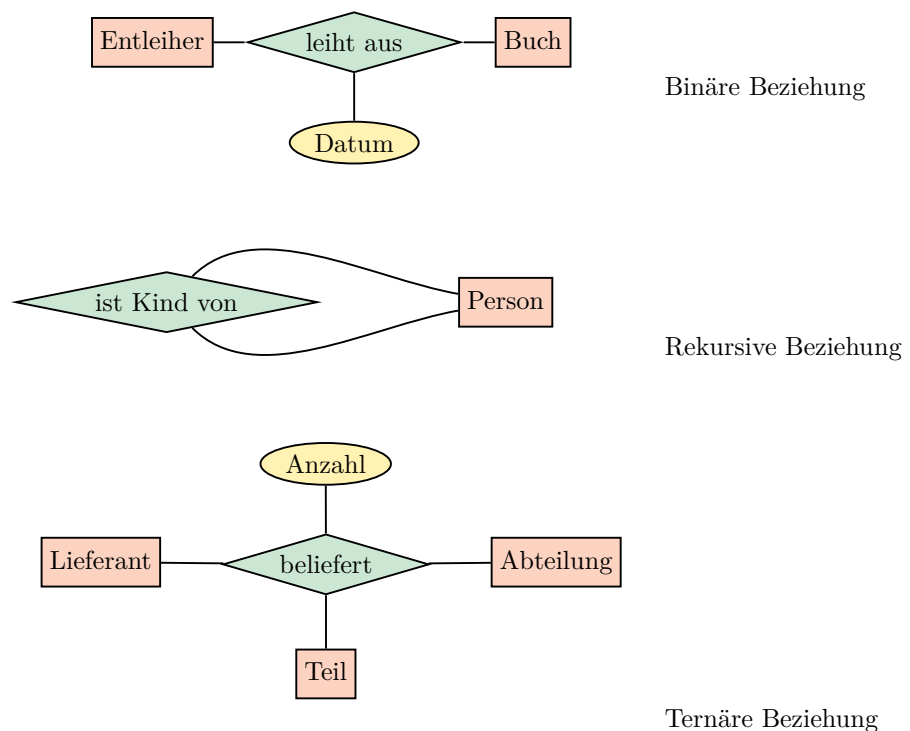


Abbildung 10: Beziehungen im ER-Modell graphisch dargestellt

In der graphischen Darstellung einer Beziehung wird die Komplexität an den Verbindungen zwischen der Beziehung und der entsprechenden Entität notiert (siehe Abbildung 11).

Komplexitäten  
Kardinalitäten

Sehr oft wird für binäre Beziehungen neben der sogenannten *min-max*-Notation der Komplexitäten eine andere gewählt. Das sogenannte *Kardinalitätsverhältnis* bzw. die *Chen-Notation* spezifiziert die Anzahl der Beziehungsinstanzen, an denen eine Entität teilnehmen kann. Für die Beziehung *istIn* von Schülern und Klassen gilt z. B. SCHUELER:KLASSE die Komplexität  $n:1$ . Folgende Tabelle zeigt eine Übersicht über die verschiedenen Komplexitätstypen der (binären) Beziehung  $R = (E_1, E_2, \mathcal{A})$ :

$R$	$comp(R, E_1)$	$comp(R, E_2)$
1:1	0..1 oder 1..1	0..1 oder 1..1
1:n	0..* oder 1..*	0..1 oder 1..1
m:n	0..* oder 1..*	0..* oder 1..*

Die Beziehungen heißen in der Chen-Notation

- $1 : 1$ , *one to one relationship*, 'eins zu eins'
- $1 : n$ , *one to many relationship*, 'eins zu viele'
- $m : n$ , *many to many relationship*, 'viele zu viele'

Durch Rollentausch von  $E_1$  und  $E_2$  erhält man  $n : 1$  Beziehungen.



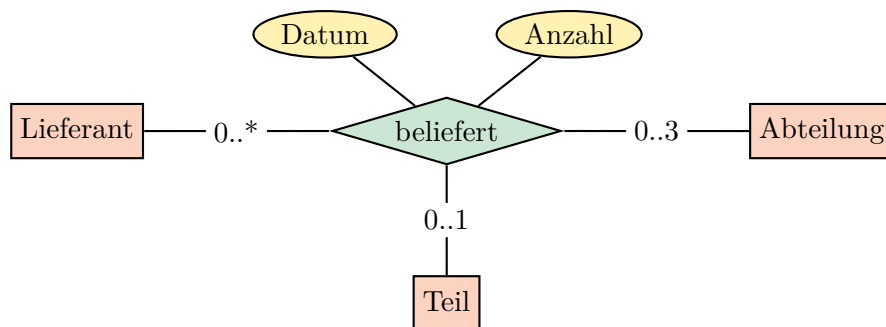


Abbildung 11: Beispiel der Beziehungskomplexitäten eines dreistelligen Beziehungstyps. Gegeben ist eine kleine Datenbank über Lieferungen an die Abteilungen einer Firma pro Monat. Die angegebenen Komplexitäten implizieren Integritätsbedingungen. So darf eine Abteilung maximal drei Lieferungen erhalten, ein Teil darf maximal einmal geliefert werden, um z. B. zu verhindern, dass durch mehrere kleine Bestellungen Mengenrabatt verloren geht. Gedanklich werden getätigte Lieferungen nach Ende des Monats aus der Datenbank gelöscht.

### 2.1.3. Weitere Konzepte

Viele semantische Eigenheiten eines Realweltausschnittes lassen sich mit den vorgestellten Konzepten noch nicht befriedigend darstellen. Neben den bisher beschriebenen Entitäten gibt es sogenannte *schwache Entitäten*, die nur existieren, wenn es einen übergeordneten Entitätstyp gibt, von dem der schwache Typ abhängt. So gibt es beispielsweise ein Konto nur dann, wenn es einen eingetragenen Kontoinhaber gibt, oder eine Bankfiliale existiert nur mit einer übergeordneten Bank. Schwache Entitäten haben daher auch keinen eigenen Schlüssel und stehen mit einem starken Typ in einer  $n : 1$ -Beziehung. Um dennoch einen Schlüssel für einen schwachen Entitätstyp zu erhalten, nimmt man oft eine Teilmenge der Attributmenge des schwachen Typs zusammen mit dem Schlüssel des starken, um einen Schlüssel für den schwachen Typ zu erhalten. Eine solche Beziehung nennt man *identifizierend*. So ist beispielsweise die Raumnummer zusammen mit der Gebäudenummer ein Schlüssel für einen Raum eines Gebäudes (siehe Abbildung 12).

Schwache Entitäten

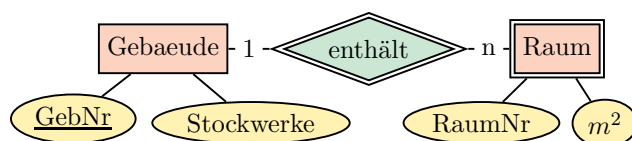


Abbildung 12: Das Konzept der schwachen Entität

Ein weiteres wichtiges Konzept ist das der Spezialisierung. So gibt es beispielsweise Entitätstypen, die sehr ähnlich aber doch nicht gleich sind, weil sie sich in einem oder zwei Attributen unterscheiden. So stelle man sich Angestellte einer Fluggesellschaft vor (Piloten, Flugbegleiter, Bodenpersonal). Während viele Attributmengen gleich sind, so ist für einen Piloten z. B. die Art der Fluglizenz (Langstrecke, Kurzstrecke) wichtig, die für einen Flugbegleiter nicht geführt werden muss. Und Flugbegleiter sprechen z. B. mehrere Sprachen, so dass sie auf unterschiedlichen Flügen eingesetzt werden können, während ein Pilot nur

Spezialisierung,  
IS-A-Beziehung

Englisch sprechen muss. Beim Bodenpersonal kommt es wiederum darauf an, zu welcher Abteilung es gehört (z. B. Service, Security o. Ä.).

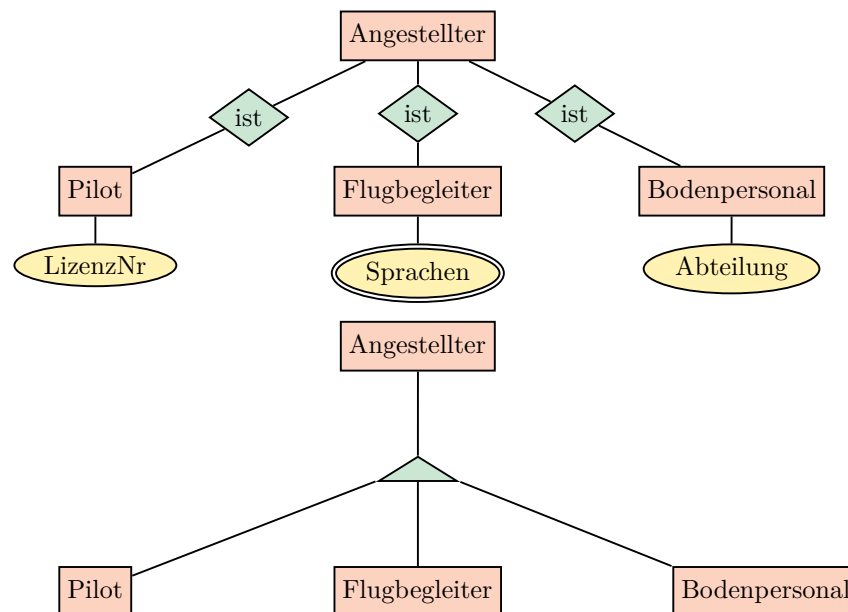


Abbildung 13: Das Konzept der Spezialisierung am Beispiel der Angestellten einer Fluggesellschaft. Darstellung von IS-A-Beziehungen kann auch als einfaches Dreieck erfolgen.

Entitäten eines Entitätstyps mit *zusätzlichen besonderen Eigenschaften* werden in einer abhängigen Entität modelliert. Die abhängige Entität 'ist eine' Entität der übergeordneten Sorte (jeder Flugbegleiter ist ein Angestellter) und zeichnet sich durch zusätzliche Attribute aus. Die 'ist ein'-Beziehung zwischen spezialisiertem und allgemeinem Entitätstyp heißt **IS-A-Beziehung**. Bei einer Entität des Untertyps handelt es sich um *kein zusätzliches* Objekt in der realen Welt. Vielmehr ist es *das selbe* Objekt der realen Welt wie in der Oberklasse (z. B.: Pilotin Marie Schultze *ist* die Angestellte Marie Schultze), es wird im Untertyp lediglich unter dem Blickwinkel seiner speziellen Rolle betrachtet. Daher ist es möglich, dass eine Entität des Overtyps in mehreren Untertypen enthalten sein kann, wenn sie mehrere Spezialisierungen hat.

**Definition 2.1.4** (IS-A-Beziehung). Man sagt, ein Entitätstyp  $E_1$  IS-A-Entitätstyp  $E_2$ , genau dann wenn gilt:

- (a) Alle Attribute von  $E_2$  kommen bei  $E_1$  vor.
- (b) Zu jedem Zeitpunkt  $t$  gilt: Für jede Entität  $e_1 \in E_1^t$  gibt es ein  $e_2 \in E_2^t$  mit  $e_1(A) = e_2(A)$  für alle Attribute  $A$  aus  $E_2$ .

Kurz schreibt man einfach  $E_1 \subseteq E_2$ .

Die Zerlegung der Gesamtmenge der Overtypen in abhängige Subtypen kann sowohl disjunkt oder überlappend als auch partiell oder total sein. Im Beispiel aus Abbildung 13 ist die Zerlegung total, wenn es keine Angestellten außer Piloten, Flugbegleiter oder Bodenpersonal

gibt. Und die Zerlegung ist disjunkt, wenn kein Angestellter aus einer der drei Subtypen zusätzlich in der Rolle eines anderen Subtyps auftritt.

Zwischen Subtyp und Obertyp treten folgende Mengenbeziehungen auf:

Gesamtmenge/Zerlegung	disjunkt	überlappend
total	t/d	t/ü
partiell	p/d	p/ü

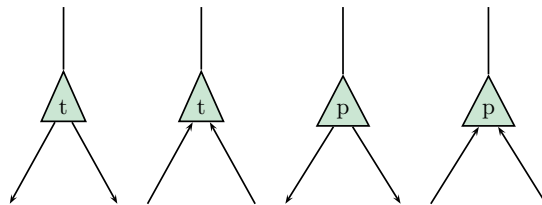


Abbildung 14: Bildliche Darstellung von Subtyp/Obertyp-Mengenbeziehungen

#### 2.1.4. Ein schneller Weg in ein ER-Modell

Um einen schnellen Einstieg in die Transformation einer Anforderungsdefinition in ein ER-Modell zu ermöglichen (s. Abb. 7: Anforderungsanalyse -> Konzeptionelles Schema), hier zwei zentrale Vorgehensweisen:

- Im Text der Anforderungsanalyse stehen *Substantive* i. d. R. für wohlunterscheidbare Dinge des gleichen Typs (Autos, Personen, Abteilungen, Produkte etc.), von denen bestimmte Eigenschaften festgehalten werden sollen (Farbe, Modell, Preis, Name, Adresse etc.). Damit sind Substantive eine der Hauptquellen für die Identifizierung von Entitätstypen und deren Eigenschaften entsprechen den zugehörigen Attributen.
- Im Text verwendete *Verben* stellen i. d. R. Verbindungen zwischen Substantiven her (Person *besitzt* Auto, Kunde *bestellt* Produkt, Mitarbeiter *arbeitet* in Abteilung etc.). Diese werden als Beziehungstypen in das ER-Modell aufgenommen, die Entitätstypen miteinander verknüpfen.

Weiteres dazu im Kapitel 2.3.

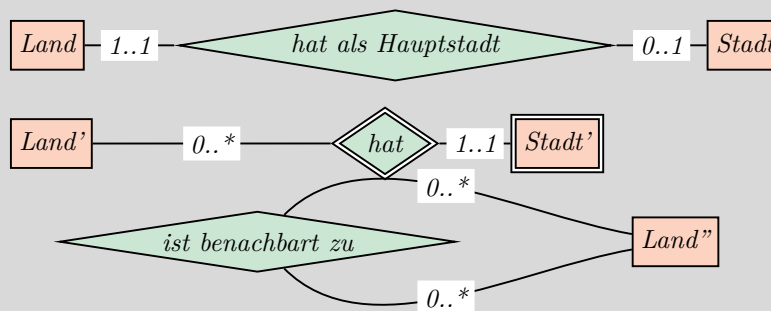
## 2.2. Übungszettel A-2

**Aufgabe 2.1.** Ein Wachschutzunternehmen möchte für Einsatzplanung und Einsatzabrechnung ein Datenbanksystem einsetzen. Dazu sollen Angaben zu den Mitarbeitern und Fahrzeugen des Unternehmens sowie dessen Kunden und deren Aufträgen gespeichert werden. Bei den Mitarbeitern wird zwischen Festangestellten und Aushilfskräften unterschieden, die für eine vereinbarte Stundenzahl eingestellt werden. Darüberhinaus sind nicht alle Mitarbeiter gleich qualifiziert, beispielsweise besitzt nicht jeder einen Führerschein. Jeder Mitarbeiter muss jedoch mind. eine Qualifikation haben. Bei den Aufträgen wird zwischen Daueraufträgen und einzelnen Aufträgen unterschieden. Die regelmäßige Überwachung eines Gebäudes stellt beispielsweise einen Dauerauftrag dar, die Sicherung eines Sportstadions (internat. Sportveranstaltung) erfordert nur einen einmaligen Einsatz. Die Kunden geben in den Aufträgen an, welchen Personal- und Fahrzeugbedarf sie haben, wobei nur max. fünf Fahrzeuge pro Einsatz zu Verfügung gestellt werden.

Mit Hilfe des Systems sollen Fragen der folgenden Art beantwortet werden:

- Welche Mitarbeiter und welche Fahrzeuge stehen für einen Einsatz zur Verfügung?
  - Welche Mitarbeiter und welche Fahrzeuge waren an einem Einsatz beteiligt?
  - Wer hat welches Fahrzeug gefahren?
  - Welche Kosten hat ein Auftrag verursacht?
- (a) Entwickeln Sie ein ER-Schema für die obige Anwendung. Treffen Sie bei Unklarheiten eigene Entscheidungen für Ihre Modellierung. Ergänzen Sie dabei das Schema um sinnvolle Attribute.
- (b) Fügen Sie Komplexitäten in ihr Schema ein.
- (c) Charakterisieren Sie alle Komplexitäten durch ein bis zwei Sätze.
- (d) Erläutern Sie die von Ihnen gemachten Annahmen und begründen Sie somit Ihre Entwurfsentscheidungen.
- (e) Nennen Sie Bedingungen der realen Welt bzw. der genannten Anforderungen, die sich in Ihrem ER-Modell nicht wiederfinden lassen.

**Aufgabe 2.2.** Die folgenden Ausschnitte entstammen Überlegungen zu einem 'Stadt-Land-Fluss' Datenbankentwurf. Sind sie plausibel modelliert (Begründung)? Geben Sie die modellierten Integritätsbedingungen möglichst präzise an, jeweils durch ein Beispiel gestützt.



**Aufgabe 2.3.** In einer Miniwelt von klassischen Filialbanken gibt es Banken und Kunden. Eine Bank wird durch ihre Bankleitzahl und ihren Namen beschrieben. Sie besitzt mindestens eine Zweigstelle. Jede Zweigstelle hat eine Adresse und eine innerhalb der Bank eindeutige Zweigstellenummer. Eine Zweigstelle verwaltet die Konten der Kunden. Es gibt Guthabenkonten und Kreditkonten. Zu beiden Arten gibt es wieder Unterarten (Girokonten und Sparbücher bei den Guthabenkonten, Konten für allgemeine Kredite und Baukredite bei den Kreditkonten etwa).

Jedes Konto hat innerhalb der Miniwelt eine eindeutige Kontonummer, einen Kontostand und einen festgelegten Zinssatz. Kreditkonten benötigen eine fest vereinbarte monatliche Tilgung. Jedes Kreditkonto braucht zur Sicherheit zwei Kontoinhaber.

Die Kunden werden mit Vorname, Nachname, Geburtsdatum, Wohnanschrift und einer bankinternen Personenkennzahl gespeichert. Kunden sind nur solche, die mindestens ein Konto besitzen. Kunden können mehrere Konten beliebiger Art haben, für jedes Konto können mehrere Inhaber eingetragen sein. Allerdings darf es keine Kunden geben, die nur ein Kreditkonto oder mehr als drei Kreditkonten haben.

Erstellen Sie einen ER-Entwurf mit Angabe der Komplexitäten. Begründen Sie alle Komplexitätsbeziehungen durch zwei Sätze. Geben Sie zum Vergleich auch die Kardinalitäten (bzw. die Chen-Notation) an. Weisen Sie schwache Entitäten aus. Dokumentieren Sie Ihre Entwurfsentscheidungen.

**Aufgabe 2.4.** Ein Staatstheater mit mehreren Bühnen (Großes Haus, Kammerspiele, Probebühne, ...) bespielt diese Spielstätten mit eigenen Inszenierungen von Stücken unterschiedlicher Sparten (Komödien, Trauerspiele, Dramen, ...). Autoren aus allen Epochen der Literaturgeschichte kommen zur Aufführung. Die am Theater dauerhaft verpflichteten oder ggf. über Zeitverträge engagierten Schauspieler verkörpern ihre Rollen Abend für Abend in für das jeweilige Stück festen Besetzungen. Neben den Rollenzuordnungen des Spielplans beherrschen die Schauspieler ein gewisses zusätzliches Rollenspektrum, so dass bei Erkrankungen Vertretungen möglich sind. Ohne die vielen 'unsichtbaren Geister', die technischen Mitarbeiter (Bühnenarbeiter, Maskenbildner, ...) und die künstlerischen Mitarbeiter (Regisseur, Dramaturg, ...), könnte die Arbeit nicht gelingen. Auch diese Mitarbeiter sind mit Dauerverträgen oder mit Zeitverträgen dabei, wobei es maximal drei Zeitverträge pro Mitarbeiter geben darf. Die Inszenierungen werden in monatelanger Probenarbeit für eine der Bühnen einstudiert.

- (a) Erstellen Sie für den skizzierten Weltausschnitt ein ER-Modell mit Angabe der Komplexitäten. Bei mehreren Modellierungsmöglichkeiten entscheiden Sie sich begründet für eine. Beschränken Sie sich auf die nötigsten Attribute, um nicht zu umfangreich zu werden.
- (b) Das von Ihnen entworfene Modell kann vermutlich viele in der Realität auftretenden Bedingungen nicht repräsentieren. Geben Sie als Illustration zwei solcher Bedingungen an.

**Aufgabe 2.5.** Ein schwacher Entitätstyp könnte leicht durch Hinzunahme weiterer Attribute zu einem starken Entitätstyp gemacht werden. Warum führt man überhaupt schwache ein?

### 2.3. Von der Anforderungsdefinition zum ER-Modell

Am Ende der Informationsbedarfsanalyse ist – neben vielen anderen Dokumenten – ein kondensierter, umgangssprachlicher Text entstanden, oft als *Miniwelt* bezeichnet, der in einheitlicher Sprechweise die Struktur der Anwendung beschreibt.

Die Aufgabe des konzeptuellen Entwurfs ist dann, ein Entity-Relationship-Modell zu entwerfen. Dazu ist eine systematische Vorgehensweise angebracht. Einige Regeln:

- (a) *Substantive* bezeichnen in der Regel Entitäten oder Entitätstypen. Ob das konkrete Objekt oder die Klasse gemeint ist<sup>15</sup>, muss aus dem Zusammenhang erschlossen werden. Der Name eines Entitätstyps wird in der Mitte eines Rechtecks notiert.
- (b) *Verben* in jeder grammatikalischen Form werden in der Regel als Beziehungen bzw. Relationships modelliert. Auch hier muss im Prinzip aus dem Zusammenhang entschieden werden, ob eine konkrete Beziehung oder ein Beziehungstyp gemeint ist.<sup>16</sup> Der Name eines Beziehungstyps wird in das Zentrum einer Raute geschrieben. Die Wahl des Beziehungsnamens erfordert Fingerspitzengefühl. Je nach Anordnung – bezogen auf die Leserichtung – kann die aktive oder passive Form ausgewählt werden ('liefert an' oder 'wird geliefert von'). Eine einheitliche Anordnung ist für das Lesen der Beziehungen hier unbedingt von Vorteil. Die Rauten werden in der Regel durch Strecken mit den beteiligten Entitäten verbunden.
- (c) Konkret genannte Entitäten oder Relationships, also in der *Objektrolle*, müssen daraufhin untersucht werden, ob sich die Verallgemeinerung in einer statischen Datenmodellierung lohnt. Möglicherweise gibt es so wenige Exemplare, dass sich der Aufwand dafür nicht rechnet. Z. B. besitzt eine Firma mglw. zwei Lagerhäuser. Es spielt jedoch in dieser Firma keine Rolle, welche Eigenschaften diese Lagerhäuser haben, es muss lediglich vermerkt werden, in welchem Lagerhaus ein Gegenstand liegt. Diese können dann außerhalb der Datenbank existieren oder – üblicherweise mit Einschränkungen – in anderen Typen untergebracht werden. In dem Beispiel könnte bei den einzulagernden Gegenständen ein Attribut eingeführt werden, das anzeigt, in welchem Lager das Teil liegt.
- (d) Ist eine *rekursive* Beziehung gegeben (Beispiel: 'ist Mutter von' bei 'Person'), so werden an den Verbindungsstrecken die *Rollen* notiert, im Beispiel 'Mutter' und 'Kind'.
- (e) Eine Analyse des Textes sollte auch Hinweise auf die *Komplexitäten* oder *Kardinalitäten* der Beziehungstypen liefern. Bei der Verwendung von *Kardinalitäten* sollte die erforderliche Beteiligung von Entitätstypen untersucht werden.
  - "*wenigstens ein ...*" übersetzt sich in die Kardinalität  $n$  mit verpflichtender Beteiligung der Entität. Aus der Sicht dieser Entität ist ihre Beteiligung an einer Beziehung also *total*.
  - "*höchstens ein ...*" übersetzt sich in die Kardinalität 1 mit wahlfreier Beteiligung, ist also *partiell*.

<sup>15</sup>Z. B. konkret *der Angestellte Herr Mustermann* oder allgemein *Angestellte*.

<sup>16</sup>Z. B. konkret *Herr Mustermann arbeitet in der Personalabteilung* oder allgemein *Angestellte arbeiten in Abteilungen*.

- Aus Sicht der Beziehungen bzw. Relationen ergeben sich in mehreren Fällen eine spezielle Form der Relation: eine *Funktion*. Dies ist der Fall, wenn eine Entität aus dem Entitätstyp  $E_1$  maximal *eine* und damit *eindeutige* Relation zu einer Entität aus einem anderen Entitätstyp  $E_2$  eingeht. Je nach dem, ob und wie oft eine Entität aus  $E_2$  an diesem Beziehungstyp beteiligt sein kann, sind die Beziehungsfunktionen
  - *surjektiv*: jede Entität in  $E_2$  muss *mind. eine* Beziehung eingehen, kann aber auch zu mehreren Entitäten in  $E_1$  in Beziehung stehen,
  - *injektiv*: jede Entität in  $E_2$  kann *max. eine* Beziehung zu einer Entität in  $E_1$  eingehen, muss aber nicht, oder
  - *bijektiv*: jede Entität in  $E_2$  muss genau eine Beziehung mit einer Entität in  $E_1$  eingehen. Formal gesehen ist sie beides, *surjektiv* und *injektiv*.

Bei der Festlegung der Komplexitäten  $comp(E_i, R), i \in \{1, 2\}$  lassen sich vier typische Standardfälle festmachen: 1..1, 0..1, 1..\* und 0..\*. Für diese vier Fälle existiert eine vereinfachte Notation:

$comp$	vereinfachte Notation	Bedeutung
1..1	1	muss vorkommen, steht zu genau einem in Bez.
0..1	c (choice)	muss nicht vorkommen, steht zu genau einem in Bez.
1..*	m (multiple)	muss vorkommen, Beziehung zu vielen
0..*	mc (multiple choice)	muss nicht vorkommen, Beziehung zu vielen

Damit ergeben sich für eine zweistellige Entitätenbeziehung eine Reihe von Kombinationsmöglichkeiten:

$E_1$	$E_2$	Relation, Funktion	$m : n$ -Not.	Beispiel
c	c	partiell, injektiv	1:1	Ang. – hat – Schreibtisch
c	1	partiell, bijektiv		Person – hat – Reisepass
1	c	total, injektiv		Hauptstadt – Bundesland
1	1	total, bijektiv		Pkw – Kfz-Schein
c	m	partiell, surjektiv	n:1/1:n	Artikel – Lieferung
1	m	total, surjektiv		Ang. – gehört zu – Abt.
c	mc	partiell		Entleiher – leiht – Buch
1	mc	total		Diplomat – vertritt – Land
m	mc	Relation	m:n	Lehrv. – belegt von – Stud.
mc	m			(umgekehrt)
m	m			Lehrer – unt. – Klasse
mc	mc			Lieferant – liefert – Artikel

### 2.3.1. Ausführliches Beispiel

In diesem Kapitel soll ein Beispiel eines umfangreicheren ER-Entwurfes vorgestellt werden.<sup>17</sup> Das Beispiel bietet neben fundamentalen Konzepten auch eine eingehende Beschäftigung mit der Überführung n-ärer Beziehungen in binäre Beziehungen, die oft mit Informationsverlust verbunden ist.

#### Miniwelt der Weine

Ein Sterne-Restaurant, das sich besonders auf Weine spezialisiert hat, möchte die angebotenen Weine und Gerichte verwalten. Ein Wein zeichnet sich durch einen Namen, eine Farbe, Restsüße und den Jahrgang aus. Kritiker, die einer bestimmten Organisation angehören, empfehlen den Genuss von Weinen zu bestimmten Gerichten. Ein Gericht hat eine Bezeichnung und Beilagen. Weine werden aus (u. U. mehreren) Rebsorten hergestellt, aber alles, was aus mehr als sieben Sorten besteht, zählt nicht mehr als Wein. Eine Rebsorte hat einen Namen und eine Farbe; jede Rebsorte tritt in einem festen Anteil in einem Wein auf. Weine werden exklusiv von einem Erzeuger produziert, der eine Adresse hat und ein Weingut sein eigen nennt. Ein Weingut befindet sich in einem Weinanbaugebiet, das durch Name, Land und Region gekennzeichnet ist. Um einen Wein in einer bestimmten Menge produzieren zu dürfen, bedarf es einer Lizenz, die durch eine Lizenznummer eindeutig gekennzeichnet ist und höchstens einmal vergeben wird.

Bei der Modellierung von ER-Modellen kann es immer wieder vorkommen, dass n-äre Beziehungen auftreten, die über eine binäre Beziehung hinausgehen. Da diese schnell unübersichtlich werden können, stellt sich die Frage, ob diese Typen auch in binäre Beziehungstypen umgewandelt werden können. In unserem Weinmodell tritt als Besonderheit der ternäre Beziehungstyp *empfiehlt* auf. Abbildung 15 zeigt die ternäre Version und die Umsetzung in drei binäre Typen.

Die Übersetzung in drei binäre Beziehungen bringt aber Probleme mit sich. So wird durch die Beziehung *empfiehlt* eine Zuordnung eines Kritikers zu einem Gericht und einem Wein vorgenommen. Es sind beispielsweise folgende Ausprägungen möglich:

empfiehlt		
Kritiker	Wein	Gericht
$k_1$	$w_1$	$g_1$
$k_2$	$w_2$	$g_1$
$k_2$	$w_1$	$g_2$

Modelliert man das in drei binären Beziehungen, wird das zu K-W, G-W und G-K:

<sup>17</sup>Das Beispiel entstammt [SSH10, S.66 ff.].



K-W		G-W		G-K	
Kritiker	Wein	Gericht	Wein	Gericht	Kritiker
$k_1$	$w_1$	$g_1$	$w_1$	$g_1$	$k_1$
$k_2$	$w_2$	$g_1$	$w_2$	$g_1$	$k_2$
$k_2$	$w_1$	$g_2$	$w_1$	$g_2$	$k_2$

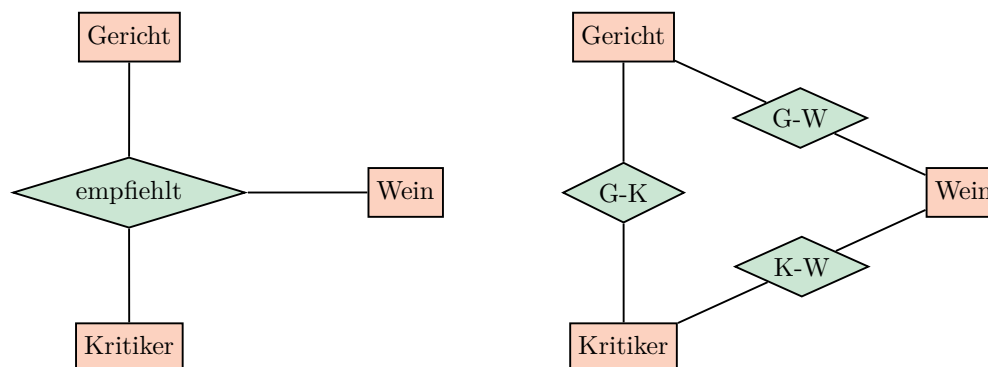


Abbildung 15: Der ternäre Beziehungstyp **empfeht** und seine Übersetzung in drei binäre Beziehungstypen

Durch diese Modellierung geht z. B. die Zusammenhörigkeit  $g_1 - w_1$  bzw.  $g_1 - k_1$  verloren. Will man aus den binären Beziehungstypen die ternäre zurückgewinnen, stellt man fest, dass dies nicht mit dem gleichen Informationsgehalt möglich ist, weil ...

- ... neue ternäre Beziehungen (nicht -typen!) ableitbar sind wie z. B.  $g_1 - w_1 - k_2$ . Damit kann ein wichtiger Aspekt der ursprünglichen Beziehungsausprägung nicht mehr rekonstruiert werden, nämlich der, dass  $k_2$  den Wein  $w_1$  nur für das Gericht  $g_2$  empfiehlt und *nicht* für  $g_1$ . Es findet ein Informationsverlust statt und Falschinformationen können abgeleitet werden.
- ... Informationen ausgedrückt werden können, die vorher nicht ausdrückbar waren: Man kann z. B. die Instanz  $g_3 - w_3$  einfügen, eine Empfehlung ohne Zuordnung zu einem Kritiker.

Das Beispiel zeigt, dass die direkte Umsetzung von  $n$ -stelligen Beziehungstypen in zweistellige zu unerwünschten Effekten führen kann. Das bedeutet aber nicht, dass man mit zweistelligen prinzipiell eine geringere Ausdrucksfähigkeit hat. Man kann z. B. die ternäre Beziehung **empfeht** durch eine künstliche Entität **Empfehlung** ersetzen, wie Abbildung 16 zeigt. Besondere Aufmerksamkeit verdienen  $n : 1$ -Beziehungen. In diesen Beziehungstypen wird jeder Entität  $e_1$  eines Entitätstyps  $E_1$  maximal eine Entität des Typs  $E_2$  zugeordnet. Eine solche Zuordnung nennt man auch **funktional**. Solche Beziehungstypen kommen in der Praxis recht häufig vor und eignen sich unter dem Implementierungsaspekt gut, weil sie zum navigierenden Zugriff verwendet werden können.

$n : 1$  heißt funktional

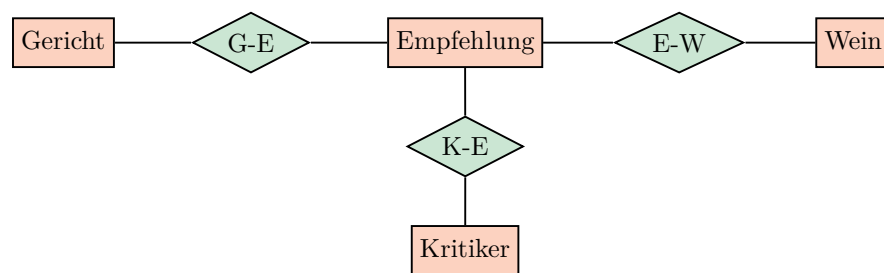


Abbildung 16: Dreistellige Beziehung durch Einführung eines künstlichen Entitäts-Typs

### Erweiterung der Wein-Miniwelt

In der Datenbank sollen zusätzlich Schaumweine verwaltet werden, die sich von normalen Weinen vom Gehalt an Kohlendioxid unterscheiden, das in einem speziellen Verfahren in den Wein eingebracht wird. Des weiteren unterscheiden sich verschiedene Jahrgänge aller Weines so deutlich in der Qualität, dass auch sie in der Datenbank erfasst werden sollen.

Will man diesen Erweiterungen Rechnung tragen, bietet es sich an, eine schwache Entität Schaumwein zu modellieren (siehe Abbildung 19) und eine Entität Wein-Jahrgang über eine Beziehung gehört zu einem Wein (genauer: Weinsorte) zuzuordnen (siehe Abbildung 18). Offenbar gilt:

- Jeder Schaumwein ist ein Wein (bzw. einer Instanz vom Entity-Typ Wein(-sorte)) zugeordnet, aber nicht jeder Wein ist ein Schaumwein.
- Zu jedem Weinjahrgang gibt es eine Sorte, aber nicht zu jeder Sorte muss es einen Jahrgang geben.

Diese beiden Erweiterungen sowie die Umwandlung der ternären Beziehung in eine neue Entität *Empfehlung* lassen sich leicht in das ER-Modell aus Abbildung 17 integrieren. Dennoch bildet das ER-Modell keine Integritätsbedingungen ab (als Farbe soll z. B. nur *rot*, *weiß* und *rosé* erlaubt sein). Hier zeigt sich, dass das ER-Modell nur zur statischen Modellierung taugt.

### 2.3.2. Zusammenfassung

Beim Erstellen eines ER-Modells sind das Erstellen einer Anforderungsdefinition und eines Pflichtenheftes unabdingbare Voraussetzungen. Dabei müssen ...

- ... die Entitätstypen (Name, Attribute) festgelegt werden.
- ... die Schlüsselmenge des Entitätstyps festgelegt werden.<sup>18</sup>
- ... die Beziehungstypen aufgefunden werden (Name, Attribute).
- ... Hierarchien und Generalisierungen ermittelt werden (Stichwort: isa).
- ... die Komplexitäten (Kardinalitäten) ermittelt und notiert werden.

<sup>18</sup>Ist man sich bei der Schlüsselfestlegung nicht sicher, die Einzigartigkeit einer Ausprägung über die Schlüsselmenge zu jedem Zeitpunkt zusichern zu können, legt man einen eindeutigen künstlichen Schlüssel fest.

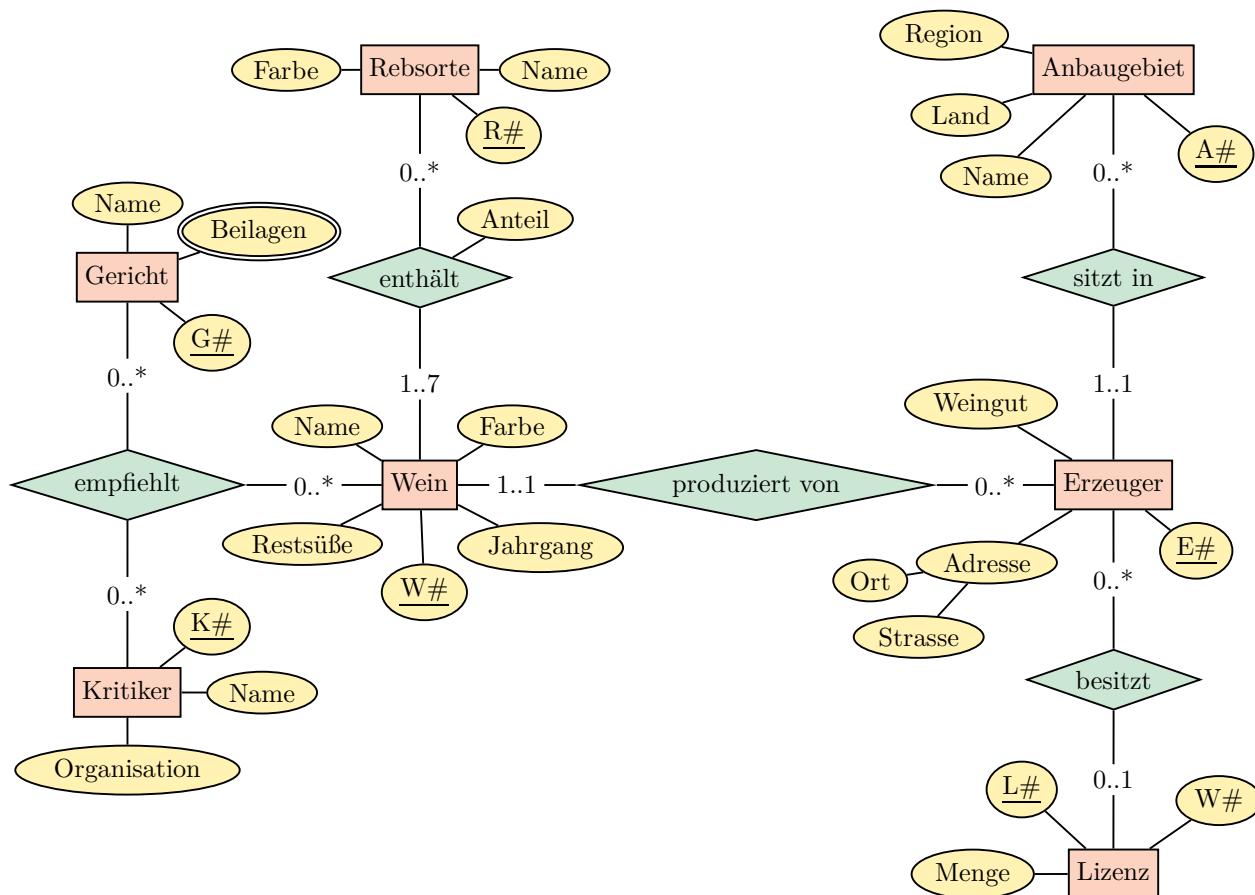


Abbildung 17: Modellierung der Wein-Miniwelt eines Sterne-Restaurants mit ternärem empfiehlt-Beziehungstyp.

Zusammenfassend lässt sich sagen, dass das ER-Modell eine Reihe von Vorteilen in der Entwurfsphase aufweist, allerdings auch einige Schwierigkeiten ungelöst lässt. Folgende Übersicht fasst dies zusammen:

### Vorteile

- einfaches, tragfähiges Konzept
- übersichtliche graphische Notation, die den Gesamtüberblick fördert
- formalisierbar

### Nachteile

- rein statische Modellierung
- Zusatzinformationen nötig (Integritätsbedingungen, Aktionen)
- keine einheitliche Norm, viele Dialekte

## Literatur zu Kapitel 2

- [KE09] KEMPER, Alfons ; EICKLER, André: *Datenbanksysteme: Eine Einführung*. 7. Auflage. München : Oldenbourg Wissenschaftsverlag, 2009. – ISBN 978-3-486-59018-0
- [SSH10] SAAKE, Gunter ; SATTler, Kai-Uwe ; HEUER, Andreas: *Datenbanken: Konzepte und Sprachen*. 4. Auflage. Heidelberg : mitp, 2010. – ISBN 978-3-8266-9057-0



Abbildung 18: Abhängige Entitäten im ER-Modell: Wein(-sorten) können ohne Wein-Jahrgänge existieren, ein Jahrgang ohne die zugehörige Sorte dagegen nicht.

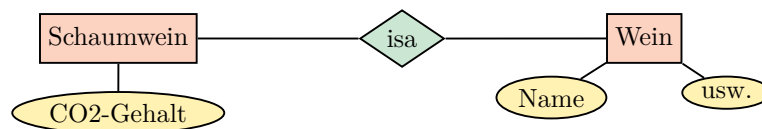


Abbildung 19: Abhängige Entitäten im ER-Modell: Jeder Schaumwein ist ein Wein, der durch das jeweilige Herstellungsverfahren gekennzeichnet ist.

[Vos08] VOSSEN, Gottfried: *Datenmodelle, Datenbanksprachen und Datenbankmanagementsysteme*. 5. Auflage. München : Oldenbourg, 2008. – ISBN 978-3-4862-7574-2

## 2.4. Übungszettel A-3

**Aufgabe 3.1.** Legen Sie die Komplexitäten der angegebenen Beziehungstypen fest. Begründen Sie Ihre Lösungen. Bilden die Beziehungstypen eine Funktion ab?

- (a) Sachbearbeiter – hat Arbeitsplatz in – Raum
- (b) Studenten – besuchen – Cafeterien
- (c) Schüler – hat – Schülerschein

**Aufgabe 3.2.** Geben Sie für den folgenden Ausschnitt einer Firmenwelt eine Generalisierungshierarchie an. Machen Sie dabei alle Annahmen bzgl. der Zerlegungsart (t/p, d/ü) durch knappe Formulierungen explizit.

Die in der Firma beschäftigten Personen sind z. B. Abteilungsleiter, Produktions- oder Sekretariatsangestellte. Es gibt Abteilungsleiter für die Produktion und für die Verwaltung. Die Angestellten in der Produktion arbeiten in den Sparten Konstruktion, Apparatebau und Lager.

**Aufgabe 3.3.** Ein Reisebüro ist auf Busreisen spezialisiert. Es fährt mit eigenen Bussen, mietet aber ab und an auch Busse dazu, wenn mehr Kunden die Reise gebucht haben, als in den Bus passen. Fällt ein Bus aus, wird ebenfalls ein Mietbus genommen. Zur Begleitung der Reisen gibt es angestellte und freie Mitarbeiter. Reisen werden für Kunden ab einer bestimmten Kundenzahl pro Bus durchgeführt. Sie haben ein Ziel, eine Dauer und einen Preis. Entwerfen Sie unter Angabe sinnvoller Annahmen ein ER-Modell für dieses Einsatzfeld an und ergänzen Sie sinnvolle Attribute. Notieren Sie alle Komplexitäten in (min,max)-Notation. Verbalisieren Sie die Komplexitäten durch je ein bis zwei knappe Sätze. Gibt es zusätzliche Integritätsbedingungen, die in der Modellierung nicht ausgedrückt sind?

**Aufgabe 3.4.** Eine Süßwarengroßhandelskette unterhält in der ganzen Republik Vertreter, die für ihre Dienste unterschiedliche Provisionen bekommen. Sie vermitteln Großaufträge von Supermarktketten-Filialen in ihrem Einzugsgebiet. Die Filialleiter handeln mit Vertretern unterschiedliche Discountbeträge für Produkte aus. Diese hängen von der Menge ab, mit der ein Produkt bestellt wird. Vertrieben werden Produkte wie Schokolade, Kekse, Waffeln, Bonbons, Pralinen etc., die in Lagern in verschiedenen Orten in bestimmten Mengen zu festgelegten Großhandelspreisen pro Stück vorgehalten werden. Als Ergebnis der Vertretertätigkeit entstehen Aufträge durch Kunden, die zu einer bestimmten Zeit über den Vertreter eine festgelegte Stückzahl eines Produktes bestellen. Modellieren Sie diese Miniwelt mit einem ER-Modell und verdeutlichen Sie die Komplexitäten durch je zwei knappe Formulierungen.



## 3. Datenmodelle und Datenbanksprachen

### 3.1. Das relationale Datenmodell

Will man etwas über relationale Datenbanken wissen, muss man sich mit dem relationalen Modell von Ted Codd auseinandersetzen. Das im Jahr 1970 entwickelte Modell hat in den Folgejahren wegen seiner überzeugenden Einfachheit und mathematischen Grundlagen sofort Aufmerksamkeit erregt.<sup>19</sup> Das Modell basiert auf dem Konzept einer mathematischen Relation, das in etwa wie eine Wertetabelle aussieht, und hat seine theoretischen Grundlagen in der Prädikatenlogik und der Mengenlehre.

Das Kapitel beschreibt die grundlegenden Merkmale des Modells und seine Einschränkungen und behandelt die relationale Algebra – eine Reihe von Operationen für das relationale Modell. Das relationale Modell wurde in den letzten 25 Jahren durch viele kommerzielle Systeme implementiert.

#### 3.1.1. Grundbegriffe

Das relationale Modell repräsentiert die Datenbank als Sammlung von Relationen. Informell ähnelt jede Relation einer Tabelle oder in gewissem Sinn einer flachen Datei. Allerdings bestehen zwischen Tabellen und Dateien große Unterschiede, wie später noch deutlich wird. Stellt man sich eine Relation als Tabelle vor, so repräsentiert jede Zeile eine Liste zusammenhängender Werte, ein sogenanntes Tupel. Dies entspricht einer konkreten Entitäts- oder Beziehungsausprägung aus dem ER-Modell. An den einzelnen Tabellenpositionen können jedoch nur atomare Attributwerte auftreten und nicht wiederum selbst Tabellen oder strukturierte Werte. Mehrwertige Attribute wie im ER-Modell sind also nicht möglich. Diese Einschränkung für Relationen nennt man **erste Normalform (1NF)**.

1NF

$R$		
$A_1$	$A_2$	$A_3$
	...	
	...	
	...	

Abbildung 20: Veranschaulichung eines Relationenschemas und einer Relation

#### Attribute, Wertebereiche, Relationsschema

Relationen zeichnen sich durch das Relationsschema, die Attribute sowie deren Wertebereiche aus.

<sup>19</sup>[EN09, S. 133]

WEINE				
W#	Name	Farbe	Jahrgang	Weingut
1042	La Rose Grand Cru	Rot	1998	Château La Rose
2168	Creek Shiraz	Rot	2003	Creek
3456	Zinfandel	Rot	2004	Helena
2171	Pinot Noir	Rot	2001	Creek
3478	Pinot Noir	Rot	1999	Helena
4711	Riesling Reserve	Weiß	1999	Müller
4961	Chardonnay	Weiß	2002	Bighorn

Abbildung 21: Eine Relation der Weindatenbank

$\mathcal{A}$	bezeichnet eine Menge von paarweise verschiedenen Attributnamen $A_1, \dots, A_n$ .
$dom(A)$	ist der jedem Attribut $A \in \mathcal{A}$ zugeordnete Wertebereich ( <i>Domain</i> ), vergleichbar einem Datentyp. Beispiele: Zeichenketten <i>string</i> , $\mathbb{N}$ , $\mathbb{Z}$ , usw. Werte der Wertebereiche sind atomar. Wertebereiche haben eine konkrete <i>Darstellung</i> , je nach DBS: <i>string</i> z. B. als <code>varchar(20)</code>
$R_{\mathcal{A}}$	ist eine <i>benannte</i> endliche Teilmenge von $\mathcal{A}$ und heißt <i>Relationsschema</i> mit Relationsname $R$ und Attributmenge $\mathcal{A}$ . (im Beispiel: <i>Weine</i> ( $W\#, Name, Farbe, Jahrgang, Weingut$ ))
$\mathcal{A}(R)$	ist die Attributmenge des jeweiligen Relationsschemas $R$ . (im Beispiel: $W\#, Name, Farbe, Jahrgang, Weingut$ )

Folgende Bemerkungen seien gemacht:

- Häufig besitzen verschiedene Wertebereiche die gleiche Darstellung (z.B. für `nat` und `nummer` die Darstellung `integer`).
- Je nach Kontext bezeichnet  $R$  den Namen des Relationsschemas, das Schema selbst oder eine Relation mit diesem Schema.
- Mit als atomar vorausgesetzten Wertebereichen ist die Relation  $R$  automatisch in der 1. Normalform (1NF).

## Tupel und Relationen

Gegeben sei ein Relationsschema  $R$  mit Attributmenge  $\mathcal{A}(R) = \{A_1, \dots, A_n\}$ .



<i>Tupel</i> $r$	ist eine Abbildung, die den Attributen konkrete Attributwerte zuordnet. Die möglichen Werte eines Tupels ergeben sich aus den Wertebereichen der Attribute: $r : \mathcal{A} \rightarrow \text{dom}(A_1) \times \text{dom}(A_2) \times \dots \times \text{dom}(A_n)$ mit $A_i \in \mathcal{A}, r(A_i) \in \text{dom}(A_i)$
<i>Relation</i> $R$	Menge von Tupeln $r$ . Vereinfachte Schreibweise als Kartesisches Produkt $R = \{r = (v_1, \dots, v_n) \mid v_1 \in \text{dom}(A_1), \dots, v_n \in \text{dom}(A_n)\}$

- Die Tupelmengen (Relationen) sind immer endlich. Sie besitzen definitionsgemäß keine Duplikate.
- Die Reihenfolge der Attribute in der Tupelschreibweise ist willkürlich festgelegt.
- Die Attributnamen von  $R$  sind paarweise verschieden. Ihre Wertebereiche und erst recht die Darstellung der Werte können gleich sein.
- Die Tupel  $r$  können *partiell* sein, d. h. nicht für jedes Attribut ist ein Wert definiert oder bekannt. Sie werden durch einen speziellen *Nullwert* (NULL), der Element jedes Wertebereichs ist, vervollständigt.
- Relationen werden als *zeitvariant* aufgefasst. Bei Entfernen oder Hinzufügen von Tupeln wird die entstehende Relation weiterhin mit  $R$  bezeichnet.

## Schlüssel

Eine besondere Bedeutung kommen Attributen oder Attributmengen zu, die eine Tupelmenge eindeutig kennzeichnen.

**Definition 3.1.1** (Schlüssel). *Eine Teilmenge*

$$K = \{A_1, \dots, A_k\} \subseteq \mathcal{A} = \mathcal{A}(R)$$

heißt *Schlüssel* von  $R$ , wenn  $r_1(K) = r_2(K) \Rightarrow r_1 = r_2$  gilt und  $K$  minimal ist.  $r_1(K) = r_2(K)$  steht abkürzend für  $r_1(A_1) = r_2(A_1), \dots, r_1(A_k) = r_2(A_k)$ .

*Bemerkungen:*

- Jede Relation besitzt einen Schlüssel, im Extremfall erfasst er alle Attribute.
- Die Attributmenge  $K$  wird nur dann Schlüssel genannt, wenn die Schlüsseleigenschaft *zeitinvariant* ist.

## Weitere Integritätsbedingungen

Beispiel: Buch(Inv#, Autor, Titel, Verlag) besitze den Schlüssel Inv#, d. h. alle Attributwerte sind eindeutig bestimmt, wenn eine Inv# gegeben ist. Zusätzlich kann aber noch gelten, dass Verlag durch Autor und Titel bestimmt ist (wenn gilt, dass keine Bücher mit gleichem Titel und Autoren gleichen Namens in verschiedenen Verlagen erscheinen).

Dies ist ein Beispiel für eine *semantische Integritätsbedingung*. Integritätsbedingungen (*constraints*) werden manchmal zum relationalen Schema hinzugenommen, dann heißt es *erweitertes Schema*.

### 3.1.2. Relationenalgebra: Operationen auf Relationen

Es werden Operationen betrachtet, die Relationen in Relationen abbilden. Man spricht deshalb von einer *relationalen Algebra*. Die relationale Algebra ist eine applikative Sprache: Die Anwendung einer z.B. zweistelligen Operation auf zwei Relationen liefert eine Relation. Im Folgenden werden die Basisoperationen betrachtet, auf deren Definition im nachfolgenden Abschnitt die Definition erweiterter Operationen beruht.

#### Kartesisches Produkt $\times$

Das kartesische Produkt  $R \times T$  besteht aus Paaren von Tupeln der Relationen  $R$  und  $T$ .

$$(v_1, \dots, v_n, w_1 \dots w_m) \in R \times T \Leftrightarrow (v_1, \dots, v_n) \in R \wedge (w_1, \dots, w_m) \in T$$

wobei  $\mathcal{A}(R) = \{A_1, \dots, A_n\}$  und  $\mathcal{A}(T) = \{B_1, \dots, B_m\}$ . Die Attributnamen müssen paarweise verschieden sein. Man vermeidet die Schreibweise mit Paaren von  $n$ - und  $m$ -Tupeln, und nennt es auch *erweitertes* kartesisches Produkt.<sup>20</sup>

Beispiel

$R$		$T$		$R \times T$			
A	B	C	D	A	B	C	D
1	2	1	2	1	2	1	2
1	3	2	3	1	2	2	3
				1	3	1	2
				1	3	2	3

#### Projektion $\pi$

Sei  $\mathcal{A} = \{A_1, \dots, A_n\} = \{A_i\}_{i \in I}$  wobei  $I = \{1, \dots, n\}$  und  $J$  sei eine Teilmenge der Indexmenge  $I$ ,  $|J| = m \leq n$ .  $X$  sei die Attributmenge  $\{A_j\}_{j \in J}$ ,  $X \subseteq \mathcal{A}$ .  $\pi_X$  bildet die Menge  $\mathcal{R}_{\mathcal{A}}$  aller Relationen, die über  $\mathcal{A}$  gebildet werden können, in die Menge  $\mathcal{R}_{\mathcal{A}}$  aller Relationen über  $\mathcal{A}$  ab:

$$\begin{aligned} \pi_X : \mathcal{R}_{\mathcal{A}} &\rightarrow \mathcal{R}_{\mathcal{A}} \\ \pi_X(R) &= \{(r_j)_{j \in J} \mid (r_i)_{i \in I} \in R, r_i = r_j \forall i, j \in J\} \end{aligned}$$

$r = (r_j)_{j \in J}$  für  $J \subseteq I$  wird auch mit  $r[X]$  bezeichnet und Beschränkung von  $r$  auf  $X$  genannt.

<sup>20</sup>Dass man bei Relationen von einem *erweiterten* kartesischen Produkt (und nicht einfach von einem kartesischen Produkt spricht, liegt daran, dass es bei Relationen nicht auf die Reihenfolge der Attribute ankommt, beim kartesischen Produkt aber sehr wohl (auf die der Komponenten). Das führt – wenn man es mathematisch sauber definieren will – zu aufwändigen Konstruktionen, in denen man Äquivalenzklassen von Relationen bildet, die zwar die gleiche Attributmenge besitzen, sich aber durch Permutation der Reihenfolge unterscheiden, und diese Äquivalenzklassen als Relation identifiziert.

**Beispiel**

$U$			$\pi_{B,C}(U)$	
A	B	C	B	C
1	a	2	a	2
2	a	2	c	1
3	c	1		

**Selektion  $\sigma$**

$A$  sei ein Attributbezeichner,  $w$  sei aus  $\text{dom}(A)$  oder selbst ein Attributbezeichner.  $\theta$  bezeichne einen Vergleichsoperator aus  $\{=, \neq, <, >, \geq, \leq\}$ .  $\sigma_{A\theta w}$  bildet Relationen über  $\mathcal{A}$  in Relationen über  $\mathcal{A}$  ab:

$$\begin{aligned}\sigma_{A\theta w} : \mathcal{R}_{\mathcal{A}} &\rightarrow \mathcal{R}_{\mathcal{A}} \\ \sigma_{A\theta w}(R) &= \{r \mid r \in R, r[A] \theta w'\} \\ w' &= \begin{cases} w, & w \in \text{dom}(A) \\ r[B], & B \in \mathcal{A}(R), \text{dom}(B) = \text{dom}(A) \end{cases}\end{aligned}$$

$\sigma$  liefert also die Teilmenge der Tupel von  $R$ , die der Bedingung  $r[A] \theta w'$  genügen. Es gilt:

$$\sigma_x(\sigma_y(R)) = \sigma_y(\sigma_x(R))$$

Man führt folgende Schreibweisen ein:

$$\begin{aligned}\sigma_{c_1 \vee c_2}(R) &:= \sigma_{c_1}(R) \cup \sigma_{c_2}(R) \\ \sigma_{c_1 \wedge c_2}(R) &:= \sigma_{c_1}(\sigma_{c_2}(R)) \\ \sigma_{\neg c}(R) &:= R \setminus \sigma_c(R)\end{aligned}$$

wobei  $c, c_1, c_2$  elementare Vergleichsterme der Form  $A \theta w$  sind.

Damit kann ein beliebiger Boole'scher Ausdruck  $F$ , der aus elementaren Vergleichen mit den logischen Operatoren gebildet wird, als Selektions-Bedingung dienen.

**Beispiel**

$U$			$\sigma_{A=2 \vee B=c}(U)$		
A	B	C	A	B	C
1	a	2			
2	a	2	2	a	2
3	c	1	3	c	1

**Der Umbenennungsoperator**

Der Umbenennungsoperator  $\rho_{S(A_1, \dots, A_n)}(R)$  bindet  $R$  an den neuen Relationsnamen  $S$  und benennt die Attribute in der angegebenen Reihenfolge in  $A_1, \dots, A_n$  um. Soll nur der Relationsname geändert werden, reicht  $\rho_S(R)$ . Diese Form wird auch häufig  $S \leftarrow R$  geschrieben und dazu benutzt, bei komplizierteren Ausdrücken benannte Zwischenrelationen zu erzeugen. Die Menge der Tupel, d. h. die Relation bleibt erhalten, die Umbenennung trifft nur das Schema.

## Mengenoperationen für vereinigungskompatible Relationen

Da eine Relation als Menge (von Tupeln) aufgefasst werden kann, sind neben den datenbank-typischen Operationen wie Selektion und Projektion auch Mengenoperationen wie Vereinigung und Differenz möglich, die für eine Algebra auch benötigt werden. Mit ihrer Hilfe lassen sich später (siehe Abschnitt 3.1.3) auch abgeleitete relationale Operationen definieren. Dies ist aber nur unter bestimmten Bedingungen möglich.

**Definition 3.1.2** (Vereinigungskompatibilität).  $R$  und  $T$  mit  $\mathcal{A}(R) = \{A_1, \dots, A_n\}$  und  $\mathcal{A}(T) = \{B_1, \dots, B_n\}$  heißen *vereinigungskompatibel*, wenn  $\text{dom}(A_i) = \text{dom}(B_i)$  für  $i = 1, \dots, n$ ; d. h. die Attributnamen müssen nicht übereinstimmen, die Wertebereiche schon.

Für vereinigungskompatible Relationen sind die Mengenoperationen  $\cup$  und  $\setminus$  definiert, wie das folgende Beispiel zeigt:

Beispiel	$R$		$T$		$R \cup T$		$R \setminus T$	
	A	B	C	D	R.A	R.B	R.A	R.B
	1	2	1	2	1	2	1	2
	1	3	2	3	1	3	1	3
					2	3		

### 3.1.3. Abgeleitete relationale Operationen

Mit den obigen Operatoren lassen sich weitere definieren.

#### Mengenschnitt $\cap$

Für vereinigungskompatible Relationen  $R$  und  $T$  gilt

$$R \cap T = R \setminus (R \setminus T)$$

Mit anderen Worten: Man kann den Schnitt auf die Differenz zurückführen.

#### Verbund $\bowtie$

Die wahrscheinlich wichtigste Operation auf Relationen ist der Verbund zweier Mengen. Wir werden später noch sehen, dass es verschiedene Varianten des Verbundes gibt, die mit der Möglichkeit der Existenz von Nullwerten Null zusammenhängen. Die Existenz des Verbundoperators zeichnet 'echte' relationale Datenbanksysteme aus.

**Definition 3.1.3** (Verbund). Seien  $R, T$  seien Relationen mit  $\mathcal{A}(R) = X, \mathcal{A}(T) = Y$ .

$$R \bowtie_{A\theta B} T := \sigma_{A\theta B}(R \times T), \quad A \in X, B \in Y$$

heißt der  $\theta$ -Verbund ( $\theta$ -Join) von  $R$  und  $T$ .  $\theta$  ist wie oben bei der Selektion als Vergleichsoperator definiert.

Ist  $\theta$  die Gleichheitsrelation, so spricht man vom Gleichverbund. Eine mögliche Verallgemeinerung auf beliebige Boole'sche Verbundbedingungen wird selten benutzt.

Wichtig ist der Spezialfall des *natürlichen Verbunds*  $\bowtie$ . Das ist ein Gleichverbund über allen gemeinsamen Attributen der Operanden  $U$  und  $V$  mit anschließender Projektion in der Weise, dass doppelte wegfallen:

Spezialfall  $\bowtie$

$$U \bowtie V = \pi_{X \cup Y}(\sigma_c(U \times V))$$

mit  $\mathcal{A}(U) = X$ ,  $\mathcal{A}(V) = Y$ ,  $X \cap Y = \{A_1, \dots, A_k\}$ ,  $c = R.A_1 = T.A_1 \wedge \dots \wedge R.A_k = T.A_k$ .  
Attributnamen sind hier in üblicher Weise durch den Relationsbezeichner qualifiziert.

**Beispiel**

U			V			U $\bowtie$ V			
A	B	C	A	C	D	U.A	U.B	U.C	V.D
1	a	2	1	3	a	1	a	2	c
2	a	2	2	2	b	2	a	2	b
3	c	1	1	2	c				

Manchmal wird der Verbund als Basisoperator gewählt.  $\sigma$  kann dann mit Hilfe von  $\bowtie$  definiert werden.

**Division  $\div$**

$X, Y$  seien Relationen,  $\mathcal{A}(Y) = P$  sei enthalten in  $\mathcal{A}(X) = Q$ . O.b.d.A. sollen die Attribute von  $Y$  die hinteren von  $X$  sein, also sei etwa  $(A_1, \dots, A_k, B_1, \dots, B_j)$  die (willkürlich fixierte) Folge der Attribute von  $X$ .

Gesucht sind die Tupel von  $r[A_1, \dots, A_k]$ ,  $r \in X$ , für die gilt:

$$r[A_1, \dots, A_k] \times Y \subseteq X$$

**Beispiel**

X		
C	A	B
a	1	1
b	1	1
a	1	2
b	1	2
b	2	2

Y	
A	B
1	1
1	2
2	2

$X \div Y$
C
b

(b) ist nämlich das einzige Tupel für das gilt: die Konkatenation mit allen Tupeln von  $Y$  ist enthalten in  $X$ .

Mittels Projektion, Differenz und kartesischem Produkt lässt sich die Division von  $X$  durch  $Y$  folgendermaßen ausdrücken:

$$X \div Y = \pi_{P \setminus Q}(X) \setminus \pi_{P \setminus Q}((\pi_{P \setminus Q}(X) \times Y) \setminus X)$$

Allgemein gilt  $\mathcal{A}(X \div Y) = \mathcal{A}(X) \setminus \mathcal{A}(Y)$ , mit den hier gewählten Bezeichnungen  $P = \mathcal{A}(X)$  und  $Q = \mathcal{A}(Y)$  also  $\mathcal{A}(X \div Y) = P \setminus Q$ .

## Syntax relationenalgebraischer Ausdrücke

In Analogie zu den regulären Ausdrücken werden auch relationale Ausdrücke rekursiv definiert. So kann man sicher stellen, dass alle relationalen Ausdrücke aus einer Grundmenge gebildet werden können.<sup>21</sup>

- (a) Bezeichner  $R, S, \dots$  von Relationen sind relationenalgebraische Ausdrücke.
- (b) Ist  $E_1$  ein relationenalgebraischer Ausdruck, so sind auch die einstelligen relationalen Operationsanwendungen

- (a)  $\pi_X(E_1)$  (Projektion)
- (b)  $\sigma_c(E_1)$  (Selektion)
- (c)  $\rho_{S(X)}(E_1)$  (Umbenennung)

mit  $X \subseteq \mathcal{A}(E_1)$  relationenalgebraische Ausdrücke.

- (c) Sind  $E_1, E_2$  relationenalgebraische Ausdrücke, so sind auch die zweistelligen (binären) relationalen Infix-Operationsanwendungen

- (d)  $E_1 \circ E_2$  (Verkettung)
- (e)  $E_1 \bowtie E_2$  (natürlicher Verbund)
- (f)  $E_1 \bowtie_{A\theta B} E_2$  (Theta-Verbund)
- (g)  $E_1 \cup E_2$  (Vereinigung)
- (h)  $E_1 \cap E_2$  (Schnitt)
- (i)  $E_1 \setminus E_2$  (Differenz)
- (j)  $E_1 \div E_2$  (Division)

mit  $A, B \in \mathcal{A}(E_1) \cup \mathcal{A}(E_2)$  relationenalgebraische Ausdrücke.

- (d) Das sind alle relationenalgebraischen Ausdrücke.
- (e) Die Ablage von Zwischenergebnissen ist üblich, um überlange Ausdrücke zu verhindern. I. d. R. werden sie in der Form  $Q_i \leftarrow E_j$  dargestellt. (Z. B.  $Q_1 \leftarrow \pi_C(R)$ ) Mit diesem Zwischenergebnis kann weitergearbeitet werden, ohne den ursprünglichen Ausdruck verwenden zu müssen. (Z. B.  $\sigma_{C=a}(Q_1)$ )

## Semantik relationenalgebraischer Ausdrücke

Im Hinblick auf die effiziente Auswertung relationaler Ausdrücke ist es notwendig, die Semantik eines relationalen Ausdrucks zu definieren. Nur so können zwei unterschiedliche relationale Ausdrücke auf ihre Äquivalenz hinsichtlich des Auswertungsergebnisses verglichen werden.

Sei  $\mu$  eine einstellige und  $\tau$  eine zweistellige relationale Operation,  $rA, rA_1, rA_2 \in RA$ . Dann definiert man:

$$\begin{aligned} \text{eval}(R) &= R \\ \text{eval}(\mu(rA)) &= \mu(\text{eval}(rA)) \\ \text{eval}(rA_1 \tau rA_2) &= \text{eval}(rA_1) \tau \text{eval}(rA_2) \end{aligned}$$

<sup>21</sup>Siehe Vorlesung zur *Theoretischen Informatik*.

ACHTUNG: In der ersten Zeile ist das Argument  $R$  ein Relationsbezeichner, das Ergebnis dagegen eine Relation.

**Definition 3.1.4** (Äquivalenz relationaler Ausdrücke). *Zwei relationenalgebraische Ausdrücke  $r_{A_1}$ ,  $r_{A_2}$  heißen äquivalent  $\Leftrightarrow \text{eval}(r_{A_1}) = \text{eval}(r_{A_2})$ .*

Ziel der *Optimierung* relationenalgebraischer Ausdrücke ist es, einen jeweils äquivalenten Ausdruck zu finden, dessen Auswertungsaufwand möglichst gering ist.

### 3.2. Software für Relationenalgebra im Unterricht

Um im Unterricht das Themenfeld Datenbanken zu lehren (und es zu lernen!), ist eine unterstützende Software sehr hilfreich. Gerade das recht theoretische Konzept der relationalen Algebra profitiert von der Möglichkeit, Durchdachtes auch ausprobieren zu können.<sup>22</sup>

Bis vor Kurzem gab es keine Software, die alle in der Lehrerweiterbildung angesprochenen Sprachkonzepte abgedeckt hat.<sup>23</sup> Bis 2011 wurde das relationale Sprachkonzept durch ein von Peter Bartke in Miranda implementierter RALG-Interpreter, Domänen- (DRC) und Tupelkalkül (TRC) mit WinRDBI und die logische Sprache Datalog mit Prolog abgedeckt.<sup>24</sup> Ab 2012 konnten mit der Software DES (*Datalog Educational System*) der Universität Madrid dann immerhin RALG, und Datalog unter einem Softwaredach zusammengefasst werden. Seit der 2016 erschienenen Versionsnummer 4.1 von DES ist es sogar möglich, alle fünf behandelten Konzepte in einer einheitlichen Umgebung zu verwenden:

Relationenalgebra		Datalog	DRC	TRC	SQL
bis 2011	RALG	Prolog	WinRDBI	WinRDBI	postgreSQL
2012-2015	DES	DES	WinRDBI	WinRDBI	DES/postgreSQL
ab 2016	DES	DES	DES	DES	DES/postgreSQL
ab 2021	Onlineversion unter <a href="http://desweb.fdi.ucm.es">desweb.fdi.ucm.es</a>				

Ein besonderer Vorteil der einheitlichen Softwareumgebung ist die Möglichkeit, einheitliche Datenbestände zu verwenden und an diese in den verschiedenen Sprachkonzepten Anfragen zu stellen. DES verfügt über eine SQL-artige Daten-Definitionssprache, die für alle Sprachkonzepte genutzt werden kann. Dies erleichtert die Pflege der Datenbestände ungemessen. Bis auf einige fehlende Datentypen ist diese Definitionssprache sogar kompatibel zum professionellen SQL-Server PostgreSQL, so dass man prinzipiell die Daten auch für ein professionelles Produktivsystem benutzen könnte.

<sup>22</sup>Dies gilt mindestens genauso für andere, ebenfalls in der Lehrerweiterbildung betrachteten, Sprachkonzepte wie das Domänen- oder das Tupelkalkül und auch logische Anfragesprachen wie Datalog.

<sup>23</sup>Zumindest ist das der Kenntnisstand eines der Autoren dieses Skriptes (Oliver Schäfer).

<sup>24</sup>Zudem hat der Relationenalgebra-Interpreter den Nachteil, dass er nur unter Linux-artigen Betriebssystemen lauffähig ist und – zumindestens beim Lehrenden – ein gehöriges Maß an Miranda-Kenntnissen erfordert.

Aus den genannten Gründen eignet sich die frei im Internet und für alle großen Plattformen (Windows, Linux/UNIX oder Mac) erhältliche Software DES (*Datalog Educational System*) sehr gut für Lehrzwecke, sowohl an der FU als auch in der Schule. Die Software wird zum freien Download angeboten, eine umfangreiche Anleitung – leider nur in Englisch – gibt es ebenfalls.<sup>25</sup> Ist zudem JAVA auf dem Rechner installiert, gibt es die (nicht unbedingt nötige) Entwicklungsumgebung ACIDE in der Version 0.17 dazu.

### 3.2.1. Installation und erste Schritte

In der LWB-Distribution ist DES mit allen Sprachkonzepten enthalten. Die jeweils aktuellste Version kann man von o. g. Quelle herunterladen und in das Verzeichnis der eigenen Version hinein entpacken (ein 'which des' hilft bei der Frage, wo die Dateien liegen).

Man startet die Nutzung durch einen Aufruf von `des` an der Kommandozeile bzw. Starten der GUI-Version über das Menu oder den Aufruf von `acide` in der Kommandozeile <sup>26</sup> (siehe Abbildung 22).

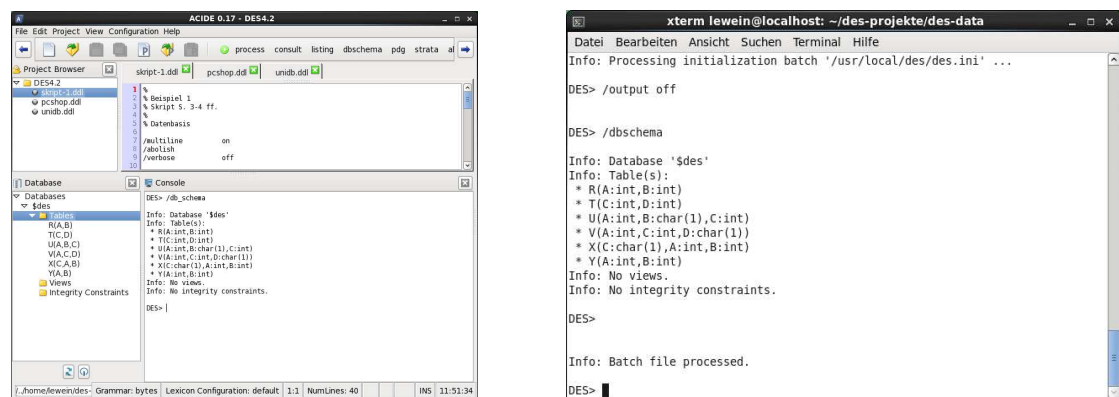


Abbildung 22: ACIDE (links), bereits mit neuen Projektdateien eingerichtet sowie Kommandozeilenversion (rechts), die nach Installation des Archivs von der Materialiensite bereits mit dem Kommando `des skript-1.ddl` gestartet wurde, so dass bereits die Datendefinition aus der Datei `skript-1.ddl` eingelesen wurde.

Hat alles geklappt, muss das System mit Daten gefüttert werden. Das kann durch zeilenweises Eingeben in der DES-Konsole geschehen oder deutlich komfortabler durch Einlesen einer Daten-Definitions-Datei (ddl-Datei). In der GUI-Version geschieht dies durch Öffnen einer geeigneten Datei und Ausführen eines Rechtsklicks im Dateifenster, gefolgt von der Auswahl 'Send File Content to Console'. Die GUI-Version liefert Syntax-Highlighting nur bei ihr bekannten Datei-Endungen (z. B. `.ra` oder `.sql`). In der Konsolenversion wird eine solche Datei durch Eingabe des Meta-Befehls `/p <Dateiname>` erledigt.<sup>27</sup> In der Konsole orientiert sich DES von seinem eigenen Ordner heraus. Wenn man eine Datei über die Konsole

<sup>25</sup>Die aktuelle Version findet man unter [des.sourceforge.net](http://des.sourceforge.net), derzeit in der Version 6.5 vom April 2020.

<sup>26</sup>Hierfür sollte man sich im des-Verzeichnis befinden, da dann erst die gespeicherten Datenbanken eingelesen werden.

<sup>27</sup>Hat man die Version von der Materialiensite installiert, wurde ein Wrapper-Skript `des` mitinstalliert, das die Übergabe eines Dateinamens an den Aufruf der Konsolenversion erlaubt.



einlesen möchte, sollte man dies berücksichtigen. Anschließend können relationalalgebraische Ausdrücke eingegeben werden, die auf der Basis der eingelesenen Daten ausgewertet werden. Stehen bereits die Daten aus der Definitionsdatei `beispiel.ddl` zur Verfügung, können beispielsweise folgende Ausdrücke ausgewertet werden:<sup>28</sup>

Beschreibung	Eingabe in DES
Kartesisches Produkt	<code>R product T;</code>
Projektion	<code>project B,C (U);</code>
Selektion	<code>select (A=2 or B='c') (U);</code>
Umbenennung	<code>rename S(M,N) (R);</code>
Vereinigung	<code>R union T;</code>
Differenz	<code>R difference T;</code>
Schnitt	<code>R intersect T;</code>

Nachfolgend sind noch einige wichtige Meta-Kommandos aufgeführt, die längst nicht vollständig ist. Für weitere Informationen kann man `\help [<Stichwort>]` an der DES-Kommandozeile eingeben. DES kommt aber auch mit einer sehr umfangreichen Anleitung daher, die allerdings nur in Englisch vorliegt.

Meta-Kommando	Bedeutung
<code>/q</code>	beendet DES
<code>/dbschema</code>	listet alle Schemas auf
<code>/listing</code>	listet alle Tupel aller Relationen auf
<code>/listing &lt;R&gt;</code>	listet alle Tupel eines bestimmten Relation auf
<code>/p &lt;Datei&gt;</code>	Interpretiert Dateiinhalt als Konsoleneingabe
<code>/ra</code>	wechselt in den Relationenalgebra-Modus
<code>/drc</code>	wechselt in den Domänenkalkül-Modus
<code>/trc</code>	wechselt in den Tupelkalkül-Modus
<code>/datalog</code>	wechselt in den Datalog-Modus (Standard)
<code>/prolog</code>	wechselt in den Prolog-Modus
<code>/sql</code>	wechselt in den SQL-Modus
<code>/log &lt;Datei&gt;</code>	Schreibt Ein- und Ausgaben zusätzlich in eine Datei
<code>/nolog</code>	Beendet das Protokollieren der Ein- und Ausgaben

<sup>28</sup>Eingaben müssen immer mit einem Semikolon abgeschlossen werden, wenn die Option `multiline on` gesetzt ist.

DES kennt viele Sprachkonzepte, allerdings ist es in der Regel nicht nötig, den Interpreter durch Eingabe eines Meta-Kommandos in den entsprechenden Sprachmodus zu schalten. Meist erkennt der Interpreter an der Eingabe, welches Sprachkonzept gerade gefragt ist. Sollten hier entsprechende Fehlermeldungen erscheinen, verschwinden diese nach expliziter Anwahl z. B. von `/ra`. Es kann aber durch ein Voranstellen von `/ra` erreicht werden, dass nur der nachfolgend angegebene Ausdruck innerhalb des angegebenen Sprachkonzepts interpretiert wird.

### 3.3. Übungszettel A-4

**Aufgabe 4.1.** *In dieser Aufgabe wird ein Ausschnitt einer Bibliotheksanwendung modelliert. Der Umfang ist durch die folgende Beschreibung umrissen.*

*Verlage sind durch einen Namen gegeben und werden durch ihn eindeutig identifiziert. Sie sind an einem Ort angesiedelt. Autoren von Büchern sind durch ihren Namen nicht eindeutig bestimmbar, deshalb werden sie durch eine Nummer identifiziert. Herausgeber von Büchern werden mit zu den Autoren gerechnet. Die Buchtitel sind Produkte eines Verlags und werden über die ISB-Nummer eindeutig identifiziert. Jedes Buch ist in genau einem Verlag in einem bestimmten Jahr zu einem Festpreis verlegt worden. An einem Buch können mehrere Herausgeber oder Autoren beteiligt sein, was durch die Rollen Autor oder Herausgeber gekennzeichnet wird. Die Stellung eines Autors/Herausgebers innerhalb der Autoren- bzw. Herausgeberliste wird durch einen numerischen Rang erfasst. In der Bibliothek befinden sich die (physisch greifbaren) Exemplare von Büchern. Sie sind mit Exemplarnummern durchnummeriert, die für jedes Buch bei eins beginnen. Der Standort eines Buchexemplars innerhalb der Bibliothek ist durch eine Regalnummer gegeben. Lesernamen sind wie Autorennamen nicht eindeutig, die Leser werden deshalb durch eine Lesernummer erfasst. Zusätzlich benötigt man alle für ein Mahnschreiben nötigen Angaben. Ein Ausleihvorgang erfasst Leser und die von ihr/ihm zu einem Zeitpunkt ausgeliehenen Buchexemplare.*

*Geben Sie hierzu ein ER-Modell mit (min,max)-Komplexitäten an.*

**Aufgabe 4.2.** *Ein Ausschnitt einer Bibliotheksverwaltung ist durch den folgenden beschreibenden Text gegeben (Abweichungen von Aufgabe 1 sind natürlich möglich):*

- *Es gibt Verlage, die einen Namen und einen Hauptverlagsort haben.*
- *Es gibt Autoren, die mit ihrem Namen, Vornamen und einer Autorennummer verwaltet werden.*
- *Bücher sind durch ihre ISBN-Nummer charakterisiert und haben Titel, Verlag, Jahr und Preis als Attribute.*
- *Autoren spielen in den Büchern verschiedene Rollen: sie sind Herausgeber oder Autor, Erst-, Zweit-, ...-Herausgeber oder -Autor.*
- *Die Buchexemplare stehen in einem numerierten Regal.*
- *Lesern werden Lesernummern vergeben, Name, Ort und Anschrift werden registriert.*
- *Für die Ausleihvorgänge wird das Ausleihdatum gespeichert.*

*Gesucht ist ein logisches Modell in einem Relationenschema.*

- Geben Sie für dieses Modell die Menge der Attribute  $\mathcal{A}$  und ihre abstrakten Wertebereiche an.*
- Notieren sie das Relationenschema mit Angabe der Schlüsselattribute.*
- Geben Sie zusätzliche Annahmen zur Semantik des Schemas durch kurze beschreibende Sätze an.*

**Aufgabe 4.3.** *Geben Sie für das Relationenschema Bibliothek aus Aufgabe 2 die nachfolgenden Anfragen als relationenalgebraische Ausdrücke an:*

- (a) *An welchem Ort verlegt der Verlag Suhrkamp?*
- (b) *Welche Anschriften haben die Teltower Leser?*
- (c) *Welche Bücher (ISBN), die in der Bibliothek vorhanden sind, wurden im Springer-Verlag verlegt?*
- (d) *Ausgabe der Liste der Bücher mit Titel, ISBN, Verlagsort.*
- (e) *Welche Leser haben ein Buch mit dem Titel ‘Datenbanken’ ausgeliehen?*
- (f) *Welche Autoren (Name, Vorname) waren nur als Herausgeber tätig, jedoch nie eigentlich als Autor?*
- (g) *Welche Leser wohnen an dem Ort, an dem der Springer-Verlag seine Bücher veröffentlicht?*
- (h) *Welche Leser (LNr, Name) haben derzeit kein Buch ausgeliehen?*
- (i) *Welche Leser (LNr, Name) haben das Buch ‘Datenbanken und ich’ mindestens zweimal ausgeliehen? Es gebe das Buch in mehreren Exemplaren.*

**Aufgabe 4.4.** *Gegeben ist ein Relationenschema mit vier Relationen,*

Hersteller(Name,Modell,Art)  
PC(Modell,GHz,RAM,GB,Preis)  
Notebook(Modell,GHz,RAM,GB,ResH,ResV,Preis)  
Drucker(Modell,Farbe,Typ,Preis)

*Die Hersteller-Relation führt den Herstellernamen, die Modellnummer eines Produktes und die Art (PC, Notebook oder Drucker) auf. Die Modellnummern seien über alle Hersteller verschieden. In der PC-Relation wird die Modellnummer, Prozessorfrequenz, Hauptspeicherausbau Festplattengröße und Preis des Modells gehalten. Ähnlich ist es in der Notebook-Relation. Dort wird die Bildschirmgröße in horizontalen und vertikalen Pixeln mitgeführt. Beim Drucker wird die Farbfähigkeit (True oder False) und der Typ (Tinte oder Laser) vermerkt.*

- (a) *Überlegen Sie sinnvolle Schlüsselattribute für die Relationen und begründen Sie Ihre Wahl, indem Sie plausible Annahmen über die beteiligten Schlüssel treffen.*
- (b) *Geben Sie relationenalgebraische Ausdrücke für die folgenden Anfragen an:*
  - (i) *Gesucht sind alle Hersteller, die Drucker verkaufen, aber keine PCs.*
  - (ii) *Welche PC-Modelle sind mindestens 2.5 GHz schnell?*
  - (iii) *Welche Hersteller fertigen Notebooks mit Festplatten von mindestens 200 GB?*
  - (iv) *Gesucht sind Modellnummer und Preise aller Produkte (beliebiger Art) von Hersteller D.*
  - (v) *Gesucht sind alle Modellnummern von Schwarzweiß-Laserdruckern.*
  - (vi) *Finde alle Hersteller von wenigstens zwei verschiedenen Rechnern (PCs oder Notebooks) mit Prozessorfrequenzen von wenigstens 2 GHz.*

**Aufgabe 4.5.** Zum Relationenmodell aus Aufgabe 4.4 sind Daten gegeben. Berechnen Sie für die Anfragen aus Aufgabe 4.4 die Ergebnismengen.

HERSTELLER			PC					
Name	Modell	Art	Modell	GHz	Cores	RAM	GB	Preis
A	1001	PC	1001	2.66	2	2048	400	2114
A	1002	PC	1002	2.10	1	1024	250	998
A	1003	PC	1003	1.6	2	1024	120	478
A	2004	Notebook	1004	2.80	1	2048	250	649
A	2005	Notebook	1005	3.20	1	2048	300	689
A	2006	Notebook	1006	3.20	2	2048	400	1049
B	1004	PC	1007	2.20	1	1024	200	530
B	1005	PC	1008	2.20	2	2048	240	789
B	1006	PC	1009	2.00	2	1024	280	699
B	2007	Notebook	1010	2.80	2	2048	270	729
C	1007	PC	1011	1.86	1	2048	160	599
D	1008	PC	1012	2.80	2	1024	160	649
D	1009	PC	1013	3.06	2	512	80	499
D	1010	PC	NOTEBOOK					
D	3004	Drucker	Modell	GHz	RAM	GB	ResH	ResV
D	3005	Drucker	2001	2.00	4096	240	1600	1200
E	1011	PC	2002	1.73	2048	120	1680	1024
E	1012	PC	2003	1.80	1024	120	1280	800
E	1013	PC	2004	2.00	1024	160	1280	800
E	2001	Notebook	2005	2.16	2048	120	1024	768
E	2002	Notebook	2006	2.00	4096	200	1440	900
E	2003	Notebook	2007	1.83	2048	160	1024	600
E	3001	Drucker	2008	1.60	2048	120	1024	576
E	3002	Drucker	2009	1.60	1024	80	1024	600
E	3003	Drucker	2010	2.00	2048	240	1280	1024
F	2008	Notebook	DRUCKER					
F	2009	Notebook	Modell	Farbe	Art	Preis		
G	2010	Notebook	3001	True	Tinte	99		
H	3006	Drucker	3002	False	Laser	299		
H	3007	Drucker	3003	True	Laser	899		
			3004	True	Tinte	139		
			3005	False	Laser	199		
			3006	True	Tinte	89		
			3007	True	Laser	399		

### 3.4. Vom ER- zum Relationenmodell

Dem konzeptuellen Datenbankentwurf ist ein genügend großer Raum einzuräumen. Steht am Ende dieser Phase ein tragfähiges Modell zur Verfügung, muss dieses in ein logisches (Relationen-)Modell überführt werden. Dabei gibt es einige Dinge zu berücksichtigen. Während Entitäten aus dem ER-Modell quasi 1 : 1 in Relationen des Relationenmodells übersetzt werden können, hängt die Transformation der Beziehungstypen des ER-Modells empfindlich von den Kardinalitäten der beteiligten Entitäten ab. In einigen Fällen treten die Relationentypen im Relationenmodell gar nicht mehr auf.

Nachfolgend ist ein mehrschrittiges, quasi algorithmisches Verfahren angegeben, das man in einer ähnlichen Form z. B. in [EN09, S. 236 ff.] findet. Die Reihenfolge der Transformationsschritte ist nicht festgelegt, jedoch sollten zuerst die Entitätstransformationen durchgeführt werden, weil sie in manchen Fällen weitere Attribute durch die transformierten Beziehungstypen aufnehmen müssen. Die einzige Ausnahme bilden dabei die Entitäten, die an einer 1 : 1-Beziehung teilnehmen (Begründung siehe unten).

**Transformation von Entitätstypen:** Ein Entitätstyp des ER-Modells mit der Attributmenge  $A$  wird in einen Relationstyp des Relationenmodells mit der (zunächst noch) gleichen Attributmenge transformiert. Zusammengesetzte Attribute werden verflacht, treten also gleichberechtigt und nebeneinander in einem Tupel auf. Hier kann es bereits zu einer Veränderung der Attributmenge kommen.

Um mehrwertige Attribute unterzubringen, wird ein neuer Relationstyp erzeugt, der als Fremdschlüssel den Schlüssel  $Key$  der Entität enthält, zu dem das mehrwertige Attribut gehört und jeden Wert dieses Attributes in einem Tupel ( $Key, \text{Attributwert}$ ) speichert, beispielsweise zum mehrwertigen Attribut  $Päkos$  einer Schule die Relation  $Päkos(\underline{SchulNr}, \underline{PersNr})$ . In der ursprünglichen Relation  $Schule$  ist das Attribut  $Päkos$  nicht mehr enthalten.

Handelt es sich um eine schwache Entität, nimmt man den Primärschlüssel der zugehörigen starken Entität hinzu. Die Kombination aus diesem Primärschlüssel und dem partiellen Schlüssel der schwachen Entität bildet den identifizierenden Schlüssel des neuen Relationstypen.

**Transformation von c:c-Beziehungstypen:** Beziehungstypen der Art c:c werden in einen eigenen Relationstyp transformiert, der die Schlüssel der beteiligten Entitäten als Fremdschlüssel sowie alle Attribute des Original-Beziehungstyps aufnimmt. Als Schlüssel dient einer der beiden Fremdschlüssel. (s. Abb. 23)

**Transformation von 1:n-Beziehungstypen:** Bei der Transformation von 1:n-Beziehungstypen werden die ER-Relationsattribute zusammen mit dem Fremdschlüssel der Entität der 1-Seite der transformierten Entität der  $n$ -Seite hinzugefügt. Dabei wird oft nicht unterschieden, ob es sich um optionale oder obligatorische (s. Abb. 26) Beziehungen handelt.

Der Unterschied ist aber, dass bei *optionaler*  $n$ -Seite (s. Abb. 24) Nullwerte auftreten können. Im abgebildeten Beispiel wäre das z. B. der Fall bei Kursen, die in keinem Raum, sondern an anderer Stelle stattfinden. Als Schlüssel dient der jeweilige Primärschlüssel des Entitätstyps. Wenn Nullwerte verhindert werden sollen, wird eine eigene Relation  $Beherbergung(\underline{R\#}, \underline{K\#}, \text{Zeit})$  erstellt (s. Abb. 25).

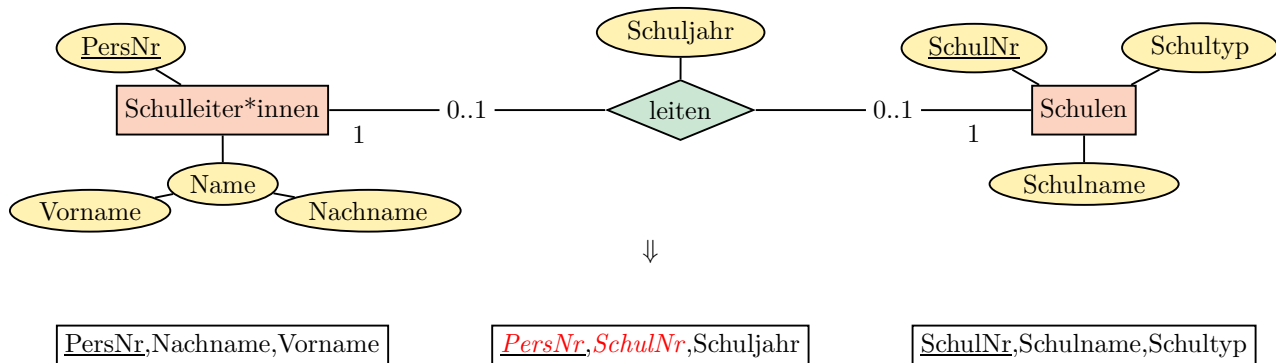


Abbildung 23: Beispiel der Transformation eines c:c-Beziehungstyps

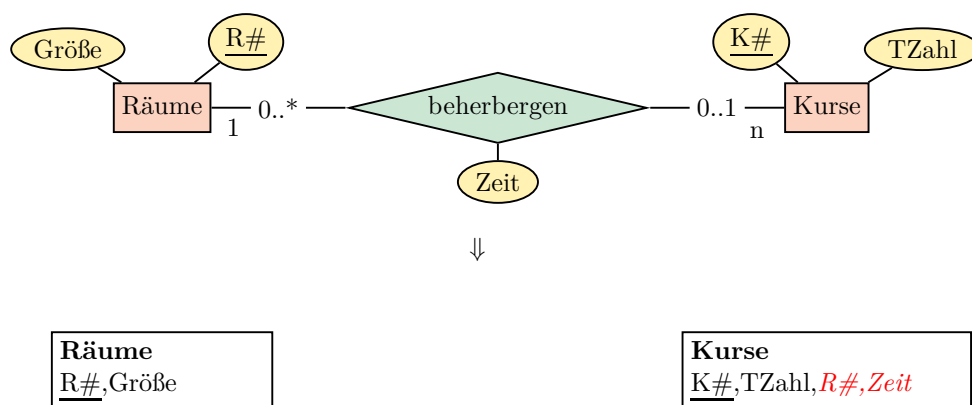


Abbildung 24: Beispiel der Transformation eines c:nc-Beziehungstyps mit Nullwerten

Diese Transformation ist mit Semantikverlust behaftet. Zum einen können optionale Beziehungen von obligatorischen nicht anhand der erzeugten Schemata unterschieden werden, zum anderen gehen (natürlich) damit auch die Komplexitäten der beteiligten Entitäten verloren. Dies lässt sich nur durch Hinzufügen geeigneter Integritätsbedingungen (constraints) auf den Tupelmengen verhindern.

**Transformation von m:n-Beziehungstypen:** Auch im Fall der Transformation von m:n-Beziehungstypen tritt ein Semantikverlust auf, da auch hier der Typ *n:m obligatorisch* (s. Abb. 28) vom Typ *m:n optional* (s. Abb. 27) nicht unterschieden werden kann. Anders als im Fall der 1:n-Transformation taucht hier der ursprüngliche Beziehungstyp im Relationenmodell als eigenständige Relation auf und enthält als Attribute sowohl die Schlüssel der beteiligten Entitäten als Fremdschlüssel als auch die Attribute des alten Beziehungstyps selbst. Als Schlüssel wird die identifizierende Kombination der beteiligten Fremdschlüssel verwendet.

**Transformation von 1:1-Beziehungstypen:** Da die beteiligten Entitätstypen einer 1:1-Beziehung existenziell voneinander abhängen, bietet es sich an, zwecks Vermeidung von Redundanzen die beiden Entitätsrelationen zu einer zusammenzufassen. Als Primärschlüssel kann man prinzipiell einen der beiden Primärschlüssel verwenden, oder – was häufig gemacht wird, um keinen Entitätstypen stärker zu betonen – einen neuen Schlüssel generieren. (s. Abb. 29)

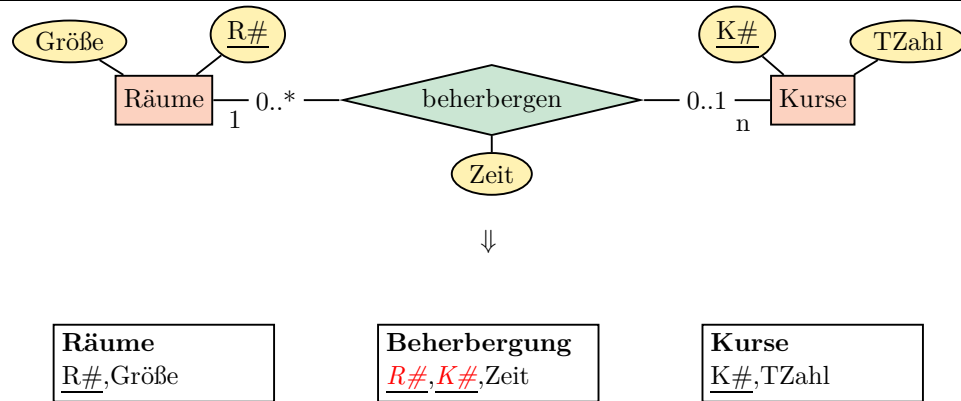


Abbildung 25: Beispiel der Transformation eines c:nc-Beziehungstyps ohne Nullwerte

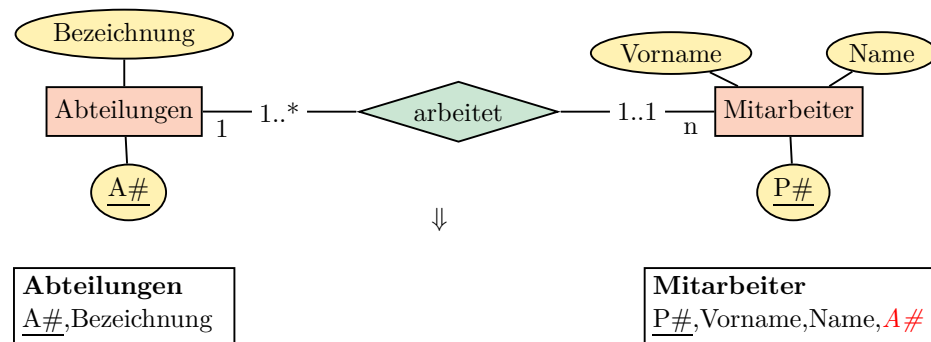


Abbildung 26: Beispiel der Transformation eines 1:n-Beziehungstyps

Handelt es sich bei der 1:1-Beziehung um einen einseitig optionalen Fall, bringt die Zusammenfassung zu einer Relation das Problem mit sich, dass Nullwerte auftreten können, allerdings wird (im Gegenzug) eine Relation eingespart.

**Transformation von Subtypenbeziehungen:** Subtypenbeziehungen kann man durch Fremdschlüsselhinzunahme modellieren. Jeder Subtyp wird zu einem neuen Relationsschema, das alle Attribute des jeweiligen Subtyps und den Primärschlüssel des Supertyps als Fremdschlüssel enthält. Dieser Fremdschlüssel wird zum Primärschlüssel des Subtyps. (s. Abb. 31)

**Transformation von  $n$ -ären Beziehungen:** Die Transformation eines  $n$ -ären Beziehungstyps verläuft nach dem gleichen Muster, wie bei einem m:n-Beziehungstyp. Es wird ein Relationsschema erzeugt, das alle Attribute des alten Beziehungstyps und alle Primärschlüssel der beteiligten Entitäten als Fremdschlüssel enthält. Der Primärschlüssel dieses Relationsschemas besteht aus allen beteiligten Primärschlüsseln zusammen.

Das Wein-Beispiel von Seite 2–17 soll das verdeutlichen. Dort gab es den ternären Beziehungstyp empfiehlt mit den beteiligten Entitäten Gericht, Kritiker und Wein.



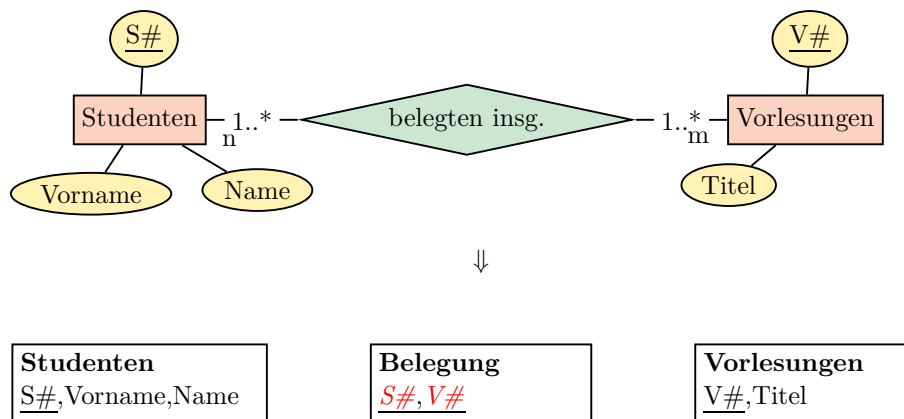


Abbildung 27: Beispiel der Transformation einer beidseitig obligatorischen m:n-Beziehung.

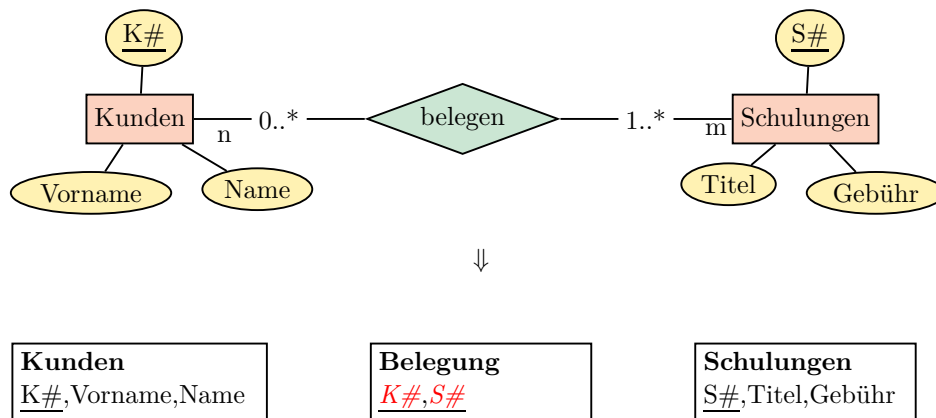


Abbildung 28: Beispiel der Transformation einer einseitig optionalen m:n-Beziehung.

GERICHT	
G#	...

KRITIKER	
K#	...

WEIN	
W#	...

Die Transformation des Beziehungstyps empfiehlt in das Relationenmodell ergibt das Relationsschema Empfehlung:

EMPFEHLUNG		
G#	K#	W#

Im ER-Modell entsprach das dem Übergang von ternärer Beziehung zu Einführung einer Entität Empfehlung und drei binären Beziehungen (siehe Abbildung 16 auf Seite 2-16).

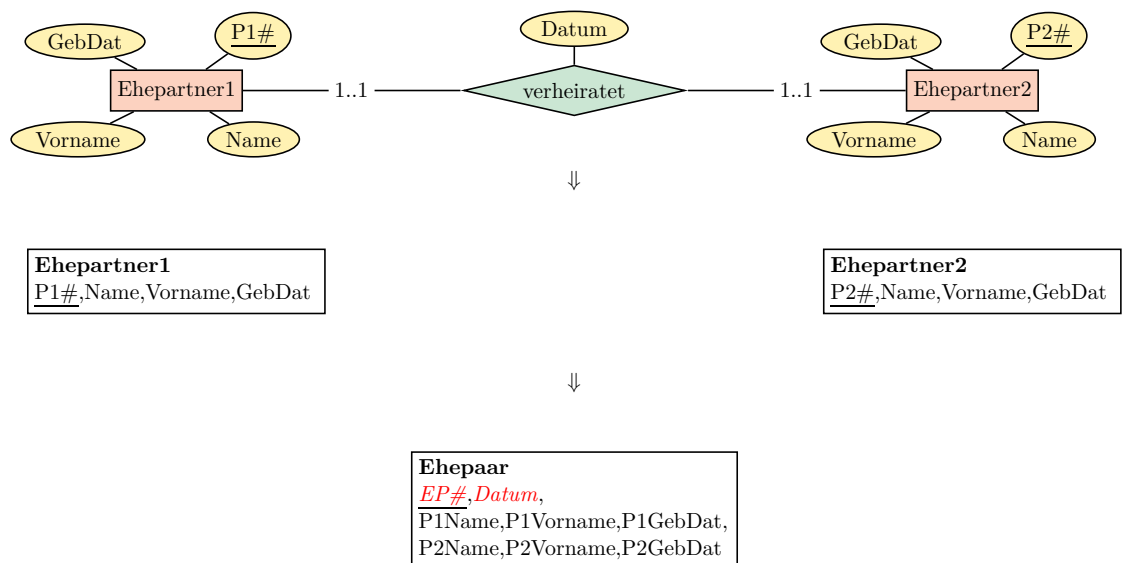


Abbildung 29: Beispiel der Transformation einer 1:1-Beziehung. Die Entitätstypen können zu einem Relationsschema verschmolzen werden.

### 3.4.1. Transformation am Beispiel der Miniwelt *Weine*

Am Beispiel der Miniwelt *Weine*, die hier in der erweiterten Form mit der *isa*-Beziehung *Schaumweine* modelliert werden soll, wird nachfolgend die Transformation vom ER- in das Relationenmodell dargestellt. Basis ist das ER-Modell zur Miniwelt *Weine* (siehe Seite 2–17), das für die Transformation um einen Subtypen *Schaumweine* ergänzt wurde (siehe Abbildung 32).

#### 1. Schritt: Transformation der Entitätstypen

Alle Entitäten (genauer: Entitätstypen) des ER-Modells mit ihren Attributen werden 1 : 1 in Relationen (Relationstypen) des Relationsmodells transformiert. Die Entität *Erzeuger* enthält ein zusammengesetztes Attribut *Adresse*, das in mehrere, einzelne Attribute zerlegt wird. Die Entität *Gerichte* enthält das mehrwertige Attribut *Beilagen*, aus dem eine eigene kleine Relation mit zusammengesetztem Schlüssel geschaffen wird. Daraus entstehen zuerst einmal folgende Relationen:

- *Weine* (W#, Name, Farbe, Jahrgang, Restsuesse)
- *Rebsorten* (R#, Name, Farbe)
- *Erzeuger* (E#, Weingut, Strasse, Ort)
- *Anbauggebiete* (A#, Name, Region, Land)
- *Lizenzen* (L#, Menge)
- *Gerichte* (G#, Name)
- *Beilagen* (G#, BName)
- *Kritiker* (K#, Name, Organisation)
- *Schaumwein* (Name, Verfahren)

#### 2. Schritt: Transformation der Beziehungstypen

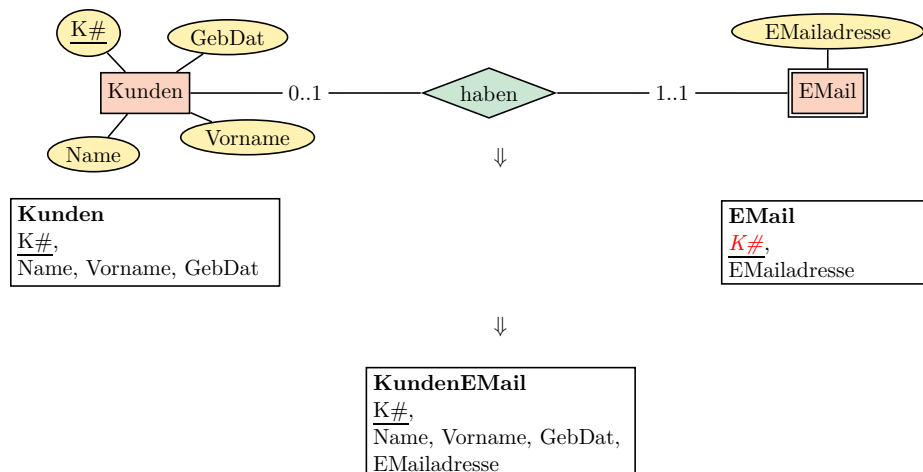


Abbildung 30: Beispiel der Transformation einer einseitig optionalen 1:1-Beziehung. Das Verschmelzen zu einem Relationsschema führt zu Nullwerten.

Bei der Transformation der Beziehungen (Beziehungstypen) müssen die beteiligten Komplexitäten berücksichtigt werden. Nicht alle Beziehungen werden zu Relationen im Relationenmodell, manche werden in bestehende transformierte Entitäten durch Attributhinzunahme integriert.

**Transformation hergestellt\_aus:** Es handelt sich um einen einseitig optionalen  $m : n$  Beziehungstyp und wird in eine eigene Relation *Herstellung* transformiert (i. d. R. werden für Relationen Substantive verwendet.), die die Fremdschlüssel der beteiligten Entitäten (Weine und Rebsorten) sowie das Attribut Anteil des Beziehungstyps enthält.

- Herstellung (W#, R#, Anteil)

Es findet ein Semantikverlust statt, da die Optionalität nicht mehr erkennbar ist. Die Integritätsbedingung höchstens 7 Rebsorten muss ebenfalls hinzugefügt werden.

**Transformation produziert\_von:** Der  $n : 1$  Beziehungstyp (Wein – Erzeuger) wird in die Entität Wein durch Hinzunahme des Fremdschlüssels der Erzeuger-ID E# integriert.

- Weine (W#, Name, Farbe, Jahrgang, Restsuesse, E#)

Semantikverlust, da Optionalität nicht mehr erkennbar. Integritätsbedingung notwendig, da Wein von genau einem Erzeuger produziert wird.

**Transformation sitzt\_in:** Genauso wie in produziert\_von: Der  $n : 1$  Beziehungstyp (Erzeuger – Anbaugesbiet) wird in die Entität Erzeuger durch Hinzunahme der Anbaugesbiet-ID A# integriert.

- Erzeuger (E#, Weingut, Strasse, Ort, A#)

Semantikverlust, da Optionalität nicht mehr erkennbar. Integritätsbedingung notwendig, da Erzeuger in genau einem Anbaugesbiet sitzen muss.

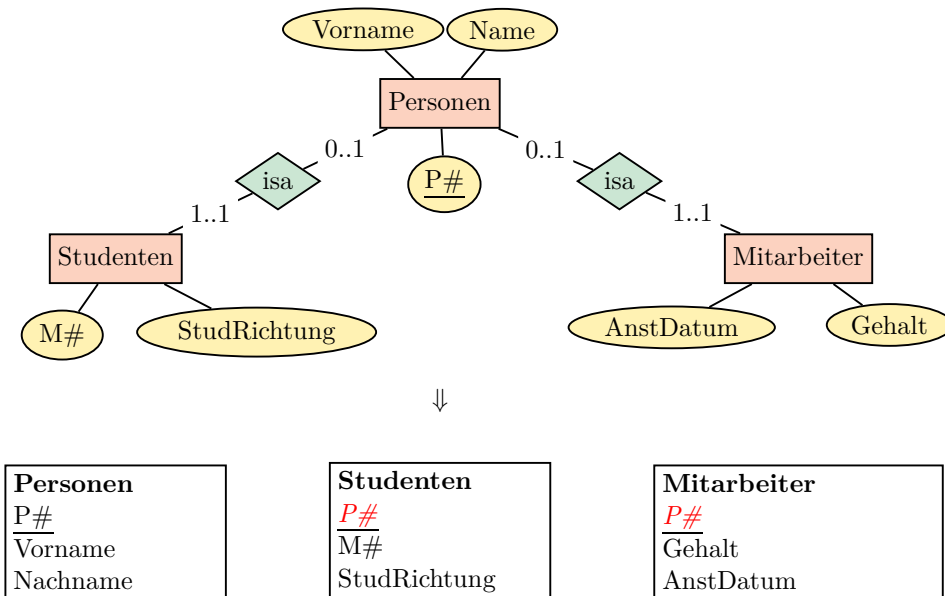


Abbildung 31: Beispiel der Transformation einer Subtypenbeziehung

**Transformation besitzt:** Der  $n : 1$  Beziehungstyp (Lizenz – Erzeuger) entspricht einer  $c : nc$  Beziehung. Da auf beiden Seiten die Teilnahme an der Beziehung optional ist, ergeben sich Nullwerte bei einer Auflösung der Beziehung. Daher wird eine eigene Relation erzeugt.

- Lizenzbesitz (E#, L#)

Semantikverlust, da Optionalität nicht mehr erkennbar. Integritätsbedingung notwendig, da eine Lizenz höchstens einmal vergeben werden darf.

**Transformation empfiehlt:** Der ternäre Beziehungstyp wird in eine eigenständige Relation Empfehlung transformiert. Die Relation enthält die drei Fremdschlüssel der beteiligten Entitäten (Gericht, Kritiker und Wein).

- Empfehlung (G#, K#, W#)

Es treten keine Semantikverluste auf.

**Transformation der isa-Beziehung:** Bei der Transformation der isa-Beziehung erhält der Subtyp (Schaumweine) den Schlüssel des Supertyps (W#) als zusätzliches Attribut, der zum Primärschlüssel des Subtyps wird. Die isa-Beziehung verschwindet.

- Schaumweine (W#, SName, Verfahren)

### 3. Schritt: Transformationsergebnis

Als Ergebnis erhält man die folgenden zwölf Relationen im Relationenmodell:

- Weine (W#, Name, Farbe, Jahrgang, Restsuesse, E#)
- Rebsorten (R#, Name, Farbe)
- Erzeuger (E#, Weingut, Strasse, Ort, A#)
- Anbaugebiete (A#, Name, Region, Land)
- Lizenzen (L#, Menge, E#)

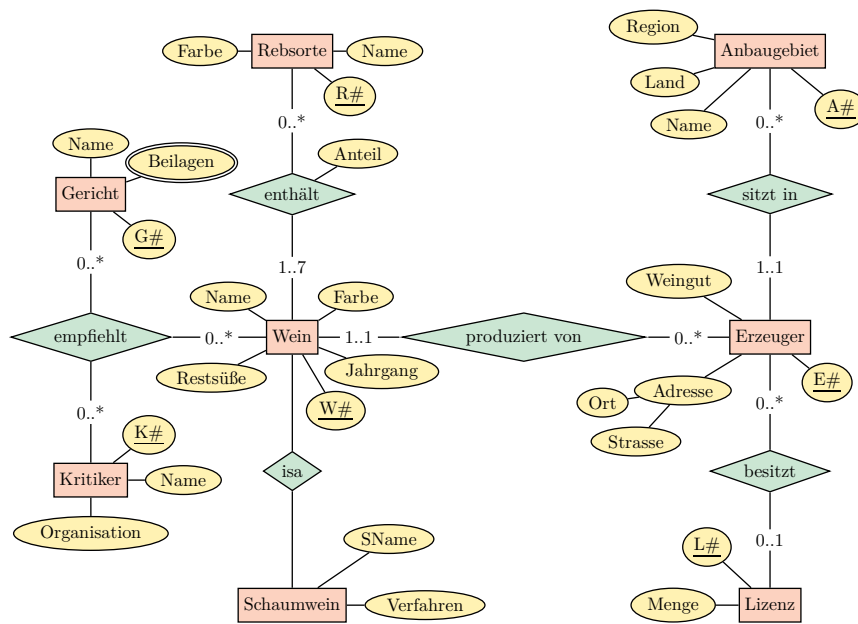


Abbildung 32: Miniwelt Weine mit zusätzlicher **isa**-Beziehung Schaumweine. Dieses ER-Modell bildet die Basis der Beispieltransformation in das Relationenmodell in diesem Kapitel.

- Gerichte (G#, Name)
- Beilagen (G#, BName)
- Kritiker (K#, Name, Organisation)
- Schaumweine (W#, SName, Verfahren)
- Herstellung (W#, R#, Anteil)
- Empfehlung (G#, K#, W#)
- Lizenzbesitz (E#, L#)

### 3.5. Übungszettel A-5

**Aufgabe 5.1.** *Transformation eines ER-Modells in ein Relationenmodell und Erstellung einer Datenbasis.*

- (a) *Transformieren Sie das ER-Modell einer Miniwelt (Beispiele unten) in das Relationenmodell. Begleiten Sie Ihre Entwurfsentscheidungen durch einen Kommentartext. Beachten Sie bei der Kommentierung insbesondere Semantikverluste, die durch die Schematransformation auftreten können und formulieren Sie ggf. Integritätsbedingungen, die für die Relationsausprägungen gelten müssen.*
- (b) *Entwerfen Sie DES-kompatible, übersichtliche Populationen ohne Nullwerte für die Relationenschemata. Geben Sie zu den Attributen Wertebereiche an. Orientieren Sie sich dabei an der Datei pcshop.ddl aus dem Material zur Vorlesung. Achten Sie auf plausible Daten. Zur Orientierung: Die Datenmenge sollte sich in Tabellenform und 10 Punkt Schriftgröße etwa auf einer DIN-A4-Seite überblicken lassen. Beachten Sie, keine Sonderzeichen in den Schemata zu verwenden. Integritätsbedingungen werden hier nicht implementiert.*

*Die folgenden Miniwelten können gerne total oder partiell, überlappend oder disjunktiv in Arbeitsgruppen zu zweit erstellt und bearbeitet werden. Zur Auswahl stehen folgende – auf den entsprechenden Übungszetteln bzw. im Vorlesungsskript umrissene – Miniwelten:*

- |                                              |                                              |
|----------------------------------------------|----------------------------------------------|
| (1) <b>Wachschutz</b> , Blatt 2, Aufgabe 2.1 | (4) <b>Reisebüro</b> , Blatt 3, Aufgabe 3.3  |
| (2) <b>Banken</b> , Blatt 2, Aufgabe 2.3     | (5) <b>Süßwaren</b> , Blatt 3, Aufgabe 3.4   |
| (3) <b>Theater</b> , Blatt 2, Aufgabe 2.4    | (6) <b>Bibliothek</b> , Blatt 4, Aufgabe 4.1 |

### 3.6. Relationenalgebra

Eine der beiden formalen Anfragesprachen, die für die Arbeit mit relationalen Datenbanken konzipiert wurde, ist die relationale Algebra (auch Relationenalgebra), wie sie bereits in Kapitel 3.1.2 beschrieben wurde.<sup>29</sup> Die Relationenalgebra ist eine prozedural orientierte Sprache; ein relationenalgebraischer Ausdruck enthält implizit einen Abarbeitungsplan, wie die Anfrage auszuwerten ist. Dies führt dazu, dass die Relationenalgebra bei der Realisierung von Datenbanksystemen eine wesentliche Rolle spielt.

In diesem Kapitel sollen Operationen der Relationenalgebra genauer unter die Lupe genommen werden und – was bisher zu kurz kam – diese an einem größeren Beispiel zum Einsatz gebracht werden, das eine echte Semantik besitzt: einer Universitäts-Miniwelt.<sup>30</sup> Ein weiterer

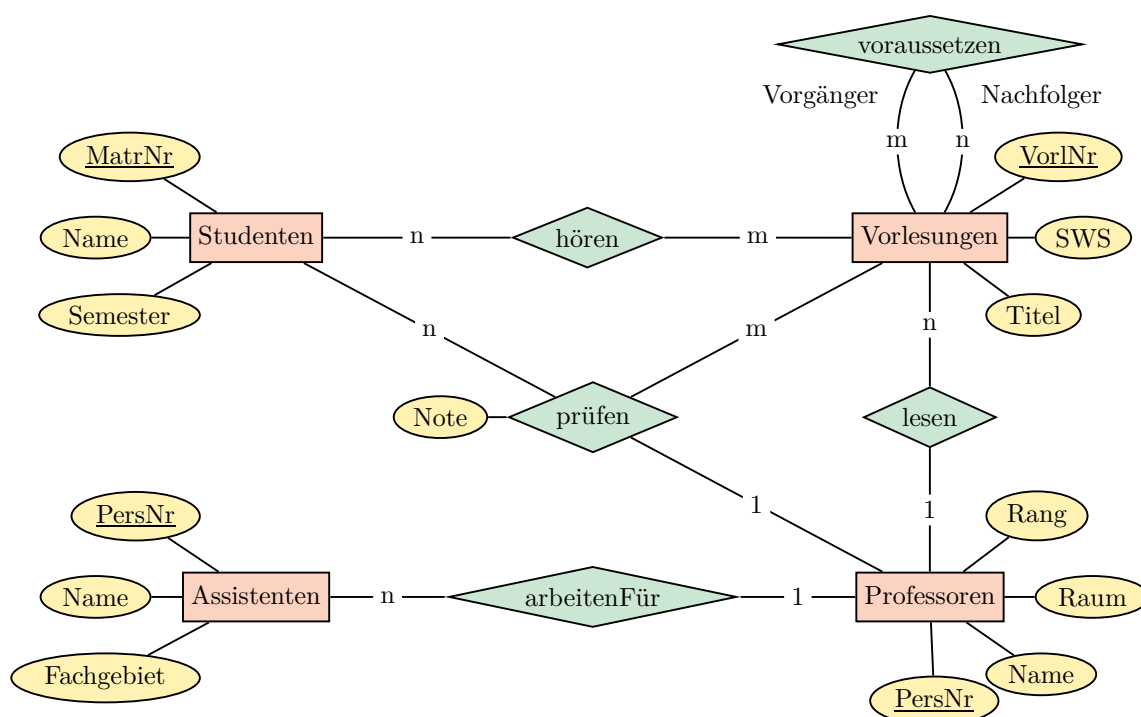


Abbildung 33: ER-Diagramm des Universitätsbeispiels in Chen-Notation

Schwerpunkt soll die simultane Umsetzung in relationenalgebraische Anfrageausdrücke unter Zuhilfenahme von DES sein. Alle Anfragen sollen neben dem mathematischen Ausdruck auch für die bereits erwähnte Software aufgeführt und die (derzeit) spezifischen Schwierigkeiten der Software beleuchtet werden.<sup>31</sup> Eine ausführliche Übersicht über die Syntax relationaler Ausdrücke in DES findet man im Manual zur Konsolenversion in Kapitel 6.5 auf Seite 156.

<sup>29</sup>Die zweite ist das sogenannte Relationenkalkül, das es in zwei Ausprägungen (Domänen- und Tupelkalkül) gibt. Diese wird im folgenden Kapitel behandelt.

<sup>30</sup>Das Beispiel ist entnommen aus [KE09].

<sup>31</sup>In der aktuellen Version 6.5 von April 2020 funktioniert die `rename`-Funktion nicht wie erwartet. Unbedingt vermeiden muss man das Erzeugen von Tabellen mit doppelten Attributnamen, dann scheint das Problem nicht aufzutreten. Auf [http://des.sourceforge.net/html/known\\_bugs.html](http://des.sourceforge.net/html/known_bugs.html) (05.11.2016) ist der `rename`-Bug beschrieben und ein Workaround benannt, da das Problem bis zur aktuellen Version nicht behoben ist (Stand: 19.09.2020).

### 3.6.1. Grundoperationen der Relationenalgebra

Voraussetzung: 1NF

Zur Erinnerung: Relationenalgebraische Ausdrücke lassen sich aus einer Basismenge von Operationen ausdrücken. Diese sind:

Notation	Name	DES
$R \cup S$	Vereinigung	union
$R \setminus S$	Differenz	difference
$R \times S$	Kartesisches Produkt	product
$\pi_A(R)$	Projektion	project
$\sigma_F(R)$	Selektion	select
$\rho_{A \leftarrow B}(R)$	Umbenennung	rename

### 3.6.2. Abgeleitete Operationen der Relationenalgebra

Mit Hilfe der Grundoperationen lassen sich abgeleitete Operationen darstellen. Eine davon – der Verbund  $\bowtie$ , auch Join genannt – stellt (in verschiedenen Ausprägungen) die wichtigste Operation dar, die in allen relationalen Anfragesprachen mit Produktivanspruch vorhanden ist.

Notation	Name	Definition	DES
$R \cap S$	Durchschnitt	$R \setminus (R \setminus S)$	intersect
$R \bowtie_{A \theta B} S$	Thetaverbund, Thetajoin	$\sigma_{A \theta B}(R \times S)$	zjoin
$R \bowtie_{A=B} S$	Gleichverbund, Equijoin	$\sigma_{A=B}(R \times S)$	-
$R \bowtie S$	Natürlicher Verbund, natural Join	(a)	njoin
$R \div S$	Division	(b)	division
$R \ltimes S$	Linker Halbverbund, Left Semijoin	$\pi_{A(R)}(R \bowtie S)$	-
$R \rtimes S$	Rechter Halbverbund, Right Semijoin	$\pi_{A(S)}(R \bowtie S)$	-
$R \Join S$	Linker äußerer natürl. Verbund, Left outer nat. Join	(c)	nljoin
$R \Join S$	Rechter äußerer natürl. Verbund, Right outer nat. Join	(c)	nrjoin
$R \Join S$	Äußerer natürlicher Verbund, (Full) outer natural Join	(c)	-
$R \Join_{A \theta B} S$	Linker äußerer Verbund, Left outer Join	(c)	ljoin
$R \Join_{A \theta B} S$	Rechter äußerer Verbund, Right outer Join	(c)	rjoin
$R \Join_{A \theta B} S$	Äußerer Verbund, (Full) outer Join	(c)	fjoin

Die äußeren Verbünde zählen zu den Operationen, die sich nur mit Hilfe von Nullwerten erklären lassen. Beim linken äußeren Verbund z. B. findet man in der Ergebnisrelation alle Tupel der linken Relation wieder, die – sofern sich kein Join-Partner in der rechten Relation findet – rechts mit Nullwerten aufgefüllt werden.

- (a) Sei  $\mathcal{R} = \mathcal{A}(R)$ ,  $\mathcal{S} = \mathcal{A}(S)$  und  $\mathcal{X} = \mathcal{A}(R) \cap \mathcal{A}(S)$ .  
Dann ist  $R \Join S = \pi_{\mathcal{R} \setminus \mathcal{S} \cup \mathcal{X} \cup \mathcal{S} \setminus \mathcal{R}}(\sigma_{R \cdot \mathcal{X} = S \cdot \mathcal{X}}(R \times S))$ .



- (b) Sei  $\mathcal{R} = \mathcal{A}(R)$ ,  $\mathcal{S} = \mathcal{A}(S)$ .  
Dann ist  $R \div S = \pi_{\mathcal{R} \setminus \mathcal{S}}(R) \setminus \pi_{\mathcal{R} \setminus \mathcal{S}}((\pi_{\mathcal{R} \setminus \mathcal{S}}(R) \times S) \setminus R)$ .
- (c) Die äußeren Verbunde sind nur unter Zuhilfenahme von Nullwerten erklärbar und werden hier nicht formal definiert.

### 3.6.3. Gesetze der Relationenalgebra

Seien  $R$ ,  $S$  und  $T$  vereinigungskompatible Relationen. Dann gilt:

- (a) Vereinigung und Durchschnitt sind *kommutativ*:  
 $R \cup S = S \cup R$   
 $R \cap S = S \cap R$
- (b) Vereinigung und Durchschnitt sind *assoziativ*:  
 $(R \cup S) \cup T = R \cup (S \cup T)$   
 $(R \cap S) \cap T = R \cap (S \cap T)$
- (c) Vereinigung und Durchschnitt sind *distributiv*:  
 $R \cup (S \cap T) = (R \cup S) \cap (R \cup T)$   
 $R \cap (S \cup T) = R \cap S \cup R \cap T$
- (d) Die Differenz  $\setminus$  ist *nicht* kommutativ:  $R \setminus S \neq S \setminus R$ .
- (e) Für das Kartesische Produkt gilt u.a.:  
 $(R \cup S) \times T = R \times T \cup S \times T$   
 $(R \cap S) \times T = R \times T \cap S \times T$   
 $(R \setminus S) \times T = R \times T \setminus S \times T$   
 $(R \times T) \cap (S \times U) = (R \cap S) \times (T \cap U)$  aber:  
 $(R \times T) \cup (S \times U) \subseteq (R \cup S) \times (T \cup U)$
- (f) Sei  $\mathcal{R} = \mathcal{A}(R)$ ,  $\mathcal{S} = \mathcal{A}(S)$  und  $\mathcal{X}$  eine Teilmenge der Attribute von  $R \times S$ . Dann gilt:  
 $\pi_{\mathcal{X}}(R \times S) = \pi_{\mathcal{X} \cap \mathcal{R}}(R) \times \pi_{\mathcal{X} \cap \mathcal{S}}(S)$ .
- (g) Ist  $P$  ein Prädikat über der Attributmenge von  $R \times S$ , das sich als Konjunktion zweier Prädikate  $P_A$  und  $P_B$  darstellen lässt, so gilt:  
 $\sigma_P(R \times S) = \sigma_{P_A}(R) \times \sigma_{P_B}(S)$ .
- (h) Der Gleichverbund und der natürliche Verbund sind kommutativ und assoziativ.  
 $R \bowtie_{A=B} S = S \bowtie_{A=B} R$   
 $R \bowtie_{A=B} (S \bowtie_{A=B} T) = (R \bowtie_{A=B} S) \bowtie_{A=B} T$   
 $R \bowtie S = S \bowtie R$   
 $R \bowtie (S \bowtie T) = (R \bowtie S) \bowtie T$
- (i) Für die Division gilt:  
 $(R \times S) \div S = R$ , aber  
 $(R \div S) \times S \subseteq R$ .

### 3.6.4. Relationenalgebraische Ausdrücke am Beispiel

Betrachten wir die Universitäts-Miniwelt aus Abbildung 33, in der Professoren, Assistenten und Studenten existieren. Professoren lesen, Studenten hören Vorlesungen, in denen sie wiederum von Professoren geprüft werden. Bei der Transformation des ER-Modells in das Relationenmodell ergeben sich die folgenden Relationsschemata:

**Assistenten** (PersNr, Name, Fachgebiet, Boss)

**Professoren** (PersNr, Name, Rang, Raum)

**Studenten** (MatrNr, Name, Semester)

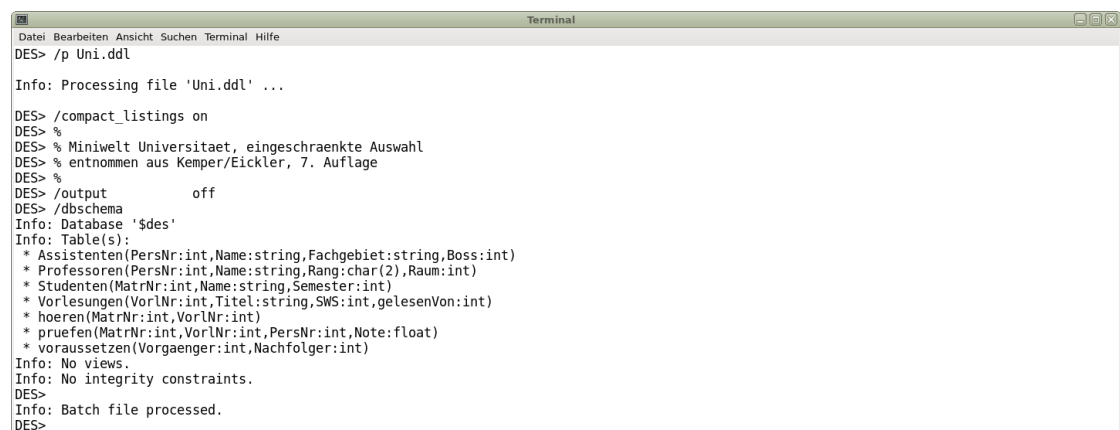
**Vorlesungen** (VorlNr, Titel, SWS, gelesenVon)

**hoeren** (MatrNr, VorlNr)

**pruefen** (MatrNr, VorlNr, PersNr, Note)

**voraussetzen** (Vorgaenger, Nachfolger)

Die Relationsausprägungen findet man in gedruckter Form im Anhang D.1. Sie stehen als ddl-Datei auf der Materialiensseite zur Verfügung. Um diese in DES einzulesen, startet man die Software am besten in dem Verzeichnis, in dem die Datei `Uni.ddl` liegt und liest die Datei mit dem `process`-Kommando auf oder gibt sie – das Wrapper-Skript von der Materialiensseite vorausgesetzt – beim Aufruf von DES als Aufrufparameter mit.<sup>32</sup>



```

Terminal
Datei Bearbeiten Ansicht Suchen Terminal Hilfe
DES> /p Uni.ddl

Info: Processing file 'Uni.ddl' ...

DES> /compact_listings on
DES> %
DES> % Miniwelt Universitaet, eingeschaenkte Auswahl
DES> % entnommen aus Kemper/Eickler, 7. Auflage
DES> %
DES> /output off
DES> /dbschema
Info: Database '$des'
Info: Table(s):
* Assistenten(PersNr:int,Name:string,Fachgebiet:string,Boss:int)
* Professoren(PersNr:int,Name:string,Rang:char(2),Raum:int)
* Studenten(MatNr:int,Name:string,Semester:int)
* Vorlesungen(VorlNr:int,Titel:string,SWS:int,gelesenVon:int)
* hoeren(MatNr:int,VorlNr:int)
* pruefen(MatNr:int,VorlNr:int,PersNr:int>Note:float)
* voraussetzen(Vorgaenger:int,Nachfolger:int)
Info: No views.
Info: No integrity constraints.
DES>
Info: Batch file processed.
DES>

```

Abbildung 34: Einlesen der Datei `Uni.ddl` in DES.

### Relationenalgebraische Anfragen mit steigender Komplexität

Im Folgenden soll an einigen Beispielen mit steigender Komplexität die Wirkung der relationenalgebraischen Anfragen gezeigt werden. Parallel zur mathematischen Formulierung des RA-Ausdruckes gibt es immer eine passende Anfrage für das DES-System.

<sup>32</sup>Stillschweigend wird davon ausgegangen, dass die Konsolenanwendung von DES verwendet wird.

### Alle Langzeitstudenten (Mindestens 10 Semester).

Das ist ein typischer Anwendungsfall des select-Operators:

$$\sigma_{\text{Semester} \geq 10}(\text{Studenten})$$

```
% Alle Langzeitstudenten (mindestens 10 Semester).  
select Semester >= 10 (Studenten);
```

### Personalnummern und Namen aller Professoren und Assistenten

Zwei Tupelmengen, die nicht vereinigungskompatibel sind (Professoren und Assistenten), müssen erst durch die Projektion auf die gemeinsame Attributmenge in die passende Form *gezwängt* werden:

$$\pi_{\text{PersNr}, \text{PName}}(\text{Professoren}) \cup \pi_{\text{PersNr}, \text{AName}}(\text{Assistenten})$$

```
% Personalnummern und Namen aller Professoren und Assistenten.  
(project PersNr, Name (Professoren))  
union  
(project PersNr, Name (Assistenten));
```

### Matrikelnummern aller Studenten ohne Prüfung

Die Matrikelnummern aller Studenten, die bereits eine Prüfung abgelegt haben, erhält man durch Projektion auf dieses Attribut des Relationenschemas *pruefen*. Die Tupelmenge der Studenten ohne Prüfung ergibt sich dann durch Differenzbildung.

$$\pi_{\text{MatrNr}}(\text{Studenten}) \setminus \pi_{\text{MatrNr}}(\text{pruefen})$$

```
% Matrikelnummern aller Studenten ohne Prüfung.  
(project MatrNr (Studenten))  
difference  
(project MatrNr (pruefen));
```

### Matrikelnummern und Namen aller Studenten ohne Prüfung

Man benötigt die Ergebnismenge der vorigen Anfrage und bildet den natürlichen Verbund mit der Relation *Studenten* mit anschließender Projektion auf die interessanten Attribute.

$$\pi_{\text{MatrNr}, \text{Name}}((\pi_{\text{MatrNr}}(\text{Studenten}) \setminus \pi_{\text{MatrNr}}(\text{pruefen})) \bowtie \text{Studenten})$$

```
% Matrikelnummern und Namen aller Studenten ohne Prüfung.  
project MatrNr, Name  
(
```

```
(
  (project MatrNr (Studenten))
  difference
  (project MatrNr (pruefen))
)
njoin
Studenten
);
```

### Der Vor-Vorgänger der Vorlesung 5216

Die Relation *voraussetzen* enthält nur den direkten Vorgänger. Will man den Vorgänger zweiter Stufe ermitteln, berechnet man das kartesische Produkt von *voraussetzen* mit sich selbst und selektiert diejenigen Tupel, für die der Nachfolger in der ersten Inkarnation von *voraussetzen* mit dem Vorgänger der zweiten übereinstimmt. Die gesuchte Vorlesungsnummer ist dann der Vorgänger der ersten Inkarnation, für die der Nachfolger der zweiten den Wert 5216 hat.

V=Vorgaenger  
N=Nachfolger

$$\pi_{V1.V}(\sigma_{V2.N=5216}(\rho_{V1}(\text{voraussetzen}) \bowtie_{V1.N=V2.V} \rho_{V2}(\text{voraussetzen}))))$$

Die Umsetzung in DES gelingt praktisch genauso, allerdings muss man wegen des rename-Bugs darauf achten, dass die beiden Hilfsrelationen keine gleich benannten Attribute aufweisen.

```
% Vor-Vorgänger von VorlNr 5216
% In DES unbedingt gleichbenannte Spalten vermeiden!
project V1
(
  select N2=5216
  (
    (rename V1 (V1,N1) (voraussetzen))
    zjoin N1=V2
    (rename V2 (V2,N2) (voraussetzen))
  )
);
```

### Alle Studenten mit den von ihnen gehörten Vorlesungen

Hier ist ein Doppel-Verbund nötig. Wegen der Assoziativität der Verbundoperation kann man die Klammern weglassen.

Studenten  $\bowtie$  hoeren  $\bowtie$  Vorlesungen

```
% Alle Studenten mit den von ihnen gehörten Vorlesungen.
Studenten njoin hoeren njoin Vorlesungen;
```

### Alle Vorlesungen mit den Professoren, die sie lesen

Zwar haben die Attribute PersNr und gelesenVon die gleiche Semantik und die gleiche

Domain, sind aber unterschiedlich benannt, so dass ein natürlicher Verbund scheitert.  
Abhilfe: Umbenennen der Attribute. Das klappt trotz rename-Bug (sogar) in DES:

$\rho_{\text{PersNr} \leftarrow \text{gelesenVon}}(\text{Vorlesungen}) \bowtie \text{Professoren}$

```
% Alle Vorlesungen mit den Professoren, die sie lesen.  
(rename Vorl (VNr,Titel,SWS,PersNr) (Vorlesungen))  
njoin  
Professoren;
```

### Alle Studenten, die bereits eine Prüfung abgelegt haben

Das Problem ist mit Hilfe eines Halbverbundes lösbar:

$\pi_{\text{MatrNr},\text{Name}}(\text{Studenten} \bowtie \text{pruefen})$

Wird ohnehin projiziert, tut es auch ein natürlicher Verbund. In DES gibt es keinen Operator für Halbverbünde.

```
% Alle Studenten, die bereits eine Prüfung abgelegt haben.  
project MatrNr,Name (Studenten njoin pruefen);
```

### Alle Studenten, die *Ethik* und *Bioethik* gehört haben

Endlich einmal eine Anwendung der Divisions-Operation. Man baue zuerst eine Relation, die Matrikelnummer, Name und den Titel aller besuchten Vorlesungen enthält. Dann selektiere man alle Tupel zu einem der beiden Titel. Diese Relation dividiere man durch die Relation, die nur diese beiden Titel enthält:

$$\frac{\pi_{\text{MatrNr},\text{Name},\text{Titel}}(\text{Studenten} \bowtie \text{ hoeren } \bowtie \text{ Vorlesungen})}{\pi_{\text{Titel}}(\sigma_{\text{Titel}='Ethik' \vee \text{Titel}='Bioethik'}(\text{Vorlesungen}))}$$

```
% Alle Studenten, die 'Ethik' und 'Bioethik' besucht haben.  
project MatrNr,Name,Titel (Studenten njoin hoeren njoin Vorlesungen)  
division  
project Titel (select Titel='Ethik' or Titel='Bioethik' (Vorlesungen));
```

### Auswertung über mehrere Tupel hinweg

Die Selektionsoperation *sieht* nur einzelne Tupel, so dass ein Vergleich über mehrere Tupel hinweg nicht möglich ist. In der (später noch behandelten) marktbeherrschenden relationalen Datenbanksprache SQL gibt es sogenannte Aggregationsfunktionen, die Attribut-Auswertungen über eine Tupelmenge vornehmen. Hier einige Beispiele, in denen das in der RA auch möglich ist.

**Matrikelnummern aller Studenten, die mindestens zwei Vorlesungen besucht haben.**

```
% Matrikelnummern aller Studenten,
% die mindestens zwei Vorlesungen besucht haben
project M1
( select V1<>V2 and M1=M2 (
  (rename T1 (M1,V1) ( hoeren)
  product
  (rename T2 (M2,V2) ( hoeren)
  ))
);
```

### Größte Semesteranzahl eines Studenten

Wenn wir z. B. wissen wollen, welcher Student die meisten Semester auf dem Buckel hat, müssen wir (im Prinzip) jede Ausprägung des Attributes Semester mit jeder anderen vergleichen.<sup>33</sup> Als ersten Schritt bilden wir das kartesische Produkt der Relation *Studenten* mit sich selbst. Damit kann man den Vergleich zweier Studiendauern zwischen beliebigen Studenten (also unterschiedlichen Tupeln) auf den Vergleich der Studiendauern innerhalb *eines* Tupels zurückführen. Um später die gleichnamigen Attribute unterscheiden zu können, wird eines umbenannt. Aus der Menge aller so entstandenen Tupel wählen wir nur die aus, die nicht die größten Studiendauern sind (und projizieren auf diese):

$$\pi_{S.Sem}(\sigma_{S.Sem < S1.Sem}(S \times \rho_{S1}(S)))$$

Nun müssen wir diese Menge nur noch von der Menge aller Studiendauern abziehen.

$$\pi_{Sem}(S) \setminus \pi_{S.Sem}(\sigma_{S.Sem < S1.Sem}(S \times \rho_{S1}(S)))$$

In DES gelingt die Anfrage mit Hilfe einer Umbenennung, weil sonst doppelte Attributnamen auftreten:

```
% Die längste Studiendauer aller Studenten.
project Semester (Studenten)
  difference
project Semester
(
  select Semester < Semester1
  (
    Studenten
    product
    (rename Studenten1 (MatrNr1,Name1,Semester1) (Studenten))
  )
);
```

<sup>33</sup>Es ist sofort klar, dass das eine *teure* Operation sein kann, weil sie  $\mathcal{O}(n^2)$  ist.

### 3.7. Übungszettel A-6

**Aufgabe 6.1.** Zum Bibliotheksbeispiel (Blatt 4, Aufgabe 4.3) finden Sie im Vorlesungsmaterial die Implementierung einer Population für den Relationenalgebra-Simulator DES. Implementieren Sie die Anfragen in DES und testen Sie diese.

**Aufgabe 6.2.** Gegeben seien drei Relationen  $R$ ,  $S$  und  $T$  durch die folgenden Ausprägungen:

$R$		$S$		$T$
$A$	$B$	$B$	$C$	$C$
$a$	$b$	$b$	$c$	$c$
$c$	$b$	$e$	$a$	$d$
$d$	$e$	$b$	$d$	

Berechnen Sie von Hand und mit DES (ggf. sind Attribute umzubenennen)

- |                     |                                           |
|---------------------|-------------------------------------------|
| (a) $R \cup S$      | (e) $\sigma_{A=C}(R \times S)$            |
| (b) $R \setminus S$ | (f) $S \bowtie R$                         |
| (c) $R \bowtie S$   | (g) $(R \bowtie S) \div T$                |
| (d) $\pi_A(R)$      | (h) $R \bowtie_{B < C} S$ (ASCII-Ordnung) |

**Aufgabe 6.3.** Geben Sie relationenalgebraische Ausdrücke an, die folgende Anfragen an die Weine-Miniwelt realisieren. Testen Sie Ihre Ergebnisse mit Hilfe von DES. Verwenden Sie dabei die Populationen aus der Datei `Weine.ddl` aus der aktuellen Vorlesung.

- Alle Weine ab dem Jahrgang 2000.
- Die Namen aller Farben, die bei Rebsorten und Weinen vorkommen.
- Die Namen aller Gerichte und der zum Gericht gereichten Beilagen.
- Die ID aller Weine, die nicht empfohlen wurden.
- Die Namen und Farben der Weine, die nicht empfohlen wurden.
- Das Weingut aller Erzeuger, die mehr als einen Wein produzieren.
- Das Weingut aller Erzeuger von Schaumweinen.
- Die Namen aller Beilagen, zu denen ein Schaumwein empfohlen wird.
- Die Namen und zugestandenen Produktionsmengen aller Weine aus Deutschland.
- Die Lizenznummern aller Weine, die aus schwarzen Trauben hergestellt wurden.
- Die Namen der Weine des Erzeugers mit der größten zugestandenen Produktionsmenge.

### 3.8. Relationale Anfragesprachen

Neben der Relationenalgebra gibt es noch weitere Anfragesprachen für das Relationenmodell, das sogenannte **Relationenkalkül**. Dieses gibt es in zwei Ausprägungen, dem **Tupelkalkül** und dem **Domänenkalkül**.<sup>34</sup> Im Relationenkalkül wird eine Anfrage als deklarativer Ausdruck formuliert. Dieser Ausdruck enthält – im Gegensatz zur Relationenalgebra – keinen Hinweis darauf, wie der Ausdruck auszuwerten ist. Das relationale Kalkül gilt daher nicht als prozedurale Sprache. In der relationalen Algebra kann eine Suchanfrage als Verschachtelung mehrerer Operationen formuliert werden, deren Reihenfolge zwar nicht das Ergebnis aber die Effizienz der Anfrage beeinflusst. Eine solche Reihenfolge ist im Relationenkalkül nicht enthalten, auch nicht implizit. Es wird, einfach gesagt, nur angegeben, was am Ende enthalten sein soll.

Wir werden in diesem Kapitel konstruktiv zeigen, dass eine Suchanfrage, die in der relationalen Algebra formuliert werden kann, auch im relationalen Kalkül formuliert werden kann und umgekehrt. Die Ausdruckskraft der beiden Anfragekonzepte ist gleich groß – sieht man einmal von der Möglichkeit ab, im Relationenkalkül auch unendliche Tupelmengen deklarieren zu können (siehe Kapitel 3.8.3).

#### 3.8.1. Das Tupelkalkül

Das Tupelkalkül basiert auf der Spezifikation sogenannter Tupelvariablen, die sich jeweils auf bestimmte Relationen beziehen. Eine Tupelvariable ist ein 'Behälter' für ein Tupel der betreffenden Relation des Relationenmodells. Eine Anfrage im Tupelkalkül hat die Form

$$\{t \mid F(t)\}$$

wobei  $t$  die Zielliste (oder Targetliste) und  $F(t)$  ein Bedingungsausdruck für die Tupelvariable  $t$  ist. Das Ergebnis ist die Menge aller Tupel  $t$ , die die Bedingung  $F(t)$  erfüllen.  $t$  ist dabei eine sogenannte freie Variable, also eine Stelle im Ausdruck, die für eine Substitution vorgesehen ist. Die Ergebnismenge hängt von den freien Variablen im Ausdruck ab.

#### Bausteine des Tupelkalküls

Den Relationenschemata  $R, S, \dots$  eines Datenbankschemas  $\mathcal{DB}$  sind jeweils Tupelmengen zugeordnet. Über diesen Tupelmengen variieren Variablen  $t, t_1, \dots, u, v, \dots$ . Die Einschränkung einer Tupelvariable  $t$  auf eine Teilmenge  $X$  der Attribute von  $DB$  wird mit  $t[X]$  (oder auch  $t.X$ ) bezeichnet. Ein Ausdruck im TRC wird mit Hilfe folgender Bausteine gebildet:

- *Alphabetzeichen*
  - *Tupelvariablen*  $t, u, v, \dots$ , die über den Tupelmengen der Relationen der DB variieren. Die Stelligkeit wird manchmal als Exponent notiert.
  - *Konstanten*  $c_1, c_2, \dots$ , die Literale der jeweiligen Wertebereiche  $D_{c_1}, D_{c_2}, \dots$  sind.
- *Prädikate*  $R(t), S(u)$ , die  $t \in R, u \in S$  bedeuten.

<sup>34</sup>Im Englischen *tupel relational calculus* (TRC) und *domain relational calculus* (DRC).



- **Vergleichsprädikate**  $t[A] \theta u[B]$  oder  $t[A] \theta c$ , mit der Bedeutung des Vergleichs zweier Tupelkomponenten auf vergleichbaren Attributen bzw. des Vergleichs mit einem vergleichbaren Literal,  $\theta \in \{=, \neq, <, >, \leq, \geq\}$
- **Logische Operatoren**  $\wedge, \vee, \neg$ ,
- **Quantoren**  $\forall, \exists$

## Atomare Formeln

Mit Hilfe dieser Grundbausteine kann man nun atomare **wohlgeformte Formeln** (sogenannte wffs) bilden.

wff = well formed  
formula

- $R(t_1, \dots, t_n)$ , wo  $R$  ein Prädikat ist und die  $t_i, 1 \leq i \leq n$  Variablen oder Konstante,
- $t_i \theta t_j$  mit  $\theta \in \{=, \neq, <, >, \leq, \geq\}$  und  $t_i, t_j$  Variable oder Konstante

Alle in den atomaren Formeln vorkommenden Variablen sind *frei*.

## Zusammengesetzte Formeln

Aus den atomaren wffs lassen sich – zusammen mit logischen Operatoren und dem All- und Existenzquantor – zusammengesetzte wffs bilden.

(a) Sind  $F_1, F_2$  und  $F$  wffs, so sind auch

$$F_1 \wedge F_2, F_1 \vee F_2, \neg F \text{ und } (F)$$

wffs. Die freien Variablen der atomaren Formeln sind auch in den zusammengesetzten wffs frei, die gebundenen bleiben gebunden.

(b) Sei  $F$  eine Formel mit freier Variable  $x$ . Dann sind auch

$$\exists x (F) \text{ und } \forall x (F)$$

wffs. Alle freien Vorkommen von  $x$  in  $F$  werden durch Quantoren gebunden. Gebundene Variablen werden eingeführt, wenn nur auf einen Teil einer Relation zugegriffen wird. Sie sind notwendig, um die ganze relation darzustellen, sind aber nicht Teil der Ergebnismenge. Sie heißen **gebundene Variablen**.

Andere zusammengesetzte wffs als die hier genannten gibt es nicht. Damit ist die eingangs genannte Notation  $\{t \mid F(t)\}$  genauer umfasst.  $F(t)$  ist also nichts anderes als eine wohlgeformte Formel mit der einzigen freien Variablen  $t$ . Stelligkeit und Attribute der Tupelvariablen  $t$  sind hier nicht festgelegt und ergeben sich implizit.<sup>35</sup> Will man es genauer formulieren, schreibt man

Vollständigkeit der wffs

$$\{t^{(n)}[A_1, \dots, A_n] \mid F(t)\}$$

Das Anfrageergebnis ist die Menge aller Tupel, deren Attributwerte bei einer Substitution in  $F$  das resultierende  $F$  wahr machen.

<sup>35</sup>Für die Verwendung von DES gibt es diese Möglichkeit so nicht, dort wird das Problem durch eine explizite Angabe der Attribute, die in  $t$  enthalten sein sollen, gelöst.

## Relationenalgebra und Tupelkalkül

Jede relationenalgebraische Anfrage kann mit Hilfe des Tupelkalküls formuliert werden, es gilt  $RA \subseteq TRC$ . Das ist leicht einzusehen, wenn man eine Konstruktionsvorschrift für jeden relationenalgebraischen Ausdruck in TRC angeben kann. Dies zeigt Tabelle 35.

RA	TRC
$\sigma_{A\theta B}(R)$	$\{t   R(t) \wedge t[A] \theta t[B]\}$
$\pi_{AB}(R)$	$\{t   \exists u (R(u) \wedge t[A] = u[A] \wedge t[B] = u[B])\}$
$R \cup S$	$\{t   R(t) \vee S(t)\}$
$R \cap S$	$\{t   R(t) \wedge S(t)\}$
$R \setminus S$	$\{t   R(t) \wedge \neg S(t)\}$
$R \times S$	$\{t^{(n+m)}[A_1, \dots, A_n, B_1, \dots, B_m]$ $\quad   \exists u \exists v (R(u) \wedge S(v) \wedge t[A_1, \dots, A_n] = u \wedge$ $\quad \quad \quad t[B_1, \dots, B_m] = v)\}$

Abbildung 35: Transformationsvorschriften für relationenalgebraische Teilausdrücke in das Tupelkalkül.

### 3.8.2. Das Domänenkalkül

Im Domänenkalkül wird das Ergebnis aus Attributwerten zusammengesetzt, die Variablen repräsentieren also keine Tupel sondern Attribute. Während sich aus dem Tupelkalkül SQL entwickelt hat, entstand aus dem Domänenkalkül die Abfragesprache QBE (question by example), die z. B. in Microsoft Access verwendet wird.

#### Bausteine des Domänenkalküls

Zum Aufbau *atomarer Formeln* über  $\mathcal{D}$  werden folgende Bausteine benötigt. Dabei ist  $\mathcal{D}$  die Menge der Wertebereiche der Attributmenge  $\mathcal{A}(DB)$  eines Relationenmodells.

- *Alphabetzeichen*
  - Variablen  $x, y, z, \dots$ , die über je einem Wertebereich  $D_x, D_y, D_z, \dots$  variieren, d. h. ein Variablenalphabet,
  - Konstanten  $c_1, c_2, \dots$ , die Literale der jeweiligen Wertebereiche  $D_{c_1}, D_{c_2}, \dots$  sind (z. B. 42 oder 'Hallo').
- Prädikate  $R(x, y, \dots)$ ,  $S(z, u, \dots)$ , die für existierende Relationstupel  $t \in R, t' \in S$  wahr sind, für nicht in  $R$  oder  $S$  existierende Tupel falsch (z. B.  $y$  ist in  $R$ ).
- Logische Operatoren  $\wedge, \vee, \neg$ ,
- Quantoren  $\forall, \exists$ ,
- Vergleichsprädikate  $=, \neq, <, >, \leq, \geq$

## Atomare Formeln

Aus den genannten Grundbausteinen kann man auch hier wieder atomare Formeln bilden. Dabei gibt es zwei verschiedene Ausprägungen:

- $R(t_1, \dots, t_n)$ , wo  $R$  ein Prädikat ist und die  $t_i, 1 \leq i \leq n$  Variablen oder Konstante,
- $t_i \theta t_j$  mit  $\theta \in \{=, \neq, <, >, \leq, \geq\}$  und  $t_i, t_j$  Variable oder Konstante

Auch hier sind alle in den atomaren Formeln vorkommenden Variablen *frei*.

## Zusammengesetzte Formeln

Die zusammengesetzten Formeln im Domainkalkül ergeben sich auf die gleiche Weise wie im Tupelkalkül (siehe Seite 3–35). Daraus lassen sich nun Anfragen im Wertebereichskalkül (kurz: DRC-Anfragen) erstellen. Eine DRC-Anfrage ist ein Ausdruck der Form

$$\{x_1, \dots, x_n \mid F(x_1, \dots, x_n)\}$$

mit freien Variablen  $x_1, \dots, x_n$  und einer wohlgeformten Formel  $F$  über diesen und weiteren (gebundenen) Variablen. Sie liefert diejenigen Tupel  $(a_1, \dots, a_n)$ , bei denen die Substitution von  $x_i$  durch  $a_i$  in  $F$  den Wahrheitswert *wahr* ergibt.

## Relationenalgebra und Domänenkalkül

Auch das DRC ist genauso ausdrucksstark wie die Relationenalgebra, was wiederum durch Konstruktion gezeigt wird. Vereinfachungen: Die  $\theta$ -Prädikate der Relationenalgebra müssen

RA	DRC
$\sigma_\theta(R)$	$\{x_1, \dots, x_n \mid R(x_1, \dots, x_n) \wedge \theta'\}$
$\pi_A(R)$	$\{x_i, \dots, x_j \mid R(x_1, \dots, x_i, \dots, x_j, \dots, x_n)\}, x_i, \dots, x_j \in \text{dom}(A)$
$R \cup S$	$\{x_1, \dots, x_n \mid R(x_1, \dots, x_n) \vee S(x_1, \dots, x_n)\}$
$R \setminus S$	$\{x_1, \dots, x_n \mid R(x_1, \dots, x_n) \wedge \neg S(x_1, \dots, x_n)\}$
$R \times S$	$\{x_1, \dots, x_n, y_1, \dots, y_m \mid R(x_1, \dots, x_n) \wedge S(y_1, \dots, y_m)\}$

Abbildung 36: Transformationsvorschriften für relationenalgebraische Teilausdrücke in das Domänenkalkül.

mit den entsprechenden Variablen in einem Prädikat  $\theta'$  des DRC angegeben werden. Existenzquantifizierungen sind weggelassen.

### 3.8.3. Sichere Ausdrücke

Bei der Verwendung von All- und Existenzquantoren muss man sicherstellen, dass der resultierende Ausdruck im betrachteten Kontext sinnvoll ist. Ein **sicherer Ausdruck** im Relationenkalkül stellt sicher, dass eine *endliche Anzahl* von Tupeln erzeugt wird, andernfalls bezeichnet man ihn als **unsicher**. Der Ausdruck

$$\{t \mid \neg \text{Professoren}(t)\}$$

ist unsicher, weil er unendlich viele Tupel (des Universums) erzeugt, nämlich alle die, die *keine* Professoren sind. Sicherheit kann durch einen TRC mit *Bereichsdeklarationen* erreicht werden:

- Definition einer *Zielliste* mit Elementen der Form  $B : t[A]$  mit  $t$  Tupelvariable zu einer Relation  $R$ , die  $A$  im Schema hat. Ist  $A = B$ , so kann kurz  $t[A]$  geschrieben werden, kommen alle Attribute von  $R$  im Ergebnis vor, so wird auch  $t[*]$  geschrieben.
- Definition einer *Bereichsliste* mit Elementen der Form  $R(t)$ ,
- einer Formel  $F$ , deren freie Variablen in der Bereichsliste genau einmal vorkommen, deren Atome von der Form  $t[A] \theta c$  oder  $t[A] \theta t[B]$  sind und deren gebundene Variablen mit Bereichen assoziiert sind.

Dieser *TRCRD* (tuple relational calculus with range declarations) liegt der Definition der *select*-Konstruktion von SQL zugrunde. Er ist weniger mächtig als die RA, denn die Vereinigung ist nicht ausdrückbar.

### 3.8.4. TRC und DRC in DES

Die Formulierung von Anfragen im Tupel- oder Domänenkalkül in DES weist gegenüber der mathematischen Formulierung einige Besonderheiten auf, auf die in diesem Kapitel kurz eingegangen werden soll.

- (a) Tupel- und Domänenvariablen müssen in DES mit einem Großbuchstaben beginnen. Um Relations- oder Attributnamen von Ihnen zu unterscheiden, müssen diese – wenn sie groß geschrieben werden, in einfache Anführungszeichen gesetzt werden. Will man das vermeiden, verwendet man stattdessen lieber klein geschriebene Relations- und Attributnamen.
- (b) In der mathematischen Formulierung einer Anfrage im Tupelkalkül werden die Attribute der Tupelvariable in der Zielliste durch die Referenzierung in der Bedingungsliste festgelegt. Will man beispielsweise den relationalalgebraischen Ausdruck  $\pi_{A,C}(R(A, B) \bowtie S(B, C))$  im TRC nachbilden, gelingt dies durch

$$\{t \mid (\exists x)(\exists y)(R(x) \wedge S(y) \wedge t[A] = x[A] \wedge t[C] = y[C] \wedge x[B] = y[B])\} \quad \text{TRC}$$

In DES ist eine solche Formulierung nicht möglich, da die Tupelvariable der Zielliste nicht in der Bedingungsliste spezifiziert wird. DES bietet hier die Möglichkeit, das Tupel in der Zielliste durch explizite Angabe der Attribute zusammenzusetzen. Zudem kann der Existenzquantor weggelassen werden, weil durch das Auftreten der Relationsnamen  $R$  und  $S$  in der Zielliste diese bereits qualifiziert sind.

$$\{x.A, y.C \mid R(x) \text{ and } S(y) \text{ and } x.B = y.B\} \quad \text{DES}$$

Einschränkend ist zu sagen, dass jede Tupelvariable in der Zielliste auf der Bedingungsseite spezifiziert sein muss!

- (c) Im Domänenkalkül ist jedes Attribut einer Relation, das nicht in der Zielliste auftritt, eine gebundene Variable und muss daher durch einen Quantor spezifiziert werden. Soll beispielsweise die Projektion auf das Attribut  $A$  der Selektion auf  $C = 3$  der Relation  $R(A, B, C)$  hergestellt werden, ist die Formulierung im TRC

$$\{A \mid (\exists B, C) R(A, B, C) \wedge C = 3\} \quad \text{DRC}$$

In DES kann die gebundene Variable  $B$  durch einen Unterstrich  $_$  ersetzt werden, so dass die Formulierung ohne die oft zahlreichen Existenzquantoren auskommt. Zudem kann die Konstante gleich als Variablenwert angegeben werden:

$$\{A \mid R(A, \_, 3)\}$$

DES

### 3.9. Übungszettel A-7

**Aufgabe 7.1.** Gegeben sind vier Relationen  $R, S, T, U$  mit den Beispielausprägungen:

$R$			$S$		$T$		$U$	
$A$	$B$	$C$	$A$	$D$	$D$	$F$	$D$	$F$
$a$	200	50	$a$	75	20	$f$	20	$k$
$c$	400	20	$a$	90	27	$p$	25	$m$
$d$	300	70	$c$	25	34	$g$	55	$h$
$e$	200	200	$c$	55	60	$r$	75	$f$
			$c$	60	75	$f$	95	$j$
			$d$	80	80	$j$		
			$d$	27	90	$v$		
			$e$	34				
			$e$	20				
			$e$	95				

(a) Berechnen Sie die Ergebnismengen der Relationenalgebra-Anfragen von Hand:

- (i)  $\pi_{BC}(\sigma_{A < c \wedge B < 250}(R))$ ,
- (ii)  $\sigma_{D < 30}(T \cup U)$ ,
- (iii)  $\pi_{AF}(R \bowtie_{C=D} U)$ ,
- (iv)  $\pi_{ACDF}(R \bowtie_{C < D} U)$ ,
- (v)  $U \setminus T$ ,
- (vi)  $T \setminus U$ ,
- (vii)  $U \cap T$ ,
- (viii)  $R \bowtie (S \bowtie T)$ ,
- (ix)  $S \bowtie U$ ,
- (x)  $S \bowtie T$ .

(b) Berechnen Sie die Ergebnismengen der Anfragen aus (a) in DES.

(c) Notieren Sie die Anfragen aus (a), soweit möglich, im DRC.

(d) Notieren Sie die Anfragen aus (a), soweit möglich, im TRC.

(e) Kontrollieren Sie ihre Anfragen mit Hilfe von DES.

**Aufgabe 7.2.** Geben Sie die Relationenalgebra-Anfragen zum Bibliotheksbeispiel (Blatt 4 Aufgabe 3) im Wertebereichskalkül (DRC) an.

**Aufgabe 7.3.** Implementieren Sie zu allen Relationenalgebra-Beispielanfragen zur Universitäts-Miniwelt (siehe Seite 3–28 ff.) entsprechende Anfragen in Tupel- und Domänenkalkül in DES und testen Sie diese am vorhandenen Datenmaterial.

**Aufgabe 7.4.** Gegeben ist das Relationenschema

**Angestellter** (ANr, Vorname, Nachname, Adresse, Gehalt, VorgANr, AbtNr)

**Abteilung** (AbtNr, AbtName, AbtLnr, AbtLSeit)

**AbtStandorte** (AbtNr, AbtOrt)

**Projekt** (PNr, PName, POrt, AbtNr)

**Mitarbeit** (ANr, PNr, Stunden)

**Angehoeiger** (ANr, AngehoeigerName, Geschl, GebDatum, Beziehung)

- (a) Gesucht sind relationenalgebraische Ausdrücke für die folgenden Anfragen:
- (i) Namen und Adressen aller Angestellten, die für die Forschungs- und Entwicklungsabteilung arbeiten.
  - (ii) Gesucht sind die Projektnummer, die Nummer der leitenden Abteilung und der Nachname des Abteilungsleiters für alle in Frankfurt angesiedelten Projekte.
  - (iii) Gesucht sind die Namen aller Angestellten, die an allen von Abteilung 5 geführten Projekten mitarbeiten.
  - (iv) Gesucht ist eine Liste von Projektnummern für Projekte, an denen Angestellte mit Nachnamen 'Schmidt' beteiligt sind (entweder als Mitarbeiter oder als Abteilungsleiter).
  - (v) Eine Liste aller Angestellten-Namen, die keine Angehörigen haben.
  - (vi) Eine Liste der Namen aller Abteilungsleiter, die mindestens einen Angehörigen haben.
  - (vii) Namen aller Angestellten in Abteilung 5, die mehr als 10 Stunden pro Woche am Produkt 'Waschmaschine' arbeiten.
- (b) Gesucht sind Domainkalkül-Ausdrücke für die Anfragen (i),(ii),(vi),(vii) aus (a)
- (c) Gesucht sind Tupelkalkül-Ausdrücke für die Anfragen (i)–(iv) aus (a).

### 3.10. Datalog

Deduktive Datenbanksysteme nähern sich aus einer ganz anderen Richtung dem Problem der Informationsgewinnung. Etwa in den 1970er Jahren suchte man nach Ansätzen, große Faktenmengen nicht durch Aufzählung, sondern durch einzelne definierende Ausdrücke darzustellen. Als vielversprechend schien damals das relationale Datenmodell zu sein. Etwa gleichzeitig entwickelte sich im Kontext der Künstlichen Intelligenz die logischen Programmiersprache Prolog (*programming in logic*). Die Zusammenführung dieser beiden Ansätze führte im Jahr 1978 zur Entwicklung der Datenbankabfragesprache Datalog.<sup>36</sup> Das Grundprinzip dieser logischen Programmier- bzw. Datenbanksprache lautet: *Fakten + Regeln = Ergebnisse*. Parallelen zur Funktionalen Programmierung werden hier deutlich.

Derzeit arbeitet Google an der weiteren Neuauflage einer deduktiven, logischen Datenbanksprache: *Logica* (Logic with aggregation). Ziel soll in erster Linie die bessere Verarbeitung von "Big Data" sein und die einfachere und übersichtlichere Erstellung von Datenbankabfragen.

<sup>37</sup>.

#### 3.10.1. Prolog

Prolog wurde in den 1970er Jahren entwickelt, jedoch unterschied sich die Syntax der verschiedenen Implementierungen stark voneinander. Erst 1995 wurde der mittlerweile zum Quasi-Standard erhobene Edinburgh-Dialekt zu einer ISO-Norm, weswegen er auch ISO-Prolog genannt wird. Prolog ist eine schwach typisierte Interpretersprache, die mit kompiliertem Zwischencode arbeitet; wird die zugrundeliegende Datenbasis verändert, wird der Zwischen-code neu erzeugt.

Prolog arbeitet auf der Basis von Fakten und Regeln. Anhand eines einfachen Beispiels soll das dargelegt werden. Das nachfolgende Beispiel entstammt dem Internet und ist quasi das *Hello world*-Programm der logischen Programmierung.<sup>38</sup> Gegeben sei die Datenbasis

```
mann(adam).  
mann(tobias).  
mann(frank).  
frau(eva).  
frau(daniela).  
frau(ulrike).  
vater(adam,tobias).  
vater(tobias,frank).  
vater(tobias,ulrike).  
mutter(eva,tobias).  
mutter(daniela,frank).  
mutter(daniela,ulrike).
```

Diese stellen wahre Aussagen dar und können als Stammbaum einer kleinen Familie interpretiert werden. Das erste Faktum in Form einer Aussage `mann(tobias)` liest sich als: *Tobias*

<sup>36</sup>Eine sehr tiefgehende Darstellung zu Deduktiven Datenbanken findet man in [CGH94].

<sup>37</sup><https://logica.dev/> (23.05.2022). Logica besitzt derzeit nur einen experimentellen Support für PostgreSQL und SQLite.

<sup>38</sup>[http://de.wikipedia.org/wiki/Prolog\\_\(Programmiersprache\)](http://de.wikipedia.org/wiki/Prolog_(Programmiersprache)), abgerufen am 06.11.2022



ist ein Mann. `vater(tobias, frank)` definiert das Faktum: *Tobias ist der Vater von Frank*. Natürlich ist die 'Leserichtung' von `vater(tobias, frank)` eine Frage der Interpretation bzw. des Konsens. Es muss eine Einigung darüber bestehen, wer hier wessen Vater ist.

Anfragen an das System können interaktiv gestellt werden:

```
?- mann(tobias).  
true.  
?- mann(heinrich).  
false.
```

Das System antwortet auf die Anfragen stets mit `true` oder `false`. Werden Variablen (die mit einem Großbuchstaben beginnen) verwendet, liefert der Interpreter alle zutreffenden Fakten.

```
?- frau(X).  
X = eva ;  
X = daniela ;  
X = ulrike.
```

Regel werden in der Form `Head :- Body` formuliert. So lässt sich beispielsweise aus den Fakten mit Hilfe der Regel

```
grossvater(X,Y) :-  
    vater(X,Z),  
    vater(Z,Y).
```

ableiten, wer Großvater väterlicherseits von wem ist. Das System liefert zwei Antworten:

```
?- grossvater(X,Y).  
X = adam,  
Y = frank ;  
X = adam,  
Y = ulrike.
```

### 3.10.2. Datalog

Datalog besteht aus einer Teilmenge von Prolog und bildet die Grundlage einer Reihe von weiteren Entwicklungen logischer bzw. deklarativer Datenbanksprachen.<sup>39</sup> Aus komplexitätstheoretischen Gründen<sup>40</sup> werden in Datalog nur sogenannte Horn-Klauseln verwendet, die zwar die Aussagekraft der Sprache verringern, dafür aber immer endliche und in  $\mathcal{O}(n^2)$  auf Erfüllbarkeit untersuchbare Ausdrücke produzieren.<sup>41</sup>

<sup>39</sup>Eine derzeit aktuelle Entwicklung ist die Datenbanksprache Bloom ([bloom-lang.net](http://bloom-lang.net)) für Cloud Computing und Filtersysteme für große Datenmengen, verwendet u. a. von Google, Bing oder Bitcoin.

<sup>40</sup>Das SAT-Problem ist NP-vollständig (siehe Vorlesung zur Theoretischen Informatik).

<sup>41</sup>In [Vos08, S. 226] findet man eine kurze Darstellung zu Datalog, die die wesentlichen Dinge – auch die Unterschiede zu Prolog – zusammenfasst.

Eine Hornklausel ist eine Klausel mit *höchstens einem* positiven Literal. Um Hornklauseln zu verstehen, muss man erst die Syntax von Datalog analysieren. Wir beginnen ganz 'unten':

## Alphabete

Alphabete bilden die Bausteine einer Sprache. Für Datalog – und genauso für Prolog – ist nennenswert, dass ein wesentlicher Unterschied besteht zwischen der Bedeutung von Zeichenfolgen, die mit einem Großbuchstaben beginnen und denen, die mit einem Kleinbuchstaben beginnen. Erstere sind im wesentlichen Variablen, letztere Konstanten.

- *Variablen* beginnen mit Großbuchstaben oder Unterstrich, wobei der alleinige Unterstrich `_` für eine sogenannte **anonyme Variable** steht.
- *Konstanten* beginnen mit Kleinbuchstaben oder sind Zahlenlitterale:
  - Folgen von Zeichen aus Buchstaben, Unterstrichen oder Ziffern, beginnend mit Kleinbuchstaben, `affe`, `eSeEl`, `g_e_i_e_r`, `k4711`
  - Zahlen, `1`, `-5`, `0`, `4711`, `3.1415926`
  - Folgen von Spezialzeichen (ohne Klammern, Unterstrich, Hochkommata, senkrechter Strich und `%`) für *Operatoren* (hier sind als *Erweiterung* von Datalog auch arithmetische Operatoren wie `=`, `<`, `>=` usw. erlaubt),
  - Beliebige Zeichenfolgen in einfachen Hochkommata, `'Affe'`, `'EsEl'`, `'_'`,
  - `[]` als spezielles Listensymbol.
- *Prädikatsnamen* sind mit Kleinbuchstaben beginnende Zeichenketten.
- *Terme* sind Konstanten oder Variable.

## Atome

Mit Hilfe der Alphabetzeichen lassen sich einfache Ausdrücke – sogenannte Atome – konstruieren. Atome entsprechen Prozeduren/Methoden bzw. Funktionen in imperativer bzw. funktionaler Programmierung, und werden wie Funktionssymbole notiert:  $p(t_1, \dots, t_n)$ . Sie bestehen aus Prädikatsnamen  $p$  und einer Argumentliste  $(t_1, \dots, t_n)$  von Termen. Die *Stelligkeit* (engl. *Arity*) ist die Anzahl der Argumente. In `vater(philip,X)` ist `vater` der Prädikatsname, `vater/2` die Angabe mit Arity, `philip` eine Konstante und `X` eine Variable.

Wichtig sind in diesem Zusammenhang die Begriffe **Litterale**, **Klauseln** und – als Spezialfall davon – sogenannte **Grundklauseln**:

- **Litterale** sind entweder negative ( $\neg p(t_1)$ ) oder positive Atome.
- **Klauseln** sind Mengen von Litteralen.
- **Grundklauseln** enthalten keine Variablen.

## Hornklauseln

Die nach dem amerikanischen Mathematiker Alfred Horn benannten Horn-Klauseln sind Klauseln, die höchstens ein positives Literal enthalten. Es gibt drei verschiedene Typen von Hornklauseln:<sup>42</sup>

- **Fakten** sind positive Grundklauseln, bestehend aus einem einzigen Literal:

*Beispiel:* `vater(tobias, frank).`

- **Regeln** haben genau ein positives Literal und mindestens ein negatives und werden in Form  $p :- q_1, \dots, q_n$  notiert. Die Semantik ist: 'p gilt, wenn  $q_1$  und ... und  $q_n$  gelten'.

Die logische Äquivalenz  $p :- q \equiv q \rightarrow p \equiv \neg q \vee p$  zeigt, dass das negative Literal im Körper (Body) der Regel versteckt ist.

*Beispiel:* `grossvater(X, Y) :- vater(X, Z), vater(Z, Y).`

- **Zielklauseln** (engl. goal) sind Klauseln, die nur aus negativen Literalen bestehen.

## Datalog-Programm

Ein Datalog-Programm besteht aus der Menge aller Fakten und Regeln und wird auch als **Klauselmenge** bezeichnet. Die Menge der Fakten heißt **extensionale Datenbank** (EDB), was den Relationen im relationalen Modell entspricht. Die Menge der Regeln heißt **intensionale Datenbank** (IDB), was im relationalen Modell der Konstruktion von Sichten entspricht. Eine Klauselmenge für genau ein Prädikat heißt **Definition**, wird ein Prädikat durch mehrere Fakten und/oder Regeln definiert, werden diese (unausgesprochen) mit einem ODER verknüpft. Alternativ lassen sich auch diese Regeln auch durch ein Semikolon trennen. Literale, die durch ein Komma separiert sind, werden mit einem UND verknüpft.

## Aussagekraft

Jedes relationenalgebraische Konstrukt lässt sich in Datalog ausdrücken, wie folgende Beispiele zeigen. Dazu seien  $r(A, B)$  und  $s(A, B)$  zwei Relationen. Nun konstruieren wir die Grundoperationen der Relationenalgebra (siehe 3–26).

Relationenalgebra	Datalog
$r \cup s$	$rUs(A, B) :- r(A, B).$ $rUs(A, B) :- s(A, B).$
$r \setminus s$	$rMs(A, B) :- r(A, B), \text{ not } s(A, B).$
$r \times s$	$rXs(RA, RB, SA, SB) :- r(RA, RB), s(SA, SB).$
$\pi_A(r)$	$rP(A) :- r(A, _).$
$\sigma_f(r)$	$rS(A, B) :- r(A, B), f.$

<sup>42</sup>Horn-Klauseln sind nach dem amerikanischen Mathematiker Alfred Horn benannt (1918-2001), der sich einen Namen in der Gitter-Theorie und der universellen Algebra machte.

Eine Selektion kann implizit durch die Verwendung der gleichen Variablenbezeichnungen erfolgen, z. B.:  $\sigma_{A=B}(r) = rS(A,A) :- r(A,A) ..$  Damit ist ein zusätzliches Literal, das die Bedingung ausdrückt, nicht nötig. Sollten Bedingungs-literale  $f_n$  eingeführt werden, so können diese z. B. aus vorher definierten Regeln bestehen, welche die Selektionsbedingung ausdrücken.

Der Umbenennungoperator ist in Datalog nicht nötig. Damit ist beispielhaft gezeigt, dass sich jeder relationalenalgebraische Ausdruck durch eine Datalog-Definition ausdrücken lässt.

Vorsicht ist geboten im Fall der Differenz. Hier handelt es sich nicht um eine Horn-Klausel, weil das negative Literal dies aushebelt. Dass diese Auswertung dennoch gelingt, liegt daran, dass die Klausel stratifiziert ist. Das bedeutet, dass ein negatives Literal  $\neg P$  nur auftreten darf, wenn  $P$  bis zu dieser Stelle vollständig berechnet ist. Für  $s(A,B)$  trifft das zu, für  $s(A, \_)$  wäre das nicht der Fall.

### Rekursion

Eine besondere Fähigkeit von Datalog ist die Möglichkeit, Rekursion auszudrücken, ohne die Rekursionstiefe zu kennen oder anzugeben. Als Beispiel diene der (männliche) Vorfahre, bei dem man nicht weiß, wie viele Generationen er zurückliegt. Dieser lässt sich folgendermaßen definieren:

```
vorfahr(X,Y) :- vater(X,Y).  
vorfahr(X,Y) :- vater(X,Z), vorfahr(Z,Y).
```

### 3.10.3. Datalog und DES

Die Syntax von DES lehnt sich an der von Datalog bzw. Prolog an. Das bedeutet konkret, dass die Notation von Fakten und Regeln vollkommen identisch geschieht und Direktiven wie z. B. das Verwerfen der Faktenmenge (Löschen der Datenbankinhalt) oder das Erstellen neuer Regeln mit den gleichen Befehlen geschieht, jedoch ein Slash (/) vorangestellt wird und man die Klammern weglässt. Während das Hinzufügen einer Regel  $q:-p_1,p_2$  in Prolog als `assert(q:-p1,p2)` notiert wird, lautet der Befehl in DES `/assert q:-p1,p2`. Alternativ werden solche Regeln in einer externen Datei (für mehrfache Verwendung) abgelegt und anschließend mit `/[datei]` konsultiert.<sup>43</sup> In der externen Datei kann dann der `/assert`-Präfix weggelassen werden, wenn DES im Terminal ausgeführt wird.<sup>44</sup>

Der Grund dafür ist klar: Normalerweise bezeichnet z. B. `vater(charles,harry)` ein Faktum, das zu `true` oder `false` auswertet. Soll dieses Faktum aber hinzugefügt werden (weil es noch nicht in der Datenbank existiert), kann das der Interpreter nicht von der Abfrage unterscheiden. Daher wird das (seltenere) Hinzufügen mit einem zusätzlichen Schlüsselwort (also hier `/assert`) gekennzeichnet werden. Da man aus Dateien eher selten einzelne Fakten abfragt, kann das Schlüsselwort in DES innerhalb der externen Datei weggelassen werden. Die nachfolgende Datei `hello.dl` enthält die Datenbasis zum 'Hello-world-Beispiel':

<sup>43</sup>In Prolog wird eine Datei mittels der Anweisung `[datei]` konsultiert; es fehlt syntaktisch nur der Slash gegenüber DES.

<sup>44</sup>Ausführliche Erläuterungen zu Datalog im DES-Manual ab S. 52 (DES V6.5).

```
mann(adam).  
mann(tobias).  
mann(frank).  
frau(eva).  
frau(daniela).  
frau(ulrike).  
vater(adam,tobias).  
vater(tobias,frank).  
vater(tobias,ulrike).  
mutter(eva,tobias).  
mutter(daniela,frank).  
mutter(daniela,ulrike).
```

Eine Datalog-Beispielsitzung in DES könnte beispielsweise so aussehen:<sup>45</sup>

```
DES> /[hello]  
Info: 12 rules consulted.  
DES> grossvater(X,Y):- (vater(X,Z),vater(Z,Y)) ; (vater(X,Z),mutter(Z,Y)).  
Info: Processing:  
    grossvater(X,Y)  
in the program context of the exploded query:  
    grossvater(X,Y) :-  
        vater(X,Z),  
        vater(Z,Y).  
    grossvater(X,Y) :-  
        vater(X,Z),  
        mutter(Z,Y).  
{  
    grossvater(adam,frank),  
    grossvater(adam,ulrike)  
}  
Info: 2 tuples computed.
```

Verwenden Sie ACIDE, die GUI für DES, dann lesen Sie die geöffnete Datei über den Button 'consult' ein.

## Direktiven in DES

Einige wichtige Direktiven im Umgang mit DES zeigt die nachfolgende Tabelle.

<sup>45</sup>Der DES-Prompt **DES** ist derjenige für Datalog. Durch Eingabe des Metakommandos `/datalog` gelangt man immer wieder dorthin.

Direktive	Effekt
<code>/abolish</code>	Löscht die gesamte Datenbank inklusive aller Regeln, Integritätsbedingungen und Sichten
<code>/abolish Name</code>	Löscht alle Prädikate, auf die der Name passt, sowie alle zugehörigen Integritätsbedingungen und Sichten.
<code>/abolish Name/Arity</code>	Löscht alle Prädikate, auf die das Muster passt. Zugehörige Integritätsbedingungen und Sichten werden ebenfalls gelöscht.
<code>/assert Head:-Body</code>	Fügt eine Datalog-Regel hinzu. Wenn Body fehlt, handelt es sich um ein Faktum. Die Reihenfolge der Regeln spielt in Datalog keine Rolle.
<code>/[Namen]</code>	Liest die Regeln der (ggf. durch Komma separierten) Dateien ein. Bereits existierende Regeln werden dabei überschrieben.
<code>/[+Namen]</code>	Liest die Regeln der (ggf. durch Komma separierten) Dateien ein. Bereits existierende Regeln werden dabei NICHT überschrieben.
<code>/consult Datei</code>	Liest die Regel einer Datei ein. Etwaige gleichnamige Regeln werden dabei überschrieben. Gleichbedeutend mit <code>/[Datei]</code> .
<code>/listing</code>	Listet alle Datalog-Fakten und -Regeln auf.
<code>/listing Name</code>	Listet alle Datalog-Fakten und -Regeln zum angegebenen Namen auf.

### 3.11. Übungszettel A-8

**Aufgabe 8.1.** Gegeben sind die Relationen  $r(A, B, C)$ ,  $s(A, B, C)$  und  $t(A, B, C)$ . Schreiben Sie Datalog-Regeln für die folgenden relationenalgebraischen Ausdrücke. Erstellen Sie sich vorher geeignete 'Spieldaten', mit deren Hilfe Sie die Korrektheit Ihrer Ausdrücke überprüfen können.

- (a)  $r \cup s$
- (b)  $r \cap s$
- (c)  $r \setminus s$
- (d)  $(r \cup s) \setminus t$
- (e)  $(r \setminus s) \cap (r \setminus t)$
- (f)  $\pi_{A,B}(r)$

**Aufgabe 8.2.** Sei  $r(X, Y, Z)$  eine Relation. Schreiben Sie Datalog-Regeln für die Selektionsausdrücke  $\sigma_p(r)$ , wobei  $p$  für eines der folgenden Prädikate steht:

- (a)  $X = Y$
- (b)  $X < Y \wedge Y < Z$
- (c)  $X < Y \vee Y < Z$
- (d)  $\neg(X < Y \vee Y < Z)$
- (e)  $\neg((X < Y \vee X > Y) \wedge (Y < Z))$

**Aufgabe 8.3.** Gegeben sind die Relationen  $r(A, B, C)$ ,  $s(B, C, D)$  und  $t(D, E)$ . Schreiben Sie Datalog-Regeln für die folgenden Verbundausdrücke:

- (a)  $r \bowtie s$
- (b)  $s \bowtie t$
- (c)  $r \bowtie s \bowtie t$

**Aufgabe 8.4.** Gegeben sind die Relationen  $r(X, Y, Z)$ ,  $s(X, Y, Z)$ . Schreiben Sie in DES Datalog-Regeln für die  $\theta$ -Verbundausdrücke  $r \bowtie_{\theta} s$ , wobei  $\theta$  die Prädikate  $p$  der Aufgabe 2 darstellt. Dabei sollen die arithmetischen Vergleiche je ein Attribut aus  $r$  auf der linken mit einem Attribut von  $s$  auf der rechten Seite vergleichen, also z. B.  $r \bowtie_{r.X \theta s.Y} s$ .

**Aufgabe 8.5.** Implementieren Sie zu den Relationenalgebra-Beispielanfragen zur Universitäts-Miniwelt (siehe Seite 3–28 ff.) entsprechende Anfragen in Datalog in DES und testen Sie diese am vorhandenen(!) Datenmaterial.

**Aufgabe 8.6.** Nicht alle Verwandtschaftsbeziehungen der altgriechischen Götterwelt sind eindeutig bestimmbar, manche Angaben in den Quellen widersprechen sich oder sind interpretierbar. Dennoch soll der Versuch gewagt werden, eine kleine Genealogie der griechischen Gottheiten in Datalog zu implementieren. Damit sollen Anfragen des Typs: Wie viele Kinder hatte Göttervater Zeus? oder Wer ist Bruder von wem? etc. möglich sein.

Implementiert werde der durch den nachfolgenden Text gegebene Ausschnitt der griechischen Götterwelt.

Alle griechischen Gottheiten stammen von Uranos und der Erdenmutter Gaia ab. Unter anderem zeugten beide Kronos und Rhea. Kronos sollte Uranos als Erdenherrscher nachfolgen. Der von Kronos getötete Uranos hatte dem Kronos vorausgesagt, dass auch ihn einst einer seiner Söhne entthronen werde. Darum verschlang er nach der Geburt alle seine mit Rhea gezeugten Kinder: Hades, Poseidon und die Töchter Hestia, Demeter und Hera. Um Zeus, den Letztgeborenen, vor dem Schicksal seiner Geschwister zu bewahren, verbarg ihn Rhea auf dem kretischen Ida. Als Zeus erwachsen war, zwang er seinen Vater, seine unsterblichen Brüder und Schwestern wieder auszuspeien und entfesselte den Titanenkampf, in dem Zeus die Weltherrschaft erlangte.

Mit Hera, seiner Schwester und Gemahlin, zeugte er Ares, den Kriegsgott, Hephaistos, den Gott des Feuers und der Schmiede; mit Metis, dem Gestalt gewordenen guten Rat, zeugte er Athene. Eine andere Version besagt, dass Athene dem Haupte des Zeus entsprang.

Als Leto, die Geliebte des Zeus, schwanger ging mit den Zwillingen Apollon und Artemis, irrte sie, umhergetrieben von der eifersüchtigen Hera, lange umher, bis sie auf der Insel Delos niederkommen konnte.

Aphrodite wird von Homer eine Tochter des Zeus und der Okeanos-Tochter Dione genannt. Eine andere Version besagt, dass sie dem Blute des entmannten Uranos entsprang.

Hermes, der Götterbote, wurde als Sohn des Zeus und der Pleiade Maia in einer Höhle des arkadischen Berges Kyllene geboren.

Dionysos ist ein Sohn des Zeus und der Semele. Von Hera dazu verleitet, wollte Semele ihren göttlichen Geliebten Zeus in all der Herrlichkeit seiner Blitze sehen, aber sie verbrannte in den himmlischen Flammen. Doch Zeus entband sie von der noch unreifen Frucht, nähte das Kind in seinen Schenkel ein und brachte es dann zur Welt.

Extrahieren Sie aus dem Text eine geeignete Faktenbasis mit allen genannten Personen, ausgehend von einer Relation bzw. Grundklausel **kind/3**. Entscheiden Sie, welche Klauseln Sie noch für Ihre Faktenbasis als sinnvoll ansehen. Stellen Sie die Verwandtschaftsbeziehungen graphisch dar. Implementieren Sie die Datenbasis in einer Datei **goetter.dl**. Testen Sie diese Datenbasis mit einfachen Anfragen.



**Aufgabe 8.7.** Anknüpfend an die Aufgabe 8.6 sind folgende Erweiterungen vorzunehmen:

(a) Entwerfen Sie ein Regelsystem, das mindestens folgende Verwandtschaftsbeziehungen enthält:

- `vater/1`, 'tritt in Vaterrolle auf',
- `vater/2`, 'ist Vater von jemand',
- `dto. für Mutter`,
- `tochter/2`, `sohn/2`, 'ist Tochter bzw. Sohn von jemandem',
- `elternteil/2`, 'ist Elternteil von jemand',
- `bruder/2`, `schwester/2`, `halbbruder/2`, `halbschwester/2`,
- `vorfahr/2`, 'jemand ist Vorfahr eines anderen'.
- `geschwister/2`, `echte_geschwister/2`, `opa/2`, `oma/2`, `enkel/2`,  
`onkel/2`, `tante/2`, `vetter/2`, `base/2`.

Achten Sie auf einen systematischen Aufbau des Regelsystems. Weitere Verwandtschaftsprädikate *ad libitum*. Implementieren Sie die Verwandtschaftsbeziehungen in einer Datei `verwandt.dl`.

(b) Testen Sie die Regelbasis mit der Götterwelt, indem Sie beide Dateien konsultieren und geeignete Anfragen stellen.

(c) Schaffen Sie eine zweite Datenbasis mit der eigenen Familie (Datei `familie.dl`) und testen Sie die Regeln ebenso.

### Literatur zu Kapitel 3

- [CGH94] CREMERS, Armin B. ; GRIEFAHN, Ulrike ; HINZE, Ralf: *Deduktive Datenbanken: Eine Einführung aus der Sicht der logischen Programmierung*. Braunschweig : Vieweg, 1994 (Künstliche Intelligenz). – ISBN 978-3-5280-4700-9
- [EN09] ELMASRI, Ramez A. ; NAVATHE, Shamkant B.: *Grundlagen von Datenbanksystemen*. 3. Auflage. München : Pearson-Studium, 2009. – ISBN 978-3-8689-4012-1
- [KE09] KEMPER, Alfons ; EICKLER, André: *Datenbanksysteme: Eine Einführung*. 7. Auflage. München : Oldenbourg Wissenschaftsverlag, 2009. – ISBN 978-3-486-59018-0
- [Vos08] VOSSEN, Gottfried: *Datenmodelle, Datenbanksprachen und Datenbankmanagementsysteme*. 5. Auflage. München : Oldenbourg, 2008. – ISBN 978-3-4862-7574-2

## 4. Funktionale Abhängigkeiten und Normalisierung

Eine Datenbank wird durch ihr Schema<sup>46</sup> beschrieben, das wiederum aus einzelnen Relationschemas besteht, die durch jeweils eine bestimmte Gruppierung der Attribute gekennzeichnet ist. Bislang sind wir stillschweigend davon ausgegangen, dass der gesunde Menschenverstand ausreicht, um diese Gruppierungen sinnvoll vorzunehmen. Der konzeptuelle Datenbankentwurf, der mit Hilfe eines ER-Diagramms vorgenommen wurde, konnte auf eine definierte Weise in ein relationales Modell überführt werden. Die Identifizierung von Entitäten und Beziehungen innerhalb der zu modellierenden Welt führte bisher in mehr oder weniger natürlicher Art zu einer Zusammenfassung von Attributen zu einem Schema.

In diesem Kapitel soll untersucht werden, ob bestimmte Gruppierungen der zu repräsentierenden Attribute zu Schemas besser oder schlechter sind als andere. Wir werden feststellen, dass aufgrund funktionaler Abhängigkeiten der Attribute bestimmte Normalformen durch unterschiedliche Gruppierung der Attribute erreicht werden können, so dass immer strengere Anforderungen an die Relationsschemas erfüllt werden, die wünschenswerte Eigenschaften zu sichern.<sup>47</sup>

### 4.1. Anomalien

Schlecht entworfene Relationsschemas können zu sogenannten Anomalien führen, die im Folgenden an einem konkreten Beispiel erläutert werden sollen.<sup>48</sup>

PROFVORL						
PersNr	Name	Rang	Raum	VorlNr	Titel	SWS
2125	Sokrates	C4	226	5041	Ethik	4
2125	Sokrates	C4	226	5049	Mäeutik	2
2125	Sokrates	C4	226	4052	Logik	4
...	...	...	...	...	...	...
2133	Popper	C3	52	5259	Der Wiener Kreis	2
2137	Kant	C4	7	4630	Die 3 Kritiken	4

Am Beispiel dieser schlecht entworfenen Relation wollen wir verschiedene Probleme aufzeigen, die man als INSERT-Anomalie, UPDATE-Anomalie bzw. DELETE-Anomalie bezeichnet.<sup>49</sup>

- (a) Soll ein Professor in die Datenbank übernommen werden, der noch keine Vorlesung hält, müssen die entsprechenden Felder mit Nullwerten gefüllt werden. Analoges gilt, wenn eine Vorlesung eingetragen werden soll, für die noch kein Leser gefunden wurde. Das Problem entsteht dadurch, dass in der Relation zwei Entitäten vermischt wurden. Das Problem nennt man *Einfüge-Anomalie*.

<sup>46</sup>... und natürlich durch die darin enthaltenen Daten.

<sup>47</sup>[EN09, S. 299ff.]

<sup>48</sup>Das Beispiel ist der Uni-Miniwelt entnommen (siehe Anhang).

<sup>49</sup>Im folgenden werden auch die deutschen Begriffe *Einfüge-Anomalie*, *Änderungs-Anomalie* und *Lösch-Anomalie* verwendet.

- (b) Zieht ein Professor – sagen wir *Sokrates* – in einen anderen Raum um, müssen mehrere Werte simultan geändert werden. Falsche oder vergessene Änderungen führen zur Korruption der Daten und Verlust der Datenintegrität. Selbst, wenn die Änderungen mit einem Hilfsmittel durchgeführt werden, hat der Entwurf zwei wesentliche Schwächen:

- Erhöhter Speicherbedarf wegen Datenredundanz.
- Leistungseinbußen, da bei Änderungen mehrere Werte geändert werden müssen.

Diese Problematik wird als *Änderungs-Anomalie* bezeichnet.

- (c) Die Vermischung zweier Entitäten führt zu Problemen, wenn man eine löschen will. Will man beispielsweise die Vorlesung *Der Wiener Kreis* löschen, so führt das auch zum Löschen des Professors *Popper*, da dies seine einzige Vorlesung ist. Natürlich könnte man die Vorlesungswerte einfach mit Nullwerten füllen, allerdings ist dies beim Löschen der Vorlesung *Mäeutik* nicht nötig, weil *Sokrates* noch andere Vorlesungen hält. Dieses Phänomen bezeichnet man als *Lösch-Anomalie*.

Die drei aufgeführten Anomalien kann man allgemein formulieren:

#### **INSERT-Anomalie**

Einfügen einer Instanz hängt ab von Zusatzinformationen über andere Instanz.

#### **UPDATE-Anomalie**

Änderung eines Attributwertes erfordert Änderung mehrerer Tupel.

#### **DELETE-Anomalie**

Löschen einer Instanz führt (automatisch) zum Verschwinden einer anderen.

Das Auftreten solcher Anomalien ist an die schlechte Gruppierung der Attribute geknüpft. Um schlechte von guten Gruppierungen zu unterscheiden, ist es notwendig, sich zuerst mit sogenannten funktionalen Abhängigkeiten zu beschäftigen.

## **4.2. Funktionale Abhängigkeiten**

Funktionale Abhängigkeiten sind Eigenschaften der Semantik der Attribute. Je nach dem, welche Bedeutung die Attribute haben, lassen sich Abhängigkeiten bestimmen bzw. festlegen (z. B. zwischen Vorlesungsnummer und -Titel). Im Folgenden wollen wir dennoch keine Relationen und Attribute mehr verwenden, die semantisch belegt sind. Vielmehr treten nur noch abstrakte Namen auf. Das hat den einfachen Grund, dass man sich so besser auf die mathematisch formulierten Zusammenhänge konzentrieren kann, ohne im Hinterkopf über die vielen Möglichkeiten von Abhängigkeiten der semantisch belegten Attribute nachzudenken. Was einem mit der Betrachtung leicht verständlicher Relationen banal erscheint, kann bei großen, unübersichtlichen, fachspezifischen Relationen schnell zu falschen Entscheidungen führen. Die Abstrahierung soll davon ablenken.

**Definition 4.2.1** (Funktionale Abhängigkeit (FA)). Seien  $A$  und  $B$  zwei Attribute aus  $\mathcal{A}(R)$ . Man sagt,  $A$  bestimmt  $B$  funktional, bzw.  $B$  ist funktional abhängig von  $A$ , geschrieben

$$A \rightarrow B,$$

genau dann, wenn für alle Zeitpunkte  $t$  gilt:

$$r_1^t[A] = r_2^t[A] \Rightarrow r_1^t[B] = r_2^t[B]$$

Dass  $A$  funktional  $B$  bestimmt, bedeutet, wenn zwei Tupel  $r$  im  $A$ -Wert übereinstimmen, müssen sie folglich auch im  $B$ -Wert übereinstimmen. Anders gesagt: Ein Attribut  $B$  ist von einem Attribut  $A$  funktional abhängig, wenn durch jeden Wert von  $A$  eindeutig der Wert von  $B$  bestimmt wird.

Typischerweise ist dies bei Schlüsselattributen der Fall. Wenn z. B. das Schlüsselattribut *Personalnummer* den Wert 42 hat, lässt sich genau sagen, welche Werte z. B. die Attribute *Name*, *Adresse*, *Telefonnummer* usw. haben. Es kann keine zwei Tupel mit z. B. der Personalnummer 42 geben, bei denen die anderen Attribute unterschiedliche Werte haben.

Das gilt aber nicht nur für Schlüsselattribute. Im obigen Beispiel gilt  $VorlNr \rightarrow Titel$ , jedoch nur, wenn im Rahmen einer INSERT-Anomalie kein weiterer Eintrag mit z. B.  $VorlNr = 5049$  und  $Titel = Maeutik$  (andere Schreibweise) hinzugefügt wurde, weil dann die funktionale Abhängigkeit nicht mehr gegeben wäre. Die (intendierte) funktionale Abhängigkeit ist durch die Einfüge-Anomalie verloren gegangen ( $VorlNr \nrightarrow Titel$ ).

Funktionale Abhängigkeiten werden aus der Semantik der Relationen heraus definiert. Sie werden also von jemandem festgelegt, der/die über die Bedeutung der Attribute und ihrer Zusammenhänge informiert ist. Damit ergibt sich, dass FAen Eigenschaften des Relationsschemas sind und nicht eines konkreten Datenbestandes. Es ist zwar möglich, aus konkreten Ausprägungen eines Datenbankzustandes Vermutungen über bestehende FAen abzuleiten, aber solange dieser Datenbestand veränderlich ist, bleiben dies nur Annahmen für diesen einen Zeitpunkt.

### Beispiel

In unserem Beispiel gelten z. B. folgende FAen (bzw. gelten nicht):

$$PersNr \rightarrow \{Name, Rang, Raum\}$$

$$VorlNr \rightarrow \{Titel, SWS\}$$

$$SWS \nrightarrow Titel$$

$$Rang \nrightarrow Name$$

$$Titel \rightarrow VorlNr$$

Die letzte FA ist vermutlich korrekt dem Modell entnommen, man kann sich aber auch Fälle vorstellen, bei denen  $Titel \nrightarrow VorlNr$  gilt.

Funktionale Abhängigkeiten lassen sich auch auf ganze Mengen von Attributen erweitern:

**Definition 4.2.2** (FA für Attributmengen). Seien  $X$  und  $Y$  zwei Teilmengen von  $\mathcal{A}(R)$ :  $X, Y \subseteq \mathcal{A}(R)$ . Man sagt,  $X$  bestimmt  $Y$  funktional, bzw.  $Y$  ist funktional abhängig von  $X$ , geschrieben

$$X \rightarrow Y,$$

genau dann, wenn für alle Zeitpunkte  $t$  gilt:

$$r_1^t[X] = r_2^t[X] \Rightarrow r_1^t[Y] = r_2^t[Y]$$

**Bemerkung:** Sei  $X, Y \subseteq \mathcal{A}(R)$  und  $Y \subseteq X$ . Dann gilt trivialerweise

$$X \rightarrow Y \quad A \rightarrow A \quad X \rightarrow X$$

Wenn  $X$   $X$  bestimmt, dann bestimmt  $X$  auch jede Teilmenge von  $X$ , also auch  $Y$ . Solche Arten der FA nennt man trivial, was Anlass gibt zu einer Definition gibt.

**Definition 4.2.3.** Eine FA heißt *trivial*, wenn sie für jede Ausprägung und für jeden Zeitpunkt gilt.

Triviale Funktionale Abhängigkeiten nehmen eine Sonderstellung ein. Gibt es eine solche spezielle FA, können die beteiligten Attributmengen nur Teilmenge und Obermenge sein.

**Satz 4.2.1.** Sei  $X \rightarrow Y$  triviale FA  $\Rightarrow Y \subseteq X$ .

**Beweis:** Sei  $Y \setminus X$  die Menge der Attribute, die nicht in  $X$  sind. OBdA ist  $Y \setminus X \neq \emptyset$ , d. h.  $\exists A \in Y \setminus X, A \notin X$ , dann kann man zwei Tupel  $r_1, r_2 \in R$  konstruieren mit  $r_1[X] = r_2[X]$  und  $r_1[A] \neq r_2[A]$ . Daraus folgt  $X \not\rightarrow Y$ . Also war die Annahme falsch und der Satz richtig.

### Beispiel

Im Folgenden betrachten wird die Relation  $R$ . Es soll angenommen werden, dass die Tupelmenge abschließend für alle Zeiten die FA-Semantik von  $R$  beschreibt.

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

Die Frage ist, von welchen (nichttrivialen) FAen man bei diesem Beispiel ausgehen kann und von welchen nicht. Das kann man an den vorliegenden Ausprägungen erkennen, da der datenbestand nicht mehr verändert werden kann. Durch Überprüfen erhält man:<sup>50</sup>

<sup>50</sup>Häufig wird in der Entwurfstheorie statt der mathematisch korrekten Formulierung  $\{A, B, C\} \rightarrow \{A\}$  die etwas saloppe Formulierung  $ABC \rightarrow A$  verwendet.

linke Seite einattributig			
$A \rightarrow B$	$B \not\rightarrow A$	$C \not\rightarrow A$	$D \rightarrow A$
$A \not\rightarrow C$	$B \not\rightarrow C$	$C \rightarrow B$	$D \rightarrow B$
$A \not\rightarrow D$	$B \not\rightarrow D$	$C \not\rightarrow D$	$D \rightarrow C$
linke Seite zweiattributig			
$\{A, B\} \not\rightarrow \{C\}$	$\{A, B\} \not\rightarrow \{D\}$	$\{A, C\} \rightarrow \{B\}$	$\{A, C\} \rightarrow \{D\}$
$\{A, D\} \rightarrow \{B\}$	$\{A, D\} \rightarrow \{C\}$	$\{B, C\} \not\rightarrow \{A\}$	$\{B, C\} \not\rightarrow \{D\}$
$\{B, D\} \rightarrow \{A\}$	$\{B, D\} \rightarrow \{C\}$	$\{C, D\} \rightarrow \{A\}$	$\{C, D\} \rightarrow \{B\}$
linke Seite dreiattributig			
$\{A, B, C\} \rightarrow \{D\}$	$\{A, B, D\} \rightarrow \{C\}$	$\{A, C, D\} \rightarrow \{B\}$	$\{B, C, D\} \rightarrow \{A\}$
triviale FAen (Auswahl)			
$A \rightarrow A$	$\{A, B\} \rightarrow \{A\}$	$\{A, B\} \rightarrow \{B\}$	...

In der Realität ist es praktisch unmöglich, alle Tupelmengen zu untersuchen, um funktionale Abhängigkeiten aufzudecken. Stattdessen liefert das Datenmodell selbst die FAen. Dass diese dann eingehalten werden, ist Aufgabe des Datenbankmanagementsystems und kann als Maß für die Qualität des Relationenmodells dienen. Funktionale Abhängigkeiten lassen sich aus der Anwendungssemantik ableiten; dies ist Aufgabe des Datenbankdesigners, der für eine sichere Gruppierung der Attribute verantwortlich ist. Aber nicht immer werden alle funktionalen Abhängigkeiten erkannt. Einige ergeben sich aus anderen. Im Allgemeinen ist man – bei gegebener Menge  $F$  aller funktionalen Abhängigkeiten – daran interessiert, die Menge aller daraus ableitbaren FAs  $F^+$  zu ermitteln. Diese Menge nennt man **Hülle** der Menge  $F$ . Zur Herleitung genügen die folgenden drei Inferenzregeln – die auch **Armstrong-Axiome** genannt werden.

**Definition 4.2.4** (Armstrong-Axiome). Seien  $X, Y, Z \subseteq \mathcal{A}(R)$ . Dann heißen

- (A1)  $Y \subseteq X \Rightarrow X \rightarrow Y$  Reflexivität/Inklusionsregel  
 (A2)  $X \rightarrow Y \wedge Y \rightarrow Z \Rightarrow X \rightarrow Z$  Transitivität  
 (A3)  $X \rightarrow Y \Rightarrow XZ \rightarrow YZ$  Verstärkung/Erweiterungsregel

Armstrong-Axiome.

Auch, wenn die Armstrong-Axiome vollständig sind – man also alle FAs mit ihrer Hilfe aus den gegebenen FAs ableiten kann – so ist es in vielen Fällen doch erheblich einfacher, noch weitere, aus den Armstrong-Axiomen direkt ableitbare Schlüsse zu verwenden.

**Satz 4.2.2.** Seien  $X, Y, Z \subseteq \mathcal{A}(R)$ . Dann gilt:

- |      |                                                                         |                         |
|------|-------------------------------------------------------------------------|-------------------------|
| (F1) | $X \rightarrow Y \wedge X \rightarrow Z \Rightarrow X \rightarrow YZ$   | Vereinigung             |
| (F2) | $X \rightarrow YZ \Rightarrow X \rightarrow Y \wedge X \rightarrow Z$   | Zerlegung/Dekomposition |
| (F3) | $X \rightarrow Y \wedge WY \rightarrow Z \Rightarrow XW \rightarrow Z$  | Pseudotransitivität     |
| (F4) | $X \rightarrow YZ \wedge Z \rightarrow W \Rightarrow X \rightarrow YZW$ | Akkumulation            |

*Beweis:*

- (F1)  $X \rightarrow Y \stackrel{A3}{\Rightarrow} XX \rightarrow XY \Rightarrow X \rightarrow XY$   
 $X \rightarrow Z \stackrel{A3}{\Rightarrow} XY \rightarrow YZ$   
 $X \rightarrow XY \wedge XY \rightarrow YZ \stackrel{A2}{\Rightarrow} X \rightarrow YZ$
- (F2)  $YZ \rightarrow Y$  wegen A1.  
 $X \rightarrow YZ \wedge YZ \rightarrow Y \stackrel{A2}{\Rightarrow} X \rightarrow Y$   
 $YZ \rightarrow Z$  wegen A1.  
 $X \rightarrow YZ \wedge YZ \rightarrow Z \stackrel{A2}{\Rightarrow} X \rightarrow Z$
- (F3)  $X \rightarrow Y \stackrel{A3}{\Rightarrow} XW \rightarrow YW$   
 $XW \rightarrow YW \wedge YW \rightarrow Z \stackrel{A2}{\Rightarrow} XW \rightarrow Z$
- (F4)  $Z \rightarrow W \stackrel{A3}{\Rightarrow} YZZ \rightarrow YZW \Rightarrow YZ \rightarrow YZW$   
 $X \rightarrow YZ \wedge YZ \rightarrow YZW \stackrel{A2}{\Rightarrow} X \rightarrow YZW$

### 4.3. Schlüssel

Funktionale Abhängigkeiten stellen eine Verallgemeinerung des Schlüsselbegriffs dar. Das soll jetzt präzisiert werden. Man unterscheidet Schlüssel, Superschlüssel, Schlüsselkandidaten und Primärschlüssel.

**Definition 4.3.1** (Schlüssel, Superschlüssel volle funktionale Abhängigkeit). Sei  $X, Y \subseteq \mathcal{A}(R)$ .

- (a) Gilt  $X \rightarrow \mathcal{A}(R)$ , heißt  $X$  **Superschlüssel** von  $R$ .
- (b) Gilt  $X \rightarrow \mathcal{A}(R)$  und ist  $X$  minimal, gilt also  $\forall A \in X : X \setminus A \not\rightarrow \mathcal{A}(R)$ , heißt  $X$  **Schlüssel** von  $R$ .
- (c) Gilt allgemein  $X \rightarrow Y$  und ist  $X$  minimal, gilt also  $\forall A \in X : X \setminus A \not\rightarrow Y$ , heißt  $Y$  **voll funktional abhängig** von  $X$ .

*Bemerkungen*

- (a) Aufgrund der trivialen FA ist für jede Relation die Gesamtheit aller Attribute  $\mathcal{A}(R)$  ein Superschlüssel.
- (b) Gibt es mehrere Schlüssel, nennt man diese auch **Schlüsselkandidaten** und zeichnet bzw. wählt einen davon als **Primärschlüssel** aus.



(c) Superschlüssel, die keine Schlüsselkandidaten sind, nennt man **echte Superschlüssel**.

Besteht eine FA bei Teilmengen von Attributen der Form  $X \rightarrow Y$ , bedeutet das, dass  $Y$  funktional von  $X$  abhängt. Damit ist  $X$  ein Superschlüssel für die Teilmenge  $Y$  und sogar für  $XY$ . Lässt sich  $X$  nicht weiter reduzieren, ohne dass die funktionale Abhängigkeit von  $Y$  verloren geht (s. (c)), ist  $Y$  *voll funktional* von  $X$  abhängig. Hängt  $Y$  voll funktional von  $X$  ab, dann ist  $X$  ein Schlüsselkandidat für  $XY$ . An dieser Betrachtung wird deutlich, dass die Schlüsseleigenschaft ein Spezialfall der funktionalen Abhängigkeit ist.

## 4.4. Hülle und minimale Überdeckung

Funktionale Abhängigkeiten der Form  $X \rightarrow Y$  stellen logische Abhängigkeiten zwischen Attributmengen dar. Ein DBMS muss diese logischen Abhängigkeiten verfolgen und ggf. zur Erhaltung der Datenintegrität bei Änderungen von Tupeln  $r$  auf  $X$  dafür sorgen, dass sich diese auch auf  $Y$  auswirkt. Solche Überprüfungen beeinflussen die Performanz des Systems erheblich. Man wird daher darauf bedacht sein, die Anzahl der FAen möglichst klein zu halten. Da sich mit Hilfe der Armstrong-Axiome aus einer gegebenen Menge von FAen weitere ableiten lassen, kann es mehrere Mengen funktionaler Abhängigkeiten geben, aus denen sich die gleichen FAen ableiten lassen. Wie man dies prüfen kann, soll im Kapitel zur Hüllenbildung geklärt werden.

Dieses Kapitel beschäftigt sich mit Methoden, für gegebene Mengen FAen zu ermitteln, ob es andere – zu der gegebenen Menge äquivalente – Mengen von FAen gibt, aus der sich die gleichen FAen ableiten lassen. Im Hinblick auf die bereits erwähnten Probleme der Performanz eines DBMS soll auch noch die Frage geklärt werden, ob es eine in gewissem Sinn minimale Menge unter diesen gibt. Diese Frage wird im Kapitel zur minimalen Überdeckung behandelt.<sup>51</sup>

### 4.4.1. Hüllenbildung

Die Frage nach der Hülle einer Menge von FAen (also inklusive aller weiteren, ableitbaren FAen) liegt auf der Hand, ist aber nicht so einfach zu entscheiden. Zwar kann man mit Hilfe der Armstrong-Axiome aus einer gegebenen Menge von FAen weitere ableiten, jedoch ist nicht klar, wie dies systematisiert werden kann. Woher weiß man, dass man alle ableitbaren FAen abgeleitet hat? Um diese Frage zu klären, soll im folgenden Kapitel ein Umweg eingeschlagen werden. Statt die Hülle einer Menge  $\mathcal{F}$  von FAen zu bestimmen, ermittelt man die Hüllen von *Attributmengen*, also Teilmengen  $X \subseteq \mathcal{A}(R)$  eines Relationenschemas  $R$ . Aus diesen lassen sich alle neuen, nichttrivialen FAen ermitteln und damit die sogenannte Hülle von  $\mathcal{F}$  ermitteln.

<sup>51</sup>Es gibt eine Analogie aus der Mathematik: Dort kann man zu einer gegebenen Menge von Vektoren die sogenannte lineare Hülle und eine Basis dieser linearen Hülle ermitteln. Das entspricht der Hülle und der minimalen Überdeckung in diesem Fall.

**Definition 4.4.1** (Äquivalenz und Hülle von Mengen FAen). Seien  $\mathcal{F}_1$  und  $\mathcal{F}_2$  zwei Mengen funktionaler Abhängigkeiten. Dann sagt man  $\mathcal{F}_2$  folgt aus  $\mathcal{F}_1$  – geschrieben  $\mathcal{F}_1 \Rightarrow \mathcal{F}_2$  – genau dann, wenn jede Relationsinstanz, die alle FAen von  $\mathcal{F}_1$  erfüllt, auch alle FAen von  $\mathcal{F}_2$  erfüllt.

Zwei Mengen funktionaler Abhängigkeiten  $\mathcal{F}_1$  und  $\mathcal{F}_2$  heißen **äquivalent** – geschrieben  $\mathcal{F}_1 \equiv \mathcal{F}_2$  – wenn gilt  $\mathcal{F}_1 \Rightarrow \mathcal{F}_2 \wedge \mathcal{F}_2 \Rightarrow \mathcal{F}_1$ .

Gegeben ist eine Menge  $\mathcal{F}$  von FAen. Dann heißt die Menge  $\mathcal{F}^+$  aller mit Hilfe der Armstrong-Axiome daraus ableitbarer funktionaler Abhängigkeiten die **Hülle** von  $\mathcal{F}$ .

Mit Hilfe des Begriffs der Hülle lässt sich die Äquivalenz von Mengen von FAen anders beschreiben. Es gilt folgender Satz:

**Satz 4.4.1** (Äquivalenz von Mengen von FAen). Seien  $\mathcal{F}_1$  und  $\mathcal{F}_2$  zwei Mengen funktionaler Abhängigkeiten. Dann gilt:

$$\mathcal{F}_1 \equiv \mathcal{F}_2 \quad \Leftrightarrow \quad \mathcal{F}_1^+ = \mathcal{F}_2^+$$

Die Hülle  $\mathcal{F}^+$  einer Menge von FAen kann sehr groß werden. Ihre Berechnung ist von exponentieller Komplexität. Oft ist es aber nur notwendig, eine Frage der Form *Gehört die FA  $X \rightarrow Y$  zu  $\mathcal{F}^+$  oder nicht?*. Dieser sogenannte **Membership-Test** lässt sich bereits in linearer Zeit durchführen. Betrachte dazu folgende Definition:

membership engl. für  
Mitgliedschaft

**Definition 4.4.2** (Hülle einer Attributmenge). Die Hülle  $X^+$  einer Teilmenge  $X \subseteq \mathcal{A}(R)$  bezüglich einer Menge  $\mathcal{F}$  von FAen ist die Menge aller Attribute  $B \in \mathcal{A}(R)$  derart, dass jede Relation, die die FAen  $\mathcal{F}$  erfüllt, auch  $X \rightarrow B$  erfüllt.

Ist die Attributhülle  $X^+$  bekannt, kann die Membership-Frage leicht beantwortet werden.

**Satz 4.4.2.** Die FA  $X \rightarrow Y$  ist genau dann in der Hülle  $\mathcal{F}^+$  einer Menge von FAen enthalten, wenn  $Y \subseteq X^+$  bezüglich  $\mathcal{F}$ .

In Abbildung 37 ist der Algorithmus zur Ermittlung der Attributhülle  $X^+$  bezüglich einer Menge  $\mathcal{F}$  von FAen dargestellt. In der Praxis wird man also so vorgehen, dass man zur Bestimmung der Hülle  $X^+$  mit  $X$  selbst beginnt und alle FAen aus  $\mathcal{F}$  herausucht, deren linke Seite Teilmenge von  $X$  ist. Die entsprechenden rechten Seiten werden mit  $X$  vereinigt und der Schritt solange iteriert, bis sich keine Veränderung der Menge mehr ergibt. Diese Menge ist dann die Attributhülle  $X^+$ .

---

Eingabe:	$(X, \mathcal{F})$ , Attributmenge $X$ und Menge $\mathcal{F}$ von FAen
Ausgabe:	Attributhülle $X^+$
Algorithmus:	$X^+ := X$ <b>repeat</b> $oldX^+ := X^+$ <b>for each</b> $Y \rightarrow Z \in \mathcal{F}$ <b>do</b> <b>if</b> $Y \subseteq X^+$ <b>then</b> $X^+ := X^+ \cup Z$ <b>until</b> $(X^+ = oldX^+)$ <b>return</b> $(X^+)$

---

Abbildung 37: Algorithmus zur Ermittlung der Attributhülle (aus [EN09, S. 315])

### Beispiel 1

Das erste Beispiel beschäftigt sich mit der Ermittlung einer Attributhülle und der Frage, ob sich bestimmte FAen aus der gegebenen Menge von FAen ableiten lassen. Betrachte dazu die Relation  $R$  mit  $\mathcal{A}(R) = ABCDEF$  und den FAen  $\mathcal{F} = \{AB \rightarrow C, D \rightarrow E, BC \rightarrow AD, CF \rightarrow B\}$ .

(a) Ermittle  $AB^+$ :

- $X^+ := AB$
- Für  $AB \rightarrow C$  ist  $AB \subseteq X^+$   
 $\Rightarrow X^+ = AB \cup C = ABC$
- Für  $BC \rightarrow AD$  ist  $BC \subseteq X^+$   
 $\Rightarrow X^+ = ABC \cup AD = ABCD$
- Für  $D \rightarrow E$  ist  $D \subseteq ABCD$   
 $\Rightarrow X^+ = ABCD \cup E = ABCDE$
- Keine weiteren FAen, deren linke Seite Teilmenge von  $X^+$  ist.

Daraus ergibt sich  $AB^+ = ABCDE$ .

(b) Folgt  $AB \rightarrow D$  aus  $\mathcal{F}$ ?

Ja, da  $D \subseteq AB^+$ .

(c) Folgt  $D \rightarrow A$  aus  $\mathcal{F}$ ?

Nein, da  $D^+ = DE$  und  $A \not\subseteq D^+$ .

### Beispiel 2

Im zweiten Beispiel soll für alle nichtleeren Attributmengen die Hülle ermittelt werden und daraus Schlüssel und (echte) Superschlüssel ermittelt werden. Betrachte dazu die Relation  $R$  mit  $\mathcal{A}(R) = ABCD$  und  $\mathcal{F} = \{AB \rightarrow C, C \rightarrow D, D \rightarrow A\}$ .

(a) Berechne die Hüllen aller  $2^4 - 1 = 15$  nichtleeren Teilmengen und ermittle alle sich daraus ergebenden, neuen FAen.

Attributhüllen			
$A^+ = A$	$AB^+ = ABCD$	$ABC^+ = ABCD$	$ABCD^+ = ABCD$
$B^+ = B$	$AC^+ = ACD$	$ABD^+ = ABCD$	
$C^+ = ACD$	$AD^+ = AD$	$ACD^+ = ACD$	
$D^+ = AD$	$BC^+ = ABCD$	$BCD^+ = ABCD$	
	$BD^+ = ABCD$		
	$CD^+ = ACD$		
neue FAen			
$C \rightarrow A$	$AB \rightarrow D$	$ABC \rightarrow D$	
	$AC \rightarrow D$	$ABD \rightarrow C$	
	$BC \rightarrow AD$	$BCD \rightarrow A$	
	$BD \rightarrow AC$		
	$CD \rightarrow A$		

- (b) Bestimme alle Schlüssel von  $R$ .  
Schlüssel(-kandidaten) sind alle minimalen Teilmengen von  $\mathcal{A}(R)$ , deren Hülle  $ABCD$  ergibt. Dies sind  $AB$ ,  $BC$  und  $BD$ .
- (c) Bestimme alle (echten) Superschlüssel von  $R$ .  
Echte Superschlüssel sind alle nicht minimalen Teilmengen von  $\mathcal{A}(R)$ , deren Hülle  $ABCD$  ergibt. Dies sind  $ABC$ ,  $ABD$ ,  $BCD$  und (natürlich)  $ABCD$ .

#### 4.4.2. Minimale Überdeckung

Unter allen Mengen  $\mathcal{F}'$ , die zu einer gegebenen Menge  $\mathcal{F}$  äquivalent sind, ist aus Performanzgründen diejenige gesucht, die in einem gewissen Sinn minimal ist. Diese Überdeckung nennt man auch **mini-male** oder **kanonische Überdeckung**. Eine solche minimale Menge erfüllt drei Bedingungen:

1.  $\mathcal{F}' \equiv \mathcal{F}$ , d. h.  $\mathcal{F}'^+ = \mathcal{F}^+$
2. In  $\mathcal{F}$  existieren keine FAen  $X \rightarrow Y$ , bei denen  $X$  oder  $Y$  überflüssige Attribute enthalten. D. h., entfernt man in der FA  $X \rightarrow Y$  in  $X$  oder  $Y$  ein Attribut, so gilt  $\mathcal{F}'^+ \neq \mathcal{F}^+$ .
3. Jede linke Seite einer funktionalen Abhängigkeit in  $\mathcal{F}'$  ist einzigartig. Dies kann durch sukzessive Anwendung der Vereinigungsregel  $F1$  erreicht werden. Mit ihr wird aus  $X \rightarrow Y$  und  $X \rightarrow Z$  die FA  $X \rightarrow YZ$ .

Zu einer gegebenen Menge  $\mathcal{F}$  von FAen kann die minimale Überdeckung wie folgt bestimmt werden:

1. Führe für jede FA  $X \rightarrow Y \in \mathcal{F}$  die Linksreduktion durch.  
Überprüfe für jedes  $A \in X$ , ob  $A$  überflüssig ist, ob also  $Y$  bereits in der Attributhülle von  $X \setminus \{A\}$  bezüglich  $\mathcal{F}$  enthalten ist. Ist das der Fall, ersetze  $X \rightarrow Y$  durch  $X \setminus \{A\} \rightarrow Y$ .

2. Führe für jede (verbliebene) FA  $X \rightarrow Y$  die Rechtsreduktion durch.  
Überprüfe für jedes  $B \in Y$ , ob  $B$  bereits transitiv durch  $X$  funktional bestimmt ist bzw. ob  $B$  bereits in der Attributhülle von  $X$  bezüglich der FA-Menge  $\mathcal{F}' = \mathcal{F} \setminus \{X \rightarrow Y\} \cup \{X \rightarrow Y \setminus \{B\}\}$  enthalten ist. Ist das der Fall, kann  $B$  auf der rechten Seite weggelassen werden, d. h.,  $X \rightarrow Y$  wird durch  $X \rightarrow Y \setminus \{B\}$  ersetzt.
3. Entferne alle FAen der Form  $X \rightarrow \emptyset$ , die eventuell im 2. Schritt übrig geblieben sind.
4. Fasse alle FAen der Form  $X \rightarrow Y_i$  mit Hilfe der Vereinigungsregel  $F1$  zu einer Regel  $X \rightarrow Y_1 Y_2 \dots Y_n$  zusammen.

### Beispiel 3

Gesucht ist die minimale Überdeckung der Menge  $\mathcal{F} = \{A \rightarrow BD, AC \rightarrow E, CD \rightarrow E, E \rightarrow A, D \rightarrow C\}$  zum Relationenschema  $R(A, B, C, D, E)$ .

Durch Linksreduktion (Schritt 1) wird  $AC \rightarrow E$  zu  $A \rightarrow E$  reduziert, da  $E \in A^+$ . Ebenso wird  $CD \rightarrow E$  zu  $D \rightarrow E$  reduziert, wegen  $E \in D^+$ . Durch Rechtsreduktion (Schritt 2) fällt  $A \rightarrow E$  weg (Schritt 3), weil  $E$  bereits transitiv von  $A$  abhängt ( $A \rightarrow D \wedge D \rightarrow E$ ). Übrig bleibt nach Vereinigung (Schritt 4):  $\mathcal{F} = \{A \rightarrow BD, D \rightarrow CE, E \rightarrow A\}$ .

#### 4.4.3. Software-gestützte Berechnung

Mit Hilfe der in der Vorlesung vorgestellten Software lassen sich die Ergebnisse der vergangenen Beispiele kontrollieren. Sie besteht aus einem von Peter Bartke entwickelten Miranda-Skript und einem Wrapper, der die Funktionalität des Skriptes in Geany zur Verfügung stellt. Nach der Installation kann in Geany eine fa-Datei erstellt werden, die die notwendigen Angaben zu betrachteten Relation und ihren FAen enthält (siehe Abbildung 38). Eine solche

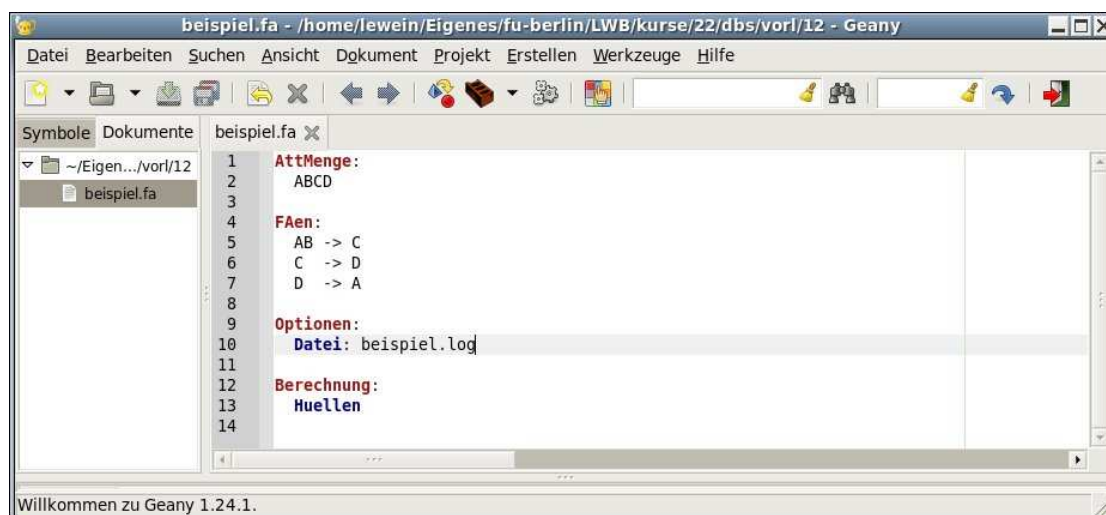


Abbildung 38: fa-Datei in Geany

fa-Datei besteht aus (mindestens) drei Abschnitten:

**AttMenge:**

Hier wird die Attributmenge als Teilmenge von ABCDEFGHI...XYZ notiert. Jeder Buchstabe steht dabei stellvertretend für einen Attributnamen.

**FAen:**

Hier werden die FAen in der Form LS → RS notiert (pro Zeile eine FA). Dabei stellen LS und RS die linke bzw. rechte Seite einer FA dar und werden als Teilmenge von ABCDEFGHI...XYZ notiert. Das Leerzeichen zwischen LS bzw. RS und → ist dabei essentiell.

**Optionen:**

Im Moment gibt es zwei Optionen:

**Format:**

Gibt das Ausgabeformat an. Möglich ist ASCII (Standard) oder TEX.

**Datei:**

Hinter dem Doppelpunkt kann der Name einer Log-Datei angegeben werden, in der die Ausgaben protokolliert werden.

Der Abschnitt **Optionen** kann auch ganz weggelassen werden.

**Berechnung:**

In diesem Abschnitt wird angegeben, was zur genannten Relation berechnet werden soll. Pro Zeile ist genau eines der Schlüsselwörter erlaubt:

**Huellen**

Berechnet die Attributhüllen aller nichtleeren Teilmengen von  $\mathcal{A}(R)$ .

**Schlüssel**

Berechnet alle Schlüssel(kandidaten), Superschlüssel und echten Superschlüssel und zusätzlich die Attributhüllen aller nichtleeren Teilmengen von  $\mathcal{A}(R)$  sowie die neu ableitbaren FAen.

**MinUeberdeckung**

Berechnet die minimale Überdeckung einer Menge von FAen.<sup>52</sup>

<sup>52</sup>In der Literatur findet man auch den Begriff **kanonische Überdeckung** statt minimale Überdeckung.

#### 4.5. Übungszettel A-9

**Aufgabe 9.1.** Gegeben ist das Relationenschema  $R(A, B, C, D, E)$  und eine Menge funktionaler Abhängigkeiten  $\mathcal{F} = \{AB \rightarrow D, E \rightarrow B, CD \rightarrow E, B \rightarrow C\}$ .

- (a) Welche (nichttrivialen) Abhängigkeiten sind aus den gegebenen ableitbar?
- (b) Welche (echten) Superschlüssel, Schlüsselkandidaten bzw. Schlüssel gibt es?
- (c) Von welcher Attributmenge ist BCDE voll funktional abhängig?

**Aufgabe 9.2.** Gegeben sind die beiden Mengen funktionaler Abhängigkeiten  $\mathcal{F}_1 = \{A \rightarrow C, AC \rightarrow D, E \rightarrow AD, E \rightarrow B\}$  und  $\mathcal{F}_2 = \{A \rightarrow CD, E \rightarrow AB\}$  über dem Relationenschema  $R(A, B, C, D, E)$ . Prüfen Sie mithilfe der Armstrong-Axiome (A1-A3) bzw. deren Folgerungen (F1-F4), ob  $\mathcal{F}_1$  und  $\mathcal{F}_2$  äquivalent sind, sich also jeweils die eine Menge aus der anderen ableiten lässt.

**Aufgabe 9.3.** Gegeben ist das Relationenschema  $R(A, B, C, D)$  und die Menge funktionaler Abhängigkeiten  $\mathcal{F} = \{A \rightarrow CD, AC \rightarrow D, BD \rightarrow A, B \rightarrow A\}$ . Ermitteln Sie eine minimale Überdeckung von  $\mathcal{F}$ .

## 4.6. Normalformenlehre

Die zu Beginn des Kapitels angesprochenen Probleme, die beim Einfügen, Ändern oder Löschen entstehen können, basieren auf der falschen Gruppierung von Attributen in Relationen. Ziel dieses Kapitels ist es, Kriterien zu entwickeln, die zu guten Relationsschemata führen, solchen, in denen solche Anomalien nicht mehr auftreten können. Wir werden sehen, dass man mit Hilfe funktionaler Abhängigkeiten zu einer Zerlegung eines Relationenschemas gelangen kann, das sich hinsichtlich von Einfüge-, Änderungs- und Löschoperationen gutmütig verhält.

Dabei muss das Relationenschema in mehrere Relationen zerlegt werden. Wichtig bei einer solchen Zerlegung ist, dass man ohne Verlust von Information wieder zu der Ausgangsrelation gelangen kann. Eine solche Relation nennt man **verlustlos**. Nicht alle Zerlegungen erfüllen diese Eigenschaft, was wir in diesem Kapitel an einem Beispiel belegen werden.

### 4.6.1. Erste Normalform

1NF

Die erste Normalform – oft als 1NF bezeichnet – bedeutet lediglich, dass das Datenbankschema aus *flachen* Relationenschemata besteht, die Attribute also *atomar* sind. In allen behandelten Relationenschemata war das bisher der Fall (und soll auch so bleiben). Das sogenannte *non first normal form model* – auch als NF<sup>2</sup> bezeichnet – löst sich bewusst von dieser Einschränkung und lässt mengenwertige Attribute zu, die als listenwertige Attribute realisiert werden.

NF<sup>2</sup>

Ein solches Relationenschema kann – wie Abbildung 39 zeigt – durch *Flachklopfen* in 1NF überführt werden. Erst durch eine solche Transformation ist es möglich, gezielt einzelne Attributwerte abzufragen, die in NF<sup>2</sup> noch im Mengen zusammengefasst waren. Dabei steigt entsprechend der Mächtigkeit der mengenwertigen Attribute die Anzahl der Tupel, wobei offensichtlich Redundanzen entstehen können, die man durch eine Zerlegung der Relation wieder beheben kann.

ELTERN			ELTERN		
Vater	Mutter	Kinder	Vater	Mutter	Kind
Johann	Martha	{Else, Lucia}	Johann	Martha	Else
Johann	Maria	{Theo, Josef}	Johann	Martha	Lucia
Heinz	Martha	{Cleo}	Johann	Maria	Theo
			Johann	Maria	Josef
			Heinz	Martha	Cleo

Abbildung 39: Eine Relation in NF<sup>2</sup> kann durch *Flachklopfen* in 1NF überführt werden.

### 4.6.2. Zweite Normalform

2NF

Die zweite Normalform 2NF wird von einem Relationenschema verletzt, wenn, einfach gesagt, in der Relation Informationen über mehr als eine Entität bzw. ein Konzept modelliert werden. Jedes Nichtschlüsselattribut darf nur von einem vollständigen Schlüssel(-kandidaten)



abhängen, so dass dieses Attribut ein Fakt ausdrückt, das durch diesen (minimalen) Schlüssel identifiziert wird. Anders gesagt, wenn die Relation einen zusammengesetzten Schlüssel hat, sollen alle anderen Attribute auch wirklich nur von diesem *zusammengesetzten* Schlüssel abhängen und nicht nur von einem Teil davon. Ausnahmen davon sind FAen, bei denen ein Attribut von einem Nicht-Schlüssel-Attribut abhängt. Es spielt hierbei keine Rolle, ob wir vom Schlüssel oder einem Schlüsselkandidaten sprechen, da Schlüsselkandidaten lediglich mehrere mögliche (minimale) Schlüssel bezeichnen.

Formal bedeutet das, dass jedes Nichtschlüssel-Attribut  $A \in \mathcal{A}(R)$  eines Relationenschemas  $R$ , das sich in 2NF befindet, nur entweder *voll funktional* von jedem Schlüssel(-Kandidaten) abhängen darf oder von keinem Schlüsselattribut. Die volle funktionale Abhängigkeit basiert hierbei auf der Zusammensetzung der Attribute des jeweiligen Schlüsselkandidaten – sobald ein Attribut entfernt wäre, ginge auch die FA verloren. Eine Relation mit einattributigem Schlüssel liegt damit automatisch in 2NF vor. Nichtschlüssel-Attribute nennt man auch **nicht-prim** (z. T. auch *nicht-primär*), während man Attribute eines Schlüssels als **prim** (z. T. auch *primär*) bezeichnet.

**Definition 4.6.1** (2. Normalform (2NF)). Ein Relationenschema  $R$  mit der Menge funktionaler Abhängigkeiten  $\mathcal{F}$  ist in zweiter Normalform (2NF), wenn für jede nicht-triviale FA  $X \rightarrow A \in \mathcal{F}$  eine der beiden Bedingungen gilt:

- (a)  $A$  ist prim (also Teil des Schlüssels bzw. eines Schlüsselkandidaten), oder
- (b)  $\nexists \text{Schlüssel(-kandidat)} K : X \subset K$   
(also hängt  $A$  nicht nur von einem Teil von  $K$  ab)

Bemerkungen:

- Die zweite Bedingung lässt sich auch anders formulieren: Nicht-prime Attribute hängen nur voll funktional von einem Schlüssel bzw. -Kandidaten ab.
- In 2NF sind aber sehr wohl zusätzliche FAen erlaubt, in denen die linke Seite – die sogenannte Determinante – ein Nichtschlüssel-Attribut (oder eine Menge von Nichtschlüssel-Attributen) darstellt.

Das folgende Beispiel zeigt, dass bei einer Verletzung von 2NF INSERT-, UPDATE- und DELETE-Anomalien auftreten können. Dieses Beispiel ergibt sich aus dem Join der Relationen HOEREN und STUDENTEN aus dem Uni-Miniwelt-Beispiel (siehe Anhang).

STUDENTENBELEGUNG			
MatrNr	Name	VorlNr	Semester
26120	Fichte	5001	10
27550	Schopenhauer	5001	6
27550	Schopenhauer	4052	6
28106	Carnap	5041	3
28106	Carnap	5052	3
28106	Carnap	5216	3
28106	Carnap	5259	3
...	...	...	...

Der zusammengesetzte Schlüssel dieser Relation ist  $\{MatrNr, VorlNr\}$ , allerdings ergeben sich noch funktionale Abhängigkeiten, welche 2NF verletzen, weil partielle Abhängigkeiten auftreten. Das Attribut *Name* ist nicht nur vom zusammengesetzten Schlüssel  $\{MatrNr, VorlNr\}$  funktional abhängig, sondern auch von nur einem Teil dieses Schlüssels:  $MatrNr \rightarrow Name$ . Ebenso verhält es sich mit  $MatrNr \rightarrow Semester$ .

Die Ausprägungen der Relation STUDENTENBELEGUNG zeigt die (bekannten) schwerwiegenden Anomalien:

#### Einfügeanomalie

Was geschieht mit Studenten, die (noch) keine Vorlesung hören?

#### Änderungsanomalie

Rückt *Carnap* ins vierte Semester vor, müssen vier Tupel synchron geändert werden mit dem Risiko des Verlustes der Datenintegrität.

#### Löschanomalie

Was passiert, wenn Frau *Fichte* die Teilnahme an ihrer einzigen Vorlesung absagt?

Die Lösung des Problems besteht in der Zerlegung der Relation in die Relationen

- HOEREN (*MatrNr, VorlNr*)
- STUDENTEN (*MatrNr, Name, Semester*)

wie dies auch im Universitäts-Beispiel geschehen ist. Hierfür wurden die Attribute der problematische FAen *Name* und *Semester*, die nur von einem Teil des Schlüssels abhängen, in eine neue Relation verschoben und der bestimmende Schlüsselteil zusätzlich hinzugefügt. Die Zerlegung ist verlustlos, alle Informationen der Ausgangsrelation lassen sich über einen Join über das Attribut *MatrNr* wiederherstellen.

### 4.6.3. Dritte Normalform

3NF

In der Praxis wird die zweite Normalform durch die schärfere dritte Normalform 3NF ersetzt, die v. a. Redundanzen und damit Änderungsanomalien minimiert. Hier werden FAen unterbunden, die abseits vom Schlüssel existieren.

**Definition 4.6.2** (3. Normalform (3NF)). Ein Relationenschema  $R$  mit der Menge funktionaler Abhängigkeiten  $\mathcal{F}$  ist in dritter Normalform (3NF), wenn für jede nicht-triviale FA  $X \rightarrow A \in \mathcal{F}$  eine der beiden Bedingungen gilt:

- (a)  $A$  ist prim, oder
- (b)  $X$  ist Superschlüssel

Bemerkungen:

- Im Unterschied zur 2NF sind nun auf der Determinante weder Teilschlüssel, noch Nichtschlüssel-Attribute erlaubt, wenn  $A$  nicht-prim ist.
- Alternativ formuliert heißt das, dass in 3NF kein Nichtschlüssel-Attribut transitiv von einem Schlüssel(-Kandidaten) abhängt. Nichtschlüssel-Attribute dürfen nur noch von Schlüssel(-Kandidaten) bzw. Superschlüsseln abhängen.

Folgendes Beispiel zeigt, wie die Verletzung der 3NF zu Datenredundanz führen kann.<sup>53</sup>

CD			
CDNr	Albumtitel	Interpret	Gründungsjahr
4711	Not That Kind	Anastacia	1999
4712	Wish You Were Here	Pink Floyd	1964
4713	Freak of Nature	Anastacia	1999

Offensichtlich gilt  $CDNr \rightarrow Interpret$ , aber auch  $Interpret \rightarrow Gründungsjahr$ , das Gründungsjahr (von Sonderfällen wie mehrfacher Gründung einmal abgesehen) hängt damit auch transitiv vom Primärschlüssel ab. Probleme können hier durch das Einfügen oder Ändern eines Tupels zu einem vorhandenen Interpreten entstehen: Das Gründungsjahr wird redundant gespeichert, bei Fehlern droht Verlust der Integrität. Eine Transformation in die 3NF besteht wie bereits bei der 2NF darin, die Relation zu zerlegen: Die problematischen, transitiv abhängigen Attribute werden in eine neue Relation gebracht und der entsprechende Schlüssel hinzugefügt. Damit würde das Beispiel zerlegt werden in:

- CD ( $CDNr$ , Albumtitel, Interpret)
- INTERPRET ( $Interpret$ , Gründungsjahr)

Als Merkspruch dient ein (im angloamerikanischen Bereich geläufiger und witziger) Merkspruch. Er lehnt sich an den amerikanischen Gerichtseid *The truth, the whole truth and nothing but the truth. So help me god!* an und lautet:<sup>54</sup>

*The key, the whole key and nothing but the key. So help me Codd!*<sup>55</sup>

Das bedeutet:<sup>56</sup>

<sup>53</sup>Quelle: [de.wikipedia.org/wiki/Normalisierung\\_\(Datenbank\)](https://de.wikipedia.org/wiki/Normalisierung_(Datenbank)) (07.12.2014)

<sup>54</sup>Quelle: [de.wikipedia.org/wiki/Normalisierung\\_\(Datenbank\)](https://de.wikipedia.org/wiki/Normalisierung_(Datenbank)) (07.12.2014)

<sup>55</sup>zu deutsch: *Der Schlüssel, der ganze Schlüssel und nichts als der Schlüssel. So wahr mir Codd helfe!*

<sup>56</sup>Anmerkung: Dieser Merkspruch mit Bezug auf *den Schlüssel* ist im Grunde etwas verkürzt: man muss die Normalformen 1-3 auf alle Schlüsselkandidaten beziehen, sonst ergibt die Boyce-Codd-Normalform gar keinen Sinn und führt zu völlig sinnlosen Tabellenaufteilungen! In der Praxis weisen allerdings sehr viele Tabellen außer dem Primärschlüssel gar keine weiteren Schlüsselkandidaten auf, und die Formulierung bringt das Wesentliche sehr einprägsam auf den Punkt.

- alle (impliziert: atomaren) Werte beziehen sich auf den Schlüssel (*the key*) (1NF)
- bei zusammengesetzten Schlüsseln beziehen sie sich jeweils auf den gesamten Schlüssel (*the whole key*) (2NF)
- die Werte hängen nur vom Schlüssel ab, und nicht von weiteren Werten (*nothing but the key*) (3NF)

#### 4.6.4. Boyce-Codd-Normalform

BCNF

Die Boyce-Codd-Normalform wurde von Raymond F. Boyce und Edgar F. Codd, dem Entwickler des relationalen Datenmodells, entworfen. Sie fanden heraus, dass sich die im Zusammenhang mit der 3NF angesprochene Einfügeanomalie mit dem Problem redundanter Daten besser vermeiden lässt, wenn man eine noch schärfere Bedingung an die Relation stellt. In der 3NF kann es vorkommen, dass mehrere Schlüsselkandidaten vorhanden sind, die sich "überlappen", sprich, dass ein Teil eines Schlüsselkandidaten funktional abhängig ist von einem Teil eines anderen Schlüsselkandidaten. Die Verschärfung vermeidet diese wechselseitige Abhängigkeit von Teilen von Schlüsselkandidaten.

**Definition 4.6.3** (Boyce-Codd-Normalform (BCNF)). *Ein Relationenschema  $R$  mit der Menge funktionaler Abhängigkeiten  $\mathcal{F}$  ist in Boyce-Codd-Normalform (BCNF), wenn für jede nichttriviale FA  $X \rightarrow A \in \mathcal{F}$  gilt, dass  $X$  ein Superschlüssel ist.*

Bemerkung: Im Unterschied zur 3NF muss nun **jede** Determinante einer nichttrivialen FA ein Superschlüssel sein.

Als Beispiel diene die Relation

STÄDTE			
Ort	Bundesland	MinisterpräsidentIn	Einwohnerzahl

Offensichtlich gibt es zwei Schlüsselkandidaten:

- $\{\text{Ort}, \text{Bundesland}\}$
- $\{\text{Ort}, \text{MinisterpräsidentIn}\}$

Zudem gelten offensichtlich die folgenden FAen:

- $\text{Bundesland} \rightarrow \text{MinisterpräsidentIn}$
- $\text{MinisterpräsidentIn} \rightarrow \text{Bundesland}$

Im ganzen Beispiel soll einmal von allerlei Sonderfällen abgesehen werden, wie z. B. zwei Ministerpräsidenten mit gleichem Namen o. ä.

Da in (a) und (b) die rechten Seiten Primattribute sind, ist die Relation in 3NF. Diese beiden FAen verletzen hier aber die BCNF, da weder die FA in (a), noch die FA in (b) einen Superschlüssel auf der linken Seiten hat. Diese Verletzung führt beim Einfügen von Städten zu redundanter Speicherung der Information, welches Bundesland von welchem/welcher Ministerpräsident/in regiert wird. Dies kann bei Änderungen, z. B. bei Neuwahlen, zu Anomalien führen.

#### 4.6.5. Verlustlose Zerlegung

Die Herstellung der genannten 2NF, 3NF und BCNF laufen in der Regel auf eine Zerlegung der Ursprungsrelation hinaus (s. S. 4–20). Die Zerlegung einer Relation stellt ganz allgemein eine Art Neugruppierung der Attribute dar, und zwar dergestalt, dass die Attributmenge jeder Teilrelation eine Teilmenge der Attributmenge der Ausgangsrelation ist und jedes Attribut der Ausgangsrelation in einer der Teilrelationen erfasst wird. Es geht also kein Attribut verloren, dafür können Attribute aber in mehreren Relationen auftauchen, v. a. wenn sie als Schlüsselattribut dienen. Dennoch gibt es einige Fallstricke zu beachten, wie folgendes Beispiel zeigt:

Man betrachte als Beispiel die Relation **BIERTRINKER**. Sie erfasst eine Zuordnung von Person, Kneipe und Biersorte, die dort jeweils getrunken wird.<sup>57</sup>

BIERTRINKER		
Kneipe	Gast	Bier
Kowalski	Kemper	Pils
Kowalski	Eickler	Hefeweizen
Innsteg	Kemper	Hefeweizen

Nun wird diese Relation in die beiden Teilrelationen **BESUCHT** und **TRINKT** zerlegt.

BESUCHT		TRINKT	
Kneipe	Gast	Gast	Bier
Kowalski	Kemper	Kemper	Pils
Kowalski	Eickler	Eickler	Hefeweizen
Innsteg	Kemper	Kemper	Hefeweizen

Die Informationen dieser beiden Relationen sollen nun mit einem Natural Join miteinander verbunden werden. Der Join über das Attribut *Gast* liefert jedoch mehr Tupel (rot markiert), als vorher vorhanden waren.

BIERTRINKER		
Kneipe	Gast	Bier
Kowalski	Kemper	Pils
Kowalski	Kemper	Hefeweizen
Kowalski	Eickler	Hefeweizen
Innsteg	Kemper	Pils
Innsteg	Kemper	Hefeweizen

<sup>57</sup>Das Beispiel ist [KE09, S. 189] entnommen.

Die zusätzlichen Tupel führen zu einem Informationsverlust, was auf den ersten Blick merkwürdig erscheint. Es spiegelt allerdings die Tatsache wieder, dass Zuordnungen verloren gingen. Gewünscht ist ein Verhalten, bei dem *keine* zusätzlichen Tupel entstehen, wenn man die Teilrelationen wieder zusammenfügt. Daraus ergibt sich eine Definition für die verlustlose Zerlegung.

**Definition 4.6.4** (Verlustlose Zerlegung). *Eine Zerlegung  $\{R_1, R_2, \dots, R_n\}$  einer Relation  $R$  mit  $\mathcal{A}(R_i) \subseteq \mathcal{A}(R)$  heißt verlustlos, wenn für alle Inkarnationen  $r \in R$  gilt*

$$r = r_1 \bowtie r_2 \bowtie \dots \bowtie r_n$$

*mit  $r_i = \pi_{\mathcal{A}(R_i)}(r)$  und außerdem kein Tupel durch den Join entsteht, das nicht in  $R$  enthalten war.*

#### 4.6.6. Zerlegung in BCNF

Die BCNF hat alle zu Beginn des Kapitels genannten *schönen* Eigenschaften und kann daher auf Wunsch als Standard-Normalform realisiert werden. Dazu ist es meist nötig, eine gegebene Relation – das kann auch zu Beginn die Menge aller Attribute einer Datenbank sein – so zu zerlegen, dass alle Relationen in BCNF vorliegen. Für diese Zerlegung existiert ein Algorithmus, der in diesem Kapitel vorgestellt wird. In diesem Zusammenhang soll auf die Frage der verlustlosen Zerlegung noch einmal genauer eingegangen werden.

Zuerst jedoch wollen wir jedoch einen besonderen Typ von Relationsschema untersuchen, der *immer* in BCNF ist.

**Satz 4.6.1** (Besonderheit zweiattributiger Relationenschemata). *Jede 2-attributige Relation ist in BCNF.*

Beweis: Sei  $\mathcal{A}(R) = \{A, B\}$  die Attributmenge der Relation.

**Fall 1:** *Es existieren keine nicht-trivialen FAen. Dann ist  $\mathcal{F} \subseteq \{A \rightarrow A, B \rightarrow B, AB \rightarrow A, AB \rightarrow B\}$ . Hier ist  $AB$  Schlüssel, die Relation automatisch in BCNF, da keine Bedingung zu prüfen ist.*

**Fall 2:**  *$A \rightarrow B \wedge B \not\rightarrow A$ . In diesem Fall ist  $A$  der (einzige) Schlüssel. Jede nicht-triviale FA hat  $A$  als Determinante,  $R$  ist in BCNF.*

**Fall 3:**  *$B \rightarrow A \wedge A \not\rightarrow B$ . Analog zu Fall 2 (Symmetrie).*

**Fall 4:**  *$A \rightarrow B \wedge B \rightarrow A$ . Sowohl  $A$  als auch  $B$  sind Schlüssel, alle FAen haben einen Schlüsselkandidaten als Determinante,  $R$  ist in BCNF.*

Nun wollen wir uns dem Problem der (verlustlosen) Zerlegung einer BCNF-verletzenden Relation  $R$  in Teilrelationen  $R_i$  zuwenden.

Ziel ist es, eine BCNF-verletzende Relation so zu zerlegen, dass alle Teilrelationen BCNF erfüllen und sich aus ihnen die ursprüngliche Relation verlustlos wiederherstellen lässt. Die

Idee dabei ist, sich diejenigen FAen herauszupicken, die BCNF verletzen und die Relation entlang dieser FA zu zerlegen. Das führt man solange durch, bis keine BCNF-verletzende FA mehr vorhanden ist. Dabei können allerdings FAen verloren gehen. In diesem Fall belässt man die BCNF-verletzende Relation in 3NF.

Der Algorithmus in Abbildung 40 zeigt das Vorgehen der Zerlegung in BCNF-konforme Teilrelationen im Detail. Es ist auch möglich, diesen Algorithmus auf andere gewünschte Normalformen anzuwenden, man tausche BCNF jeweils durch 2NF oder 3NF aus.

Ausgangslage:	Relation $R$ und Menge von FAen $\mathcal{F}$
Ausgabe:	Zerlegung von $R$ in $R_1, R_2, \dots, R_n$ mit $R_i$ in BCNF.
Algorithmus:	<ol style="list-style-type: none"> <li>(1) Ist <math>R</math> in BCNF, dann Ausgabe von <math>R</math> und Ende.</li> <li>(2) Sei <math>X \rightarrow Y</math> eine BCNF-verletzende FA. Berechne <math>X^+</math>. Zerlege <math>R</math> in <math>R_1</math> und <math>R_2</math> mit <math>\mathcal{A}(R_1) = X^+</math> und <math>\mathcal{A}(R_2) = X \cup (\mathcal{A}(R) \setminus X^+)</math>.</li> <li>(3) Bestimme die <math>\mathcal{F}\mathcal{A}</math>-Mengen zu <math>R_1</math> und <math>R_2</math> und erstelle daraus die neuen Mengen <math>\mathcal{F}_1</math> und <math>\mathcal{F}_2</math>.</li> <li>(4) Fahre rekursiv mit <math>R_1, \mathcal{F}_1</math> und <math>R_2, \mathcal{F}_2</math> mit neuen Indizes fort.</li> </ol>

Abbildung 40: Verlustlose Zerlegung einer Relation  $R$  mit der Attributmenge  $\mathcal{A}(R)$  und der Menge  $\mathcal{F}$  funktionaler Abhängigkeiten in BCNF-erfüllende Teilrelationen.

### Beispiel

Wir betrachten noch einmal das Beispiel 2 von Seite 4–9. Aus der Relation  $R(A, B, C, D)$  mit der Menge  $\mathcal{F} = \{AB \rightarrow C, C \rightarrow D, D \rightarrow A\}$  ergeben sich mit den Schlüsselkandidaten  $AB$ ,  $BC$  und  $BD$  die BCNF-verletzenden FAen

$$C \rightarrow A \quad C \rightarrow D \quad D \rightarrow A \quad AC \rightarrow D \quad CD \rightarrow A$$

weil in BCNF die Determinanten nur (Super-)Schlüssel sein dürfen. Betrachte  $C \rightarrow D$ . Wegen  $C^+ = ACD$  ist der erste Schritt

$$R_1 = (A, C, D) \quad R_2 = (B, C)$$

mit den FAen  $\mathcal{F}_1 = \{D \rightarrow A, C \rightarrow D\}$  und dem Schlüssel  $C$ .  $R_2$  ist bereits in BCNF, da die Relation zweiattributig ist.  $R_1$  muss dagegen weiter zerlegt werden, weil  $D \rightarrow A$  quer steht.

Wegen  $D^+ = AD$  ist der zweite Schritt

$$R_{11} = (A, D) \quad R_{12} = (C, D)$$

Man erkennt allerdings schon nach dem ersten Zerlegungsschritt, dass Abhängigkeiten verloren gehen. Nicht immer ist die BCNF nämlich abhängigkeiterhaltend. Hier lässt sich die FA  $AB \rightarrow C$  nicht erhalten. In solchen Fällen nimmt man davon Abstand, die Relation in BCNF-erfüllende Teilrelationen zu zerlegen und begnügt sich mit 3NF. Die Zerlegung in 3NF ergibt in diesem Fall  $R = \{ABC, AD, CD\}$ .

## Literatur zu Kapitel 4

- [EN09] ELMASRI, Ramez A. ; NAVATHE, Shamkant B.: *Grundlagen von Datenbanksystemen*. 3. Auflage. München : Pearson-Studium, 2009. – ISBN 978-3-8689-4012-1
- [KE09] KEMPER, Alfons ; EICKLER, André: *Datenbanksysteme: Eine Einführung*. 7. Auflage. München : Oldenbourg Wissenschaftsverlag, 2009. – ISBN 978-3-486-59018-0



## 4.7. Übungszettel A-10

**Aufgabe 10.1.** Gegeben ist der folgende Ausschnitt der Extension einer Relation (*K* Kunden, *T* Teile):

LIEFERUNGEN						
KNr	KName	KStadt	KTel	TNr	Datum	Anzahl
1	Date	Paris	2270123	1	3.12.	10
1	Date	Paris	2270123	2	13.12.	20
2	Ullman	London	2134256	1	8.12.	20
3	Codd	Paris	3241789	2	9.12.	10

- Geben Sie eine Reihe von funktionalen Abhängigkeiten an, die gelten bzw. nicht gelten. Welche eignet sich als Schlüssel? Machen Sie dazu notwendige Annahmen über die zugrundeliegende Welt.
- Welche Attribute hängen von welchen voll funktional ab?
- Beschreiben Sie die Effekte, die auftreten können, wenn
  - sich Dates Telefonnummer ändert,
  - Kunde Korth, Wien, Tel. 42921789 eingefügt werden soll,
  - die Transaktion vom 8.12. gelöscht werden soll.
- Spalten Sie die Relation so auf, dass keine Teilschlüsselabhängigkeiten mehr vorhanden sind und zeigen Sie, dass die geschilderten Anomalien hier nicht mehr auftreten können.
- Die Kunden erhalten städtische Vertriebsbeauftragte zugeordnet, Date und Codd den Schmidt und Ullman den Schulz. Sie werden in die normalisierte Relation (d) eingefügt. Wie sehen nun die funktionalen Abhängigkeiten aus?
- Kann der Vertriebsbeauftragte Schmidt ohne Probleme durch Müller ersetzt werden? Kann ein Vertriebsbeauftragter für Berlin eingetragen werden? Was passiert, wenn Ullman gelöscht wird?
- Welche indirekte (transitive) Abhängigkeit gibt es in der erweiterten Relation?
- Durch welche Zerlegung kann die transitive Abhängigkeit beseitigt werden?

**Aufgabe 10.2.** Für das Relationenschema

$R_1(\underline{A}, C, D, E)$

$R_2(\underline{A}, B, C, F)$

$R_3(\underline{E}, C, E)$

ist eine Menge funktionaler Abhängigkeiten

$$\mathcal{F} = \{AB \rightarrow F, AC \rightarrow D, B \rightarrow C, D \rightarrow E, F \rightarrow C, F \rightarrow E\}$$

gegeben. Begründen Sie, ob die Relationen in zweiter bzw. dritter Normalform sind. Konvertieren Sie ggf. das Relationenschema durch Zerlegung in die dritte Normalform.

**Aufgabe 10.3.** Gegeben sei ein Warenlager, in dem in numerierten Behältern Teile aufbewahrt werden. In den Behältern kommt immer nur eine Teileart vor. Gespeichert wird die Anzahl der Teile pro Behälter, die Lieferzeit der damit verbundenen Produkte und die Wiederbeschaffungszeit der Teile. Eine Beispielausprägung sei etwa:

LAGER				
BNr	TNr	Anzahl	LZeit	WZeit
1	1	5	6	1
2	1	8	6	1
3	2	6	10	2
4	2	9	10	2

- Ist die Relation in 2NF?
- Konstruieren Sie je eine Änderungs-, Einfüge- und Löschanomalie.
- Geben Sie die funktionalen Abhängigkeiten für die Lagerrelation an.
- Welche Zerlegung der Relation beseitigt die Anomalien?
- Führen Sie die Zerlegung mit den obigen Daten durch.

**Aufgabe 10.4.** Gegeben sei die Ausprägung einer Relation  $R$  mit  $\mathcal{A}(R) = \{A, B, C\}$ .

R		
A	B	C
5	x	y
5	z	u
6	v	y
7	t	s
8	x	y
9	t	s

- Prüfen Sie unter der Annahme, dass die Ausprägung die Abhängigkeitsverhältnisse für alle Zeitpunkte wiedergibt, ob die folgenden Abhängigkeiten in  $R$  erfüllt sind:  
 $(i) A \rightarrow B$     $(ii) B \rightarrow C$     $(iii) C \rightarrow B$     $(iv) B \rightarrow A$     $(v) C \rightarrow A$   
 Begründen Sie unter Angabe des verletzenden Tupels, falls eine Abhängigkeit nicht gilt.
- Hat  $R$  einen Schlüsselkandidaten? Wenn ja, welchen? Wenn nein, warum nicht?

## 5. SQL

Im Jahre 1970 schrieb E. F. Codd seine inzwischen klassische Arbeit mit dem Titel 'A Relational Model of Large Shared Data Banks' [CACM, 13(6), 1970]. Hierin legte er die theoretische Grundlage für das relationale Datenmodell. Alle weiteren Entwicklungen basieren im wesentlichen auf dieser Arbeit.<sup>58</sup> In der Folgezeit entstand eine Vielzahl von Produkten, die auf der Grundlage des relationalen Modells Datenbanksysteme realisierten. Um Zugang zu den in Tabellen gehaltenen Daten zu bekommen, war es nötig, eine Sprachschnittstelle zu entwickeln. 1974 entstand in den IBM-Laboratorien San Jose (Chamberlain et. al.) die Sprache SEQUEL, aus der sich SEQUEL/2 in den Jahren 1976/77 entwickelte. Parallel dazu wurde, wieder von IBM, ein Prototyp eines Datenbanksystems, 'System R' entwickelt, das nicht nur bei IBM sondern auch in einigen Kundenumgebungen installiert wurde. Durch den Erfolg von System R wurde klar, dass IBM früher oder später kommerzielle Produkte vertreiben würde, die SQL<sup>59</sup> als Sprachkomponente benutzten. So kam es, dass Konkurrenten versuchten, mit marktfähigen Datenbank-Produkten eher als der blaue Riese herauszukommen, dies gelang ORACLE. IBM verkaufte dann seit 1981 ein Produkt 'SQL/DS', das erst unter DOS/VSE, später auch für VM/CMS und MVS installiert war. Das unter MVS installierte Produkt erhielt schließlich den Namen DB2.

Im Jahre 1982 stellte das Amerikanische Institut für Nationale Standardisierung ein ANSI-Gremium zusammen, dessen Vorschlag zur Normierung von SQL schließlich 1986 ratifiziert wurde. Der Vorschlag war mehr oder weniger entlang der Haupt-IBM-Produktlinie entworfen worden. Schließlich übernahm die Internationale Standard-Organisation (ISO) den ANSI-Standard und SQL1 (SQL-1989) war definiert.

Das Standardisierungsgremium hat seither unter intensiver Beteiligung von Forschung und Industrie weiter getagt. Entstanden sind folgende SQL-'Meilensteine':

SQL-89	SQL1	Erste Spezifikation
SQL-92	SQL2	JOINs, EXCEPT, INTERSECT
SQL:1999	SQL3	Core-SQL, Objektrelationale Konzepte, BLOBs, CLOBs, ARRAYs.
SQL:2003	–	BIGINT, MULTISSET, Sequenzen, XML
SQL:2008	–	Modulkonzept
SQL:2011	–	Temporäre Datenbasen

Kein relationales Datenbanksystem implementiert exakt den gesamten SQL-Standard. Vielmehr gibt es einen Wildwuchs von mehr oder weniger standardkonformen Teilimplementierungen. Die SQL-Sprachkomponenten von DB2 und Oracle dienten zum Beispiel immer wieder als Referenz-Implementierungen für später im Standard normierte Konzepte.

Die SQL-Standard-Spezifikation ist ein vielbändiges Werk mit einer Reihe von Anhängen. Sprachanbindungen sind definiert zu nahezu allen Programmiersprachen.

<sup>58</sup>Bereits 1969 war allerdings in Deutschland das DB-System ADABAS erschienen, bei dessen Implementierung – ohne durchgängige theoretische Fundierung – relationale Konzepte verwirklicht waren.

<sup>59</sup>SQL wird offiziell 'ess-k(j)uh-ell' gesprochen, von manchen aber weiterhin nach der ersten Prototypimplementierung 'sequel' ('structured english query language'). SQL ist heute die Abkürzung für 'Standard Query Language'.

## 5.1. Übersicht zum Sprachkern von SQL

Man unterteilt die SQL-Anweisungen<sup>60</sup> in solche zur

- (a) *Datendefinition*: Data Definition Language *DDL*,
- (b) *Datenmanipulation*: Data Manipulation Language *DML*, und
- (c) *Datenkontrolle*: Data Control Language *DCL*.

Daneben gibt es noch eine ganze Reihe von Anweisungen, die für die Benutzerverwaltung, der damit verbundenen Rechtevergabe und dem Verhalten des DBMS unter dem Aspekt der Performanz zuständig sind. Eine gute Übersicht – bei bereits vorhandenem Grundwissen – findet man in [Eis05].

### 5.1.1. Datendefinition (DDL)

Ähnlich wie in Programmiersprachen sind für die Datendefinition in DBMS zum einen Deklarationsanweisungen notwendig, die entsprechende Strukturen – hier nicht im Speicher sondern im Datenbanksystem – erzeugen, zu deren Formulierung die Kenntnis von Datentypen notwendig ist.

#### Erzeugen und Löschen von Datenbankstrukturen

Syntax	Erläuterung
CREATE DATABASE <i>Datenbankname</i> DROP DATABASE <i>Datenbankname</i>	Erzeugt einen Satz leerer Systemtabellen Löschen der Systemtabellen (auf Betriebssystemebene: createdb, dropdb)
CREATE SCHEMA <i>schemaname</i> AUTHORIZATION <i>Nutzer</i> DROP SCHEMA <i>schemaname</i>	Namensraum für Tabellen, Datentypen und Funktionen (Modularisierung) Schema entfernen.
CREATE TABLE <i>Tabellenname</i> ( <i>Spaltenname</i> <i>Datentyp</i> [ NOT NULL   ... ] { , <i>Spaltenname</i> <i>Datentyp</i> [ NOT NULL   ... ] } [ CONSTRAINT <i>Name</i> PRIMARY KEY(...) ] ) DROP TABLE <i>Tabellenname</i> ALTER TABLE <i>Tabellenname</i> ADD ( <i>Spaltenname</i> <i>Datentyp</i> [ NOT NULL   ... ] { , <i>Spaltenname</i> <i>Datentyp</i> [ NOT NULL   ... ] }	Erzeugung von Tabellenstrukturen Löschen von Tabellenstrukturen Änderung von Tabellenstrukturen
CREATE INDEX <i>Indexname</i> ON <i>Tabellenname</i> ( <i>Spaltenname</i> ) DROP INDEX <i>Indexname</i>	Indexerzeugung (Nicht Standard SQL) Löschen eines Index
CREATE DOMAIN <i>Domainname</i> DROP DOMAIN <i>Domainname</i>	Erzeugung eines Typs (SQL:1999) Löschen eines Typs

<sup>60</sup>Die hier verwendete SQL-Ausprägung ist PostgreSQL C, das den üblichen Standards der meistverwendeten SQL-Sprachen entspricht.

## Typen

Die Sprachbereiche werden durch ein *Typsystem* von elementaren und benutzerdefinierten Typen unterstützt. Im Typsystem unterscheiden sich die Implementierungen gängiger Datenbanksysteme am meisten.<sup>61</sup> Zur Deklaration der Typen der Attribute kann u. a. man aus folgenden Datentypen wählen:

Standard	Beispiel/Bemerkung
CHARACTER [ ( <i>länge</i> ) ]	CHAR (25)
NUMERIC [ ( <i>stellenzahl</i> [ , <i>nachkomma</i> ] ) ]	NUMERIC (7,2), NUMERIC (6),
FLOAT [ ( <i>stellenzahl</i> ) ]	FLOAT (23)
INTEGER	auch INT
SMALLINT	−32768 bis 32767 (2 Bytes)
DOUBLE PRECISION	auch FLOAT8
REAL	Fließkommazahl mit 6 Nachkommastellen
BOOLEAN	auch BOOL
CHARACTER VARYING ( <i>länge</i> )	VARCHAR, VARCHAR(20)
CHARACTER	CHAR
TEXT	keine Längenbeschränkung
DATE	
TIME	
TIMESTAMP	Datum und Uhrzeit
INTERVAL	Zeitspanne
BIT	
BIT ( <i>Länge</i> )	BIT (255)
BIT VARYING ( <i>Länge</i> )	BIT VARYING (100)
[ARRAY]	INT[]

Hierbei sind *stellenzahl* und *nachkomma* die Anzahl der gültigen Stellen und die Anzahl der Nachkommastellen, das Maximum der *stellenzahl* ist implementierungsabhängig.

## Beispiele

In den nachfolgenden Beispielen werden SQL-Token stets in Großbuchstaben notiert, Bezeichner für Datenbanken, Schemas, Tabellen, Attribute etc. in gemischter Schreibweise, obwohl in SQL – sofern es nicht in einfachen Anführungszeichen notiert ist – die Tabellennamen *Buecher*, *BUECHER* und *buecher* identisch sind und als *buecher* aufgefasst werden.

### (a) Tabellen: Bücher

<sup>61</sup>Beispielsweise unterstützt *MySQL* keinen **BOOLEAN**-Typ; dieser wird mithilfe eines nicht Standard-konformen Typs *Bit* nachgebildet (Stand: 30.12.2014). Eine sehr umfangreiche Gegenüberstellung der Unterschiede verschiedener SQL-Implementierungen findet man im weltweiten Netz unter [troels.arvin.dk/db/rdbms](http://troels.arvin.dk/db/rdbms) (Stand: 03.01.2015).

```
CREATE TABLE Buecher (  
    ISBN          CHAR(13) NOT NULL,  
    Titel         CHAR(64),  
    VName         CHAR(32),  
    Jahr          INTEGER,  
    Seiten        INTEGER,  
    Preis         NUMERIC(5,2) )
```

Innerhalb der Klammern der CREATE TABLE-Anweisung werden die Tabellenspalten mit Attributname und -typ spezifiziert. Wird NOT NULL zusätzlich angegeben, *muss* für jedes Tupel in dieser Spalte ein Wert existieren, wird UNIQUE spezifiziert, sind für dieses Attribut nur *verschiedene* Werte zugelassen. Beides wird bei der Aufnahme der Tupel in die Tabelle überprüft. Am Ende der Kommaliste einer Tabellendefinition werden Beschränkungen – Primärschlüssel oder Fremdschlüssel – spezifiziert. Damit ist in vielen Implementierungen die Erzeugung eines Index verbunden.

(b) Typ mit Wertebereich

```
CREATE DOMAIN Geschlecht  
    AS CHAR  
    DEFAULT 'W'  
    CHECK (VALUE IN ('M', 'W', 'D')) )
```

mit anschließender Verwendung in einer Tabellendefinition:

```
CREATE TABLE Angestellte (  
    Name  VARCHAR(40),  
    Gen   Geschlecht )
```

(c) Index

```
CREATE INDEX KundenIndex  
    ON Kunden (Email ASC)
```

Erzeugt einen aufsteigenden (ascending) Index auf der Spalte Email in der Tabelle Kunden.

### 5.1.2. Datenmanipulation (DML)

#### INSERT

Nach dem Standard werden Tupel mit folgender Operation in eine Tabelle eingetragen:

```
INSERT INTO Tabellename [ (Kommaliste der Spalten) ]  
VALUES (Liste der Werte) | SELECT-Anweisung
```

Mit dem Schlüsselwort VALUES können einzelne Zeilen eingefügt werden.

```
INSERT INTO Autor (ANr, AName, AVorname)  
VALUES(156, 'Ullman', 'Jeffrey');  
INSERT INTO Autor (ANr, AName, AVorname)  
VALUES(167, 'Widom', 'Jennifer');
```

Alternativ wird durch eine SELECT-Anweisung das Anfrageergebnis in die Tabelle übernommen.

```
INSERT INTO TEMP (ISBN, Verlag)
  SELECT B.ISBN, B.VName
  FROM   Buecher AS B
  WHERE  B.Preis < 40
```

Die zu INSERT symmetrische Löschoperation ist DELETE:

DELETE

```
DELETE FROM Tabellename
[ WHERE Suchbedingung ]
```

Lässt man die WHERE-Klausel weg, so werden *alle* Zeilen gelöscht; sonst nur solche, die die Suchbedingung erfüllen.

Um einzelne Spaltenwerte zu verändern, eine Anforderung, die in Anwendungen häufig ausgeführt werden muss, verwendet man die UPDATE-Operation:

UPDATE

```
UPDATE Tabellename
  SET Spaltenname = Ausdruck { , Spaltenname = Ausdruck }
[ WHERE Suchbedingung ]
```

Im nachfolgenden Beispiel erfahren alle Bücher des Addison-Wesley-Verlages eine Preiserhöhung um 10%:

```
UPDATE Buecher
  SET  Preis = Preis*1.1
  WHERE Verlag = 'Oldenbourg'
```

Nahezu alle DBMS-Anbieter stellen eine Operation bzw. ein Dienstprogramm zur Verfügung, um aus Dateien (z. B. CSV-Dateien) Daten in bereits erzeugte Relationsschemata zu übernehmen. Diese Anweisung (bzw. Dienstprogramm) heißt in PostgreSQL COPY.

COPY

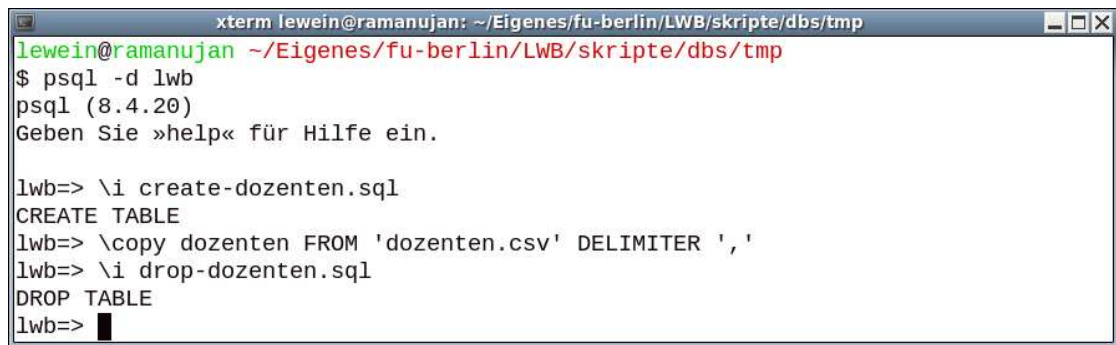
```
COPY Tabellename FROM 'Dateiname'
[ DELIMITER ',' ]
```

Damit werden kommagrenzte Tabellenzeilen aus einer Datei in eine Relation importiert (für *portable* Datenimporte von Beispieldaten verwende man besser den INSERT-Befehl). Diese Variante ist nur für Superuser möglich, die Version \copy mit vorangestelltem Backslash funktioniert mit gleicher Syntax auch für den Benutzer, der INSERT-Rechte auf der Tabelle besitzt.

\copy

Ebenso können die Daten einer Relation in eine Datei exportiert werden.

```
COPY (Tabellenabfrage) TO 'Dateiname' format csv;
(z. B. COPY (SELECT * FROM Tabelle) TO 'Dozenten.csv' format csv;)
```



```
xterm lewein@ramanujan: ~/Eigenes/fu-berlin/LWB/skripte/dbs/tmp
lewein@ramanujan ~/Eigenes/fu-berlin/LWB/skripte/dbs/tmp
$ psql -d lwb
psql (8.4.20)
Geben Sie »help« für Hilfe ein.

lwb=> \i create-dozenten.sql
CREATE TABLE
lwb=> \copy dozenten FROM 'dozenten.csv' DELIMITER ','
lwb=> \i drop-dozenten.sql
DROP TABLE
lwb=> █
```

### 5.1.3. Datenkontrolle (DCL) – die SELECT-Anweisung

In SQL ist die SELECT-Anweisung die wesentliche Datenkontrollanweisung. Nicht umsonst ist ihr in jedem SQL-Buch der längste Abschnitt gewidmet. Die SELECT-Anweisung in der heutigen Form vereinigt relationenalgebraische Elemente mit Konzepten der Tupelkalküls.

Ein Satz erzeugender Grundoperationen der Relationenalgebra sind Vereinigung, Differenz, kartesisches Produkt, Selektion und Projektion. Daraus sind z.B. die Operationen natürlicher Verbund,  $\theta$ -Verbund und Division ableitbar.

**Vereinigung.** Die Vereinigung zweier vereinigungskompatibler Relationen  $r$  und  $s$  wird in SQL als UNION zweier SELECT-Anfragen formuliert:

```
SELECT * FROM r
UNION
SELECT * FROM s
```

**Differenz.** Seit SQL-92 gibt es Durchschnitt und Differenz direkt:

```
SELECT * FROM r
INTERSECT
SELECT * FROM s
```

```
SELECT * FROM r
EXCEPT
SELECT * FROM s
```

Vorher war die Mengendifferenz zweier Relationen  $r$  und  $s$  ist nur mit einer komplizierten SELECT-Konstruktion definierbar:  $r-s$ , die Menge aller Tupel von  $r$ , die nicht in  $s$  sind (wobei  $r$  und  $s$  vereinigungskompatibel sind) lässt sich z.B. mit Hilfe des EXISTS-Prädikats und einer Unteranfrage darstellen:

```
SELECT *
FROM r
WHERE NOT EXISTS (
  SELECT *
  FROM s
  WHERE r.r1 = s.s1 AND ... AND r.rn = s.sn )
```

wobei  $r_1 \dots r_n$  und  $s_1 \dots s_n$  die jeweils kompatiblen Attribute von  $r$  bzw.  $s$  sind.



**Kartesisches Produkt.** Das kartesische Produkt  $r \times s$  der beiden Relationen  $r$  und  $s$  enthält die Attribute von  $r$  und  $s$  (im Falle der Übereinstimmung doppelt und qualifiziert). Als Tupel in  $r \times s$  treten alle Kombinationen von Tupeln aus  $r$  und  $s$  auf. Dem entspricht die einfache SQL-Anweisung

```
SELECT * FROM r,s
```

SQL-92 definiert zusätzlich den CROSS JOIN und OUTER JOINS.

**Selektion.** Die Selektion findet in der SELECT-Anweisung im WHERE-Klauselteil ihre Entsprechung. Beispielsweise lautet die Übertragung der Selektion  $\sigma_{r.a \leq 100 \wedge r.name = fritz}(r)$  in SQL:

```
SELECT * FROM r
WHERE r.a <= 100 AND r.name = 'fritz'
```

bei geeignet festgelegter Relation  $r$ .

**Projektion.** Die Projektion  $\pi_{a,b,c}(r)$  der Tabelle  $r$  auf die Attribute  $a, b, c$  (einzelne Tabellenspalten) gelingt durch einfache Angabe der Spaltenbezeichner als Kommaliste nach dem Schlüsselwort SELECT:

```
SELECT r.a, r.b, r.c
FROM r
```

**$\theta$ -Verbund.** Beim  $\theta$ -Verbund  $r \bowtie_{a\theta b} s$  wird die Menge aller Tupel des kartesischen Produkts von  $r$  und  $s$  gebildet, bei denen  $r.a$  in der Relation  $\theta$  mit  $s.b$  steht.

```
SELECT *
FROM r,s
WHERE r.a  $\theta$  s.b
```

**Der natürliche Verbund**  $\bowtie$  ist im 89er Standard durch die SELECT-Anweisung nicht elementar ausdrückbar, kann aber erzeugt werden durch eine Aufeinanderfolge von Kartesischem Produkt, Selektion und Projektion. Zum Beispiel ist für die Relationen  $r$  und  $s$  mit den Attributen  $a, b, c, d$  und  $c, d, e, f$  der Verbund  $r \bowtie s$ :

```
SELECT r.a, r.b, r.c, r.d, s.e, s.f
FROM r,s
WHERE r.c = s.c AND r.d = s.d
```

Ab SQL-92 gibt es ein reichhaltiges Sortiment von JOIN-Operationen: CROSS JOIN (Kartesisches Produkt), NATURAL JOIN (Natürlicher Verbund), JOIN ... ON  $\theta$  (Theta-Join), LEFT OUTER JOIN, RIGHT OUTER JOIN (äußere Verbundoperationen). Sie folgen im wesentlichen der Syntax `SELECT * FROM r Jointyp s`.

```
SELECT *
FROM r NATURAL JOIN s
```

```
SELECT *
FROM r JOIN s ON r.a1 = s.a3
```

## Syntax der SELECT-Anweisung

Die nachfolgende Syntaxdefinition stellt nur eine Annäherung an die komplizierte Syntax des SELECT-Kommandos in der ältesten Standardform (SQL-89) dar.

*Selectkommando ::=*

```
SELECT [ ALL | DISTINCT ] { Selectliste | * }  
FROM Tabelle [ Alias ] { , Tabelle [ Alias ] }  
[ WHERE Suchbedingung ]  
[ GROUP BY Spalte { , Spalte } ]  
[ HAVING Suchbedingung ]  
[ ORDER BY Spaltenname [ ASC | DESC ] { Spaltenname [ ASC | DESC ] } ]  
[ INTO TEMP Tabellenname ]
```

*AllgSelectkommando ::=*

```
Selectkommando UNION [ALL] Selectkommando
```

In SQL-92 können statt UNION auch alle oben aufgeführten zweistelligen Operationen (JOIN, INTERSECT, ...) stehen. Die genaue Syntax der WHERE-Klausel und der HAVING-Klausel wird im Zusammenhang erweiterten SQL-Konzepten eingeführt ebenso wie Hinweise zur Benutzung der GROUP BY- und ORDER BY-Teile des SELECT-Kommandos.

## Weitere Beispiele und Befehle

DISTINCT

DISTINCT sorgt dafür, dass in der Ergebnistabelle keine doppelten Tupel enthalten sind. Wird diese Option weggelassen, ist als Standard ALL gesetzt, womit alle Tupel angezeigt werden, auch doppelte. DISTINCT ist nicht notwendig, wenn Mengenoperationen wie UNION verwendet werden.

```
SELECT DISTINCT a,b  
FROM r
```

ORDER BY

ORDER BY sortiert die Ergebnistabelle nach dem angegebenen Attribut. Dabei ist die aufsteigende Reihenfolge ASC als Standard gesetzt, für die absteigende Reihenfolge wird die Option DESC benötigt. Sortierungen können auch kombiniert werden.

```
SELECT *  
FROM r  
ORDER BY a DESC, b ASC
```

MIN / MAX

MIN() bzw. MAX() liefert den kleinsten bzw. größten Wert des angegebenen Attributs.

```
SELECT MAX(b)  
FROM r
```

## 5.2. Übungszettel A-11

**Aufgabe 11.1.** Gegeben ist eine Minimini-Firmenwelt: Artikel mit ANr (Artikelnummer) und Bezeichnung, Lieferanten mit LNr (Lieferantennummer) und Namen und Lieferungen in Mengen zu bestimmten (Einzel-)Preisen.

- Erzeugen Sie ein Schema mit SQL-Anweisungen in `create-firma.sql`. Achten Sie dabei auf die Einhaltung eventuell notwendiger Integritätsbedingungen.
- Erfinden Sie einen kleinen Datensatz mit INSERT-Anweisungen in `insert-firma.sql`, z.B.

ARTIKEL		LIEFERANTEN		LIEFERUNGEN			
ANr	AName	LNr	LName	ANr	LNr	Menge	Preis
1000	Bärchen	1	Meier	1000	1	100	0,99
1001	Schokolade	2	Schulze	1000	2	200	0,89
1002	Kekse	3	Müller	1001	2	50	1,89
1003	Küsse	4	Krause	1002	3	100	1,29
				1002	4	200	1,19
				1003	4	30	1,99

- Definieren Sie die Löschoperation für die DB in `drop-firma.sql`
- Erzeugen Sie die Tabellen mit Population in `psql`.
- Gesucht sind folgende Anfragen und ihre Antwortmengen
  - Namen aller Lieferanten
  - gelieferte Artikel
  - Lieferanten, die Kekse liefern
  - Nummer des Artikels, der zum niedrigsten Preis geliefert wurde
  - Alle Artikel sortiert nach Artikelnummer aufsteigend, danach nach Lieferpreis absteigend.

**Aufgabe 11.2.** Gegeben sei die Attributmenge  $A = \{A, B, C\}$  (alle Wertebereiche seien vom Typ `INTEGER` und die Relationen  $R(A, B)$  und  $S(B, C)$  mit den Anfangspopulationen  $r = \{(1, 2), (1, 2), (3, 4), (3, 4), (3, 4)\}$  und  $s = \{(2, 5), (2, 5), (3, 6), (3, 4), (3, 4)\}$ . Berechnen Sie die Ergebnisse der folgenden Anfragen. Kontrollieren Sie die Ergebnisse durch Ausführung im DBS.

- `SELECT * FROM R UNION SELECT * FROM S;`
- `SELECT * FROM R INTERSECT SELECT * FROM S;`
- `SELECT * FROM R EXCEPT SELECT * FROM S;`
- `SELECT * FROM R UNION ALL SELECT * FROM S;`
- `SELECT * FROM R INTERSECT ALL SELECT * FROM S;`
- `SELECT * FROM R EXCEPT ALL SELECT * FROM S;`

**Aufgabe 11.3.** Gegeben sei die Attributmenge  $\mathcal{A} = \{A, B, C\}$  (alle Wertebereiche seien vom Typ `INTEGER`) und die Relationen  $R(A, B)$  und  $S(B, C)$  mit den Anfangspopulationen  $r = \{(1, 2), (3, 4)\}$  und  $s = \{(4, 8), (5, 9)\}$ . Berechnen Sie die Ergebnisse der folgenden Anfragen. Kontrollieren Sie die Ergebnisse durch Ausführung in `psql`.

```

1 | SELECT * from R, S WHERE R.B = S.B;
2 | SELECT * from R          JOIN S ON R.B = S.B;
3 | SELECT * from R LEFT     JOIN S ON R.B = S.B;
4 | SELECT * from R RIGHT    JOIN S ON R.B = S.B;
5 | SELECT * from R FULL     JOIN S ON R.B = S.B;
6 | SELECT * from R NATURAL  JOIN S;
7 | SELECT * from R NATURAL LEFT JOIN S;
8 | SELECT * from R NATURAL RIGHT JOIN S;
9 | SELECT * from R NATURAL FULL JOIN S;
```

#### Aufgabe 11.4.

- (a) Schreiben Sie für die Uni-Miniwelt (siehe Anhang D.1) eine Folge von SQL-DDL-Deklarationen in die Textdatei `create-uni.sql`, die ein Relationenschema für die Datenbank Universitaet erzeugen.
  - (i) Trennen Sie die Anweisungen durch Semikola.
  - (ii) Deklarieren Sie die Primärschlüssel mit Hilfe von Tabellen-Constraints.
  - (iii) Deklarieren Sie Fremdschlüsselbezüge mit `FOREIGN KEY`.
  - (iv) Überlegen Sie genau, welche Attribute `NOT NULL` deklariert werden sollten.
- (b) Schreiben Sie eine Folge von `INSERT`-Anweisungen in die Textdatei `insert-uni.sql`, die die Tabellen mit der Probe-Datenbasis befüllt. Achten Sie dabei auf folgende Punkte:
  - Zeichenketten stehen in einfachen Hochkommata,
  - Jede `INSERT`-Anweisung terminiert mit einem Semikolon.
- (c) Schreiben Sie ein Folge von `DROP`-Anweisungen in die Textdatei `drop-uni.sql`, die sämtliche Tabellen der Universitäts-DB löscht.
- (d) Testen Sie die drei Skripte mit dem Kommandozeilen-Interpreter `psql` von `PostgreSQL` und überzeugen Sie sich mit einfachen `SELECT`-Befehlen von der Korrektheit der Tabelleninhalte.
- (e) Auf den Seiten 3–28ff. im Skript finden Sie relationenalgebraische Anfragen steigender Komplexität zur Universitäts-Miniwelt. Formulieren Sie entsprechende `SELECT`-Anweisungen, sofern sie Ihnen mit Ihrem bisherigen Handwerkszeug möglich erscheinen, und testen Sie diese in `psql`.

**Aufgabe 11.5.** *Im Material zur Vorlesung finden Sie das komplette Bibliotheks-Beispiel aus dem Anhang des Skripts in SQL-Form, das auch in der Vorlesung verwendet wurde. Geben Sie für dieses Relationenschema die folgenden Anfragen mit SQL **SELECT**-Ausdrücken an (schreiben Sie sie dazu zuerst in eine Datei **query-bibliothek.sql**) und testen Sie Ihre Ergebnisse in **psql**, indem Sie die Datei der **\i**-Direktive einlesen.*

- (a) An welchem Ort verlegt der Verlag Oldenbourg?*
- (b) Welche Anschriften haben die Teltower Leser?*
- (c) Welche Bücher (ISBN), die in der Bibliothek vorhanden sind, wurden im Springer-Verlag verlegt?*
- (d) Ausgabe der Liste der Bücher mit Titel, ISBN, Verlagsort.*
- (e) Welche Leser (Namen) haben ein Buch mit dem Titel ‘Taschenbuch der Algorithmen’ ausgeliehen?*
- (f) Welche Autoren (Name, Vorname) waren als Herausgeber tätig?*
- (g) Welche Leser (ENr, EName, EVorname) haben derzeit kein Buch ausgeliehen?*



### 5.3. Erweiterte SQL-Konzepte

Dieser Abschnitt behandelt SQL-Konzepte, die über das in der Schule in der Regel verwendete Niveau hinausgehen – zumindest im Grundkursbereich. Die Konzepte gehören zum Standard-Umfang von SQL-92 und sind daher in quasi jedem DBMS implementiert.

#### 5.3.1. WHERE-Ausdrücke

In der WHERE-Klausel können zum Vergleich von Attributwerten die Operatoren BETWEEN ...AND, IN und LIKE verwendet werden.

Ausdruck	Erläuterung
BETWEEN ...AND ...	sucht nach Werten innerhalb eines Bereiches
IN (...)	sucht nach Werten innerhalb einer gegebenen Menge
LIKE	vergleicht die Zeichenfolge einer Spalte mit einer festgelegten Zeichenfolge. Das Prozentzeichen % steht dabei für den Allquantor, der Unterstrich _ für ein beliebiges einzelnes Zeichen.

```
SELECT Titel, Seiten
FROM   Buecher
WHERE  Seiten BETWEEN 400 AND 600;
```

```
SELECT Titel, Jahr
FROM   Buecher
WHERE  Jahr IN (2008,2009,2011);
```

```
SELECT Titel, ISBN
FROM   Buecher
WHERE  Titel LIKE 'Dat%';
```

In vielen DBS existieren bessere Patternmatching-Instrumente als der Standard-LIKE-Operator. So bietet beispielsweise PostgreSQL die Funktion SIMILAR TO und verschiedene Formen von Regulären Ausdrücken an.

### 5.3.2. Sortieren

Mit Hilfe der ORDER BY-Klausel werden eine oder mehrere Spalten in aufsteigender (ASC) ORDER BY bzw. absteigender (DESC) Reihenfolge sortiert. Grobsyntax:

```
SELECT [ALL | DISTINCT] <Kommaliste von Spaltenbezeichnern>  
FROM <Kommaliste von Tabellenbezeichnern>  
ORDER BY <Spaltenbezeichner> [ASC|DESC]  
[, <Spaltenbezeichner> [ASC|DESC]..]
```

Im folgenden Beispiel werden alle Entleiher (Nrn) und Entleihdaten nach Datum geordnet:

```
SELECT ENr, Datum  
FROM Ausleihe  
ORDER BY Datum ASC;
```

### 5.3.3. Aliase und temporäre Tabellen

In einer Ergebnistabelle treffen die ursprünglichen Attributnamen der verwendeten Relationen mitunter nicht mehr das, was mit dem Ergebnis ausgesagt werden soll. Auch sind Attributnamen in den Relationen oft abgekürzt, wie z. B. *ename* für *Entleihername*. Abhilfe schaffen hier mittels *AS Aliase für Attributnamen*, die als Spaltentitel in der Ergebnistabelle angezeigt werden.

```
SELECT ename AS Entleihername  
FROM entleiher
```

Bei komplexeren Anfragen kann es vorkommen, dass eine Relation mehrfach verwendet wird, jedoch in unterschiedlichen Zuständen (z. B. bei Updates). Hierbei kann es entscheidend sein, die Tabellen mit unterschiedlichen Bezeichnern identifizieren zu können. Auch kann es bei langen Relationennamen die Lesbarkeit fördern, kurze Bezeichnungen einzuführen. Hierfür werden *Aliase für Tabellennamen* erzeugt, die in der Abfrage direkt hinter dem Tabellennamen definiert werden.

```
SELECT B.titel,E.regalnr  
FROM Buecher B NATURAL JOIN Buchexemplar E
```

Um Ergebnisse aus Abfragen zwischenspeichern, können sie in *temporären Tabellen* abgelegt werden. Temporäre Tabellen werden mit dem Schlüsselbegriff `TEMP TABLE` eingeführt und nach Beendigung der aktuellen Datenbanksitzung gelöscht. Mit `SELECT ... INTO` kann eine solche Tabelle direkt innerhalb einer Abfrage erzeugt und mit den Abfrageergebnissen gefüllt werden.

```
SELECT *  
INTO TEMP TABLE Temptabelle  
FROM Autor NATURAL JOIN Rolle NATURAL JOIN Buecher
```

INTO TEMP TABLE



#### 5.3.4. Sichten

Mit Hilfe von VIEWS (Sichten) kann die Sicht auf die Daten gegenüber Benutzern eingeschränkt werden. Anstatt einen vollständigen Zugriff auf die Relationen zu gestatten, werden nur die Informationen angezeigt, die in der Sicht definiert werden. Damit und mit geeigneter Zugriffsrechtevergabe ist ein fein abgestimmtes Information Hiding möglich. Mit Sichten können auch Daten zur Verfügung gestellt werden, die in dieser Form erst zusammengestellt oder berechnet werden müssen. Die Daten werden mittels einer SELECT-Anweisung aus den vorhandenen Relationen zusammengestellt.

```
CREATE VIEW Sichtname [ ( Kommaliste von Spaltenbezeichnern )]  
AS SELECT-Anweisung  
[ WITH CHECK OPTION ]
```

Folgende Anweisung erzeugt z. B. eine virtuelle Relation *NeuereBuecher* mit allen Büchern nach 2010:

```
CREATE VIEW NeuereBuecher AS  
SELECT *  
FROM Buecher  
WHERE Jahr > 2010  
WITH CHECK OPTION
```

Es ist möglich, über eine Sicht neue Tupel in die Relationen einzufügen, was aber zu Problemen führen kann. Mit der `WITH CHECK OPTION` werden in die Sicht und damit in die Basistabelle neu einzufügende oder zu aktualisierende Zeilen auf ihre semantische Übereinstimmung mit der Sichtdefinition überprüft. Ein einzufügendes Tupel wird nur akzeptiert, wenn es in der Sicht darstellbar ist. Ist keine Übereinstimmung gegeben, so wird das Einfügen oder Auffrischen abgelehnt. Dieser Mechanismus dient der semantischen Konsistenzhaltung der Datenbank. Zu beachten ist aber, dass ein Auffrischen von Sichten nur unter ganz bestimmten an die Definition der Sicht gestellten Voraussetzungen möglich ist. Dies hat prinzipielle logische Ursachen, da es zu Anomalien kommen kann.<sup>62</sup> Die genauen Voraussetzungen für ein Update von Sichten sind implementierungsabhängig (in PostgreSQL können Updates auf Sichten nur über `CREATE RULE` realisiert werden).

#### Beispiele

- (a) Erzeugen Sie eine Sicht, die alle Buchtitel mit ISBN, Titel, Verlagsnamen und Verlagssort *München* zeigt.

```
CREATE VIEW buchmitverlag(ISBN, Titel, VName, Ort) AS  
SELECT ISBN, Titel, VName, Ort  
FROM Buecher NATURAL JOIN Verlag  
WHERE Ort = 'München';  
WITH CHECK OPTION
```

<sup>62</sup>Siehe dazu z. B. [http://de.wikipedia.org/wiki/Sicht\\_\(Datenbank\)](http://de.wikipedia.org/wiki/Sicht_(Datenbank)) (17.12.2016).

Hier muss keine Qualifizierung erfolgen, da die Attributnamen eindeutig sind. Wird evtl. versucht, über diese Sicht ein neues Tupel mit dem Ort 'Berlin' einzufügen, ergibt dies eine Fehlermeldung.

- (b) Erzeugen Sie eine Sicht aus der Bibliotheksdatenbasis, die aus allen Entleihern (Name, Adresse) besteht, die mindestens ein Buch vor dem 1.9.2014 ausgeliehen haben.

```
CREATE VIEW saeumig (Name, Vorname, PLZ, Ort, Strasse, HNr) AS
SELECT DISTINCT EName, EVorname, PLZ, Ort, Strasse, HNr
FROM Entleiher NATURAL JOIN Adresse NATURAL JOIN Ausleihe
WHERE Datum < '2014-09-01';
```

Der Zusatz DISTINCT in der SELECT-Anweisung sorgt dafür, dass ein säumiger Leser nur einmal in der Sicht auftritt.

### 5.3.5. Gruppenfunktionen, GROUP BY

Nur auf wenigen Attributtypen lassen sich numerische Funktionen ausführen. Spalten mit z. B. den Typen INTEGER, SMALLINT, FLOAT, SMALLFLOAT, DECIMAL und MONEY gestatten Summen- und Mittelwertbildungen ebenso wie die Ermittlung des Maximal- bzw. Minimalwertes. Für Spalten anderen Datentyps kann man immerhin Zähloperationen durchführen, um etwa die Anzahl der Attributwerte eines Attributs zu ermitteln, die einer bestimmten Bedingung genügen. Die wichtigsten im Standard definierten Gruppenfunktionen bzw. *Aggregatfunktionen* sind:

Ausdruck	Erläuterung
COUNT	zählt die Einträge in einer Spalte
SUM	summiert die Einträge einer numerischen Spalte
AVG	mittelt die Einträge einer numerischen Spalte
MAX	Maximum einer numerischen Spalte (je nach DBMS auch auch mit Texttypen möglich)
MIN	Minimum einer numerischen Spalte (je nach DBMS auch auch mit Texttypen möglich)
COUNT(*)	liefert die Anzahl der Zeilen, wobei Zeilen mit Nullwerten mitgezählt werden

Die Gruppenfunktionen können anstelle eines Spaltenbezeichners in der SELECT-Klausel verwendet werden. Beispiel:

```
SELECT MIN (Preis), AVG (Preis), MAX (Preis)
FROM Buecher;
```

Mit dieser Anweisung wird eine neue Relation mit drei unbenannten Attributen erzeugt, die vom gleichen Typ wie der Attributtyp von Preis sind. Nicht möglich ist an dieser Stelle eine Abfrage wie z. B. SELECT Titel, MAX(Preis), da eine Auflistung von Tupeln mit dem Titel im Konflikt steht mit der Ausgabe eines einzigen Wertes beim maximalen Preis. In

manchen Fällen – insbesondere bei numerischen Auswertungen – sind zwei Konstruktionen nützlich, die die Syntax der SELECT-Anweisung anreichern: die GROUP BY-Klausel und mit ihr die HAVING-Klausel. GROUP BY

### Beispiel

Man möchte wissen, wann die Verlage zuletzt ein Buch veröffentlicht haben. Dazu teilt man die Bücher in Gruppen zum gleichen Verlag ein und ermittelt jeweils, wann das letzte Buch geschrieben wurde. Die Ergebnistabelle hält für jeden Verlag ein Tupel bereit mit dem Verlagsnamen und dem letzten Erscheinungsjahr.

```
SELECT Verlag, MAX (Jahr) AS LetztesJahr
FROM Buecher
GROUP BY Verlag;
```

Die Syntax ist im Groben:

```
SELECT [ ALL | DISTINCT ] <Kommaliste von Spaltenbezeichnern>
FROM <Kommaliste von Tabellenbezeichnern>
[ GROUP BY <Kommaliste von Spaltenbezeichnern> ]
```

Attribute, die bei festem Wert des Gruppierungsattributs variieren, dürfen in der SELECT-Liste nicht auftreten. Genau genommen wird von einer Anfrage mit Gruppierung nicht eine Tabelle erzeugt, sondern eine Menge von Tabellen – in unserem Beispiel je eine Tabelle pro Verlag. Das Ergebnis ist nicht notwendig sortiert. Man muss unbedingt diese Neuorganisation der Daten bei GROUP BY berücksichtigen, um die Struktur der Ausgabe darauf anpassen zu können. Grob gesprochen werden z. B. bei GROUP BY Verlag eine Reihe von 'Schubladen' geschaffen – für jeden Verlag eine – in die dann die Tupel mit dem jeweiligen Verlagsnamen hineingelegt werden. Danach ist es nicht mehr möglich, mehrere Werte eines Attributes aus einer 'Schublade' darzustellen, z. B. durch SELECT Verlag, Jahr, da dies bei mehreren Werten für Jahr in Bezug auf einen Verlag zu einer Darstellung als 'Tabelle in einer Tabelle' führen müsste.<sup>63</sup> Stattdessen ist für jede 'Schublade' bzw. jeden Verlagstupel nur noch jeweils ein Wert pro Attribut darstellbar, z. B. mit SELECT Verlag, MAX(Jahr).

### 5.3.6. HAVING-Klausel

Im Zusammenhang mit der GROUP BY-Klausel tritt häufig die HAVING-Klausel auf. Durch eine logische Bedingung nach dem Schlüsselwort HAVING werden bestimmte Gruppen selektiert. Mit anderen Worten: HAVING spielt die gleiche Rolle für GROUP BY wie WHERE für SELECT. HAVING

<sup>63</sup>Während in Postgresql ein solcher Versuch zu einer Fehlermeldung führt, muss dies bei anderen Datenbank-Managementsystemen nicht der Fall sein. In MySQL würde in diesem Beispiel die Abfrage SELECT Verlag, Jahr dazu führen, dass für jeden Verlag ein *beliebiges*, dem Verlag zugeordnetes Jahr ausgegeben wird.

### Beispiel

Gesucht ist die sprachliche Formulierung zu der SQL-Anfrage

```
SELECT  COUNT(*) AS Anzahl, ENr, MAX (Datum) AS Letztes
FROM    Ausleihe
GROUP BY ENr
HAVING  MAX (Datum) > '01.09.2014';
```

*Lösung:* Gesucht ist die Anzahl und das jüngste Ausleihdatum der Entleiher (ENr), die Bücher nach dem 1.9.2014 ausgeliehen haben, gruppiert nach Entleiher-Nummern.

Weitere Anfragen zu diesem Komplex:

- (a) Ermittle den Gesamtpreis und die mittlere Seitenzahl der von Entleiher 3 entliehenen Bücher.

```
SELECT SUM (Preis) AS Gesamtpreis, AVG (Seiten) AS Durchschnitt
FROM    Ausleihe NATURAL JOIN Buecher
WHERE   ENr = 3;
```

- (b) Ermittle für alle Entleiher das älteste Rückgabedatum der entliehenen Bücher.

```
SELECT  ENr, MIN(Datum)
FROM    Ausleihe
GROUP BY ENr;
```

### 5.3.7. Unteranfragen

Unteranfragen lassen sich über zwei Wege in eine Abfrage integrieren: in der FROM-Klausel und in der WHERE-Klausel. Liefert die Unteranfrage eine Ergebnistabelle, kann diese als Grundlage für eine übergeordnete Anfrage verwendet werden, sie muss lediglich mit AS mit einem beliebigen Namen versehen werden.

```
SELECT MIN(Datum)
FROM    (
    SELECT Datum
    FROM Ausleihe NATURAL JOIN Buecher
    WHERE vname='Oldenbourg')
AS xyz;
```

Normalerweise enthält die SELECT-Anweisung im WHERE-Klauselteil einfache logische Selektionsbedingungen, entspricht also in diesem Teil der Selektionsoperation der Relationenalgebra. Da in der WHERE-Bedingung ein logischer Ausdruck stehen muss, können an dieser Stelle nicht nur elementare Vergleiche, sondern auch Aussagen über Mengen zu boole'schen Werten ausgewertet werden. Damit können hier z. B. ein Enthaltenstest, ein Nichtenthaltenstest und all- und existenzquantifizierte Aussagen über Mengen zugelassen werden. Anders gefragt: Ist ein Wert in einer Liste von (typgleichen) Werten enthalten, nicht enthalten usw.? Und so existieren in SQL die Mengenoperatoren IN, NOT IN, ALL, EXISTS und ANY für die Verwendung von Unteranfragen in der WHERE-Klausel.

IN

```
SELECT LName
FROM   Leihbuecherei NATURAL JOIN Adresse
WHERE  Bezirk IN ( 'Charlottenburg', 'Mitte' );
```

liefert als Ergebnis

TU Zentralbibliothek  
Staatsbibliothek Unter den Linden  
Zentral- und Landesbibliothek Berliner Stadtbibliothek

Die Frage ist nur, wie im allgemeinen Fall die zugehörigen Mengen bzw. Listen für die Überprüfung gewonnen werden. Hierzu dienen die Unteranfragen. Sie führen zur Schachtelung von SELECT-Anfragen. Sollen beispielsweise alle Leihbüchereinamen für Entleihvorgänge gefunden werden, die Entleiher 3 betreffen, so ist es sinnvoll, erst die Menge der Entleihvorgänge von 3 zu ermitteln und dann die Leihbüchereinamen zu diesen Nummern herauszusuchen.

```
SELECT L.LName
FROM   Leihbuecherei L
WHERE  L.LNr IN (
    SELECT A.LNr
    FROM   Ausleihe A
    WHERE  A.ENr = 3 );
```

Man erhält nur dann sinnvolle Antwortmengen, wenn bei der Unteranfrage genau eine Spalte entsteht. Der Datentyp des Attributs dieser Spalte muss kompatibel sein zum Datentyp des in der Menge zu suchenden Elements. Es ist natürlich auch möglich, die gewünschte Anfrage über einen Natural Join von Leihbuecherei und Ausleihe mit entsprechender Bedingung zu lösen. Es kann aber aus Performanzgründen geeigneter sein, auf eine Anfrage über schlanke Listen zurückzugreifen, als auf rechenintensivere Kreuzprodukte.

Die Anfrage des Beispiels könnte auch mit dem Existenz-Operator formuliert werden:

```
SELECT L.LName
FROM   Leihbuecherei L
WHERE  EXISTS (
    SELECT *
    FROM   Ausleihe A
    WHERE  A.LNr = L.LNr AND A.ENr = 3 );
```

Der Ausdruck EXISTS <Unteranfrage> wird wahr, sobald die Unteranfrage mindestens ein Tupel liefert. Man beachte, dass die einschränkende Spaltenbedingung in der EXISTS-Variante wegfällt und deshalb im WHERE-Selektionsteil der Unteranfrage auftauchen muss. Die Bedingung A.LNr=L.LNr ist hier das verknüpfende Element zur übergeordneten Anfrage. Es werden alle Tupel der Relation Leihbücherei herausgefiltert, auf die diese Bedingung (und alle zusätzlich formulierten Bedingungen der Unteranfrage) zutrifft. Da die Auswahl von Tabellenspalten in der Unteranfrage keine Rolle spielt, könnte mittels SELECT 1 statt SELECT \* darauf verzichtet werden, was die Performanz erhöht.

## Weitere Beispiele

- (a) Finde die Erscheinungsjahre aller Bücher, die von Entleiher 3 ausgeliehen wurden.

```
SELECT B.Titel, B.Jahr
FROM   Buecher B
WHERE  B.ISBN IN (
        SELECT A.ISBN
        FROM   Ausleihe A
        WHERE  A.ENr = 3 );
```

- (b) Formuliere die Anfrage mit dem EXISTS-Operator.

```
SELECT B.Titel, B.Jahr
FROM   Buecher B
WHERE  EXISTS (
        SELECT *
        FROM   Ausleihe A
        WHERE  A.ENr = 3 AND A.ISBN = B.ISBN );
```

- (c) Ermittle die ISBN-Nummern für Bücher, die in *Dahlem* entliehen wurden.

```
SELECT DISTINCT A.ISBN
FROM   Ausleihe A
WHERE  A.LNr IN (
        SELECT L.LNr
        FROM   Leihbuecherei L NATURAL JOIN Adresse AD
        WHERE  AD.Bezirk= 'Dahlem' );
```

## 5.4. Integritätsbedingungen

Eine der großen Stärken von SQL ist die Möglichkeit, durch das Einfügen von Integritätsbedingungen – sogenannten **constraints** – die Daten auf das Einhalten semantischer Bedingungen zu überprüfen. Das können Bedingungen sein, die beim Einfügen oder Ändern von Daten erfüllt sein müssen, das können aber auch solche sein, die beim Löschen von Daten weitere Operationen nach sich ziehen (müssen). Die Bedingungen können sich auf einzelne Attribute beziehen, auf Bedingungen, die an verschiedene Attribute einer Relation geknüpft sind oder auch solche, die Bezug auf Tupel anderer Relationen Bezug nehmen.

Integritätsbedingungen werden in der Data Definition Language (DDL) verfasst und bei der Deklaration von Tabellen notiert. Im Folgenden sollen die verschiedenen Möglichkeiten kurz vorgestellt und an ein paar Beispielen illustriert werden.

Integritätsbedingungen lassen sich im Wesentlichen in zwei Arten klassifizieren: statische und dynamische. Statische Integritätsbedingungen beziehen sich auf einen Datenbankzustand, dynamische auf die Änderung von Datenbankzuständen. Dynamische werden ihrerseits noch in Übergangsbedingungen und sogenannte temporale Bedingungen unterschieden. Letztere sind Integritätsbedingungen, die sich auf eine Folge von Datenbankzuständen beziehen und eine zeitliche Einschränkung der Änderung beschreiben.

### (a) *Statische Bedingungen*

- beziehen sich auf **einen** Datenbankzustand
- Einschränkung der möglichen DB-Zustände durch Prädikate

### (b) *Dynamische Bedingungen*

- beziehen sich auf Änderungen, also auf **mehrere** Datenbankzustände
- Man unterscheidet:
  - *Übergangsbedingungen*  
Änderung von altem zu neuem DB-Zustand wird eingeschränkt
  - *temporale Bedingungen*  
Änderungen in einem bestimmten Zeitfenster werden eingeschränkt

### 5.4.1. Statische Integritätsbedingungen

Statische Integritätsbedingungen lassen sich ihrerseits wieder in unterschiedliche Gruppen einteilen. Solche, die einzelne Attribute betreffen, solche die unterschiedliche Attribute eines Tupels betreffen, solche, die Prädikate über eine Menge von Tupeln darstellen und zum Schluss solche, die Tupel über verschiedene Relationen hinweg miteinander in Beziehung setzen.

Im Folgenden werden die einzelnen Typen benannt und anhand kleiner Beispiele erläutert.

#### **Domain- bzw. Attribut-Bedingungen**

Diese Bedingungen sichern Beschränkungen des Wertebereiches einzelner Attribute zu wie z. B.  $\text{Preis} > 0$ .

```
CREATE TABLE Lieferungen (  
    ...  
    Preis NUMERIC(10,2) CHECK (Preis > 0)  
);
```

Wird versucht, ein Tupel einzufügen, das dieser Bedingung widerspricht, wird das Einfügen vom DBMS abgelehnt.

### Tupelbedingungen

Eine Tupelbedingung betrifft Attribute eines Tupels, stellt also ein Prädikat dar, das Attributwerte vergleicht. So könnte man z.B. verlangen, dass Telefon- und Faxnummer verschieden sind.<sup>64</sup>

```
CREATE TABLE Adressen (  
    ...  
    TelNr VARCHAR (20),  
    FaxNr VARCHAR (20) CHECK (TelNr != FaxNr),  
    ...  
);
```

### Relationen-Bedingungen

Relationen-Bedingungen sind Prädikate über die Menge *aller* Tupel einer Relation.<sup>65</sup> Spezielle Formen sind:

- *Schlüsselbedingungen* wie z. B. in der folgenden Form, in der die Bedingung an das Attribut angehängt wird

```
CREATE TABLE Artikel (  
    ANr INTEGER PRIMARY KEY,  
    ...  
);
```

oder also gesonderter Eintrag aufgeführt, z. B. für *zusammengesetzte* Schlüssel,

```
CREATE TABLE Artikel (  
    ANr          INTEGER,  
    ...,  
    PRIMARY KEY (ANr,...)  
);
```

oder in der alternativen Form mit benanntem Tabellenconstraint. Hier wird für die (normalerweise automatisch erzeugte) Indexdatei eine eigene Bezeichnung für den Schlüssel angelegt:

<sup>64</sup>Heutzutage ist das vielleicht vernünftig. In den 90er Jahren des vergangenen Jahrhunderts war analoge Telefonie noch weit verbreitet und Geräte, die zwischen Wandanschluss des Telefons und Telefon bzw. Fax geschaltet waren, haben bei einem eingehenden Anruf das Signal nach typischen Fax-Kommunikationslauten abgehört und je nach Vorhandensein solcher Töne die Verbindung an das Fax oder das Telefon weitergeleitet.

<sup>65</sup>Wir befinden uns bei dieser Fragestellung wieder im Themengebiet der funktionalen Abhängigkeit (siehe Kapitel 4).



```
CREATE TABLE Artikel (  
    ANr          INTEGER,  
    ...,  
    CONSTRAINT pkey_artikel PRIMARY KEY (ANr,...)  
);
```

- *Aggregatbedingungen* wie z. B.  $\text{SUM}(\text{Gehaelter}) < \text{Budget}$
- *Rekursive Bedingungen* wie z. B., dass in einer Relation Zugverbindungen jeder Ort von jedem anderen aus erreichbar ist.

### Referentielle Bedingungen

Referentielle Bedingungen stellen semantische Bezüge zwischen Paaren von Relationen sicher. Eine Form ist z. B. die *Fremdschlüsselbeziehung* als Teilmengenprädikat.

```
CREATE TABLE Lieferungen (  
    ANr INTEGER NOT NULL REFERENCES Artikel,  
    LNr INTEGER NOT NULL REFERENCES Lieferanten  
    ...  
);
```

In diesem Beispiel kann ein Tupel nur dann in die Relation *Lieferungen* eingefügt werden, wenn die Artikel- und die Lieferantenummer der Lieferung in der Tabelle *Artikel* bzw. *Lieferungen* bereits vorhanden ist.

Besteht eine solche Fremdschlüsselbeziehung zu einer anderen Relation, dann ist es nicht ohne weiteres möglich, jene Relation oder auch nur bestimmte Tupel darin zu löschen, wenn deren Werte als Fremdschlüssel verwendet werden. Eine einfache `DROP TABLE`-Anweisung für z. B. *Artikel* führt zu einer Fehlermeldung, da das die Konsistenz der Datenbank gefährdet. Abhilfe schafft hier die Option `DROP TABLE ... CASCADE`. Das Kaskadieren der Löschanweisung führt dazu, dass auch abhängige Objekte in anderen Relationen gelöscht werden (hier der Fremdschlüssel-Constraint).

Das Konzept der statischen Integritätsbedingungen ist sehr mächtig. Zusätzlich können in einigen DBMS für die Bedingungsprüfung Unterabfragen verwendet werden können, was aber aufgrund der sehr stark zunehmenden Komplexität der Abläufe Risiken für die Konsistenz der Daten birgt. Das folgende Beispiel funktioniert in dieser Form der `CHECK`-Bedingung aus Sicherheitsgründen nicht in PostgreSQL, macht aber deutlich, wie ein solcher Constraint aufgebaut ist. Es wird überprüft, ob Studenten die Vorlesung auch gehört haben, bevor man sie zu einer entsprechenden Prüfung zulässt. Ebenso werden Fremdschlüsselbeziehungen festgelegt; es wird festgelegt, was mit dem Wert bzw. Tupel passieren soll, falls der Original-Schlüsselwert gelöscht wird; es wird ein zusammengesetzter Schlüssel für diese Relation festgelegt.

```
CREATE TABLE pruefen (  
  MatrNr      INTEGER      REFERENCES Studenten ON DELETE CASCADE,  
  VorlNr      INTEGER      REFERENCES Vorlesungen,  
  PersNr      INTEGER      REFERENCES Professoren ON DELETE SET NULL,  
  Note        NUMERIC (2,1) CHECK (Note BETWEEN 1.0 AND 5.0),  
  CONSTRAINT pkey_pruefen PRIMARY KEY (MatrNr, VorlNr)  
  CONSTRAINT vorher hoeren CHECK (EXISTS (  
    SELECT *  
    FROM   hoeren h  
    WHERE  h.VorlNr = pruefen.VorlNr AND h.MatrNr = pruefen.MatrNr ))  
);
```

#### 5.4.2. Dynamische Integritätsbedingungen

Bisher gingen wir davon aus, dass das DBMS ein passives System ist in dem Sinne, dass Anfragen oder Transaktionen nur auf Veranlassung von außen, also durch den Benutzer, ausgeführt werden. Insbesondere hinsichtlich der Integritätskontrolle ist auch ein anderer Weg denkbar: Eine bestimmte Datenbank-Operation löst einen sogenannten **Trigger** aus, der als kleines Programm ausgeführt wird und Aktionen auf den Daten ausführt.

Sehr häufig werden die Trigger erst ausgelöst, nachdem eine Einfügung umgesetzt wurde, so dass die Datenbank unter Umständen dadurch in einen unzulässigen – inkonsistenten – Zustand gerät. In solchen Fällen muss die Operation, die zum Auslösen des Triggers geführt hat, wieder rückgängig gemacht werden. Dafür ist der sogenannte **Transaktions-Manager** zuständig, dem ein eigenes Kapitel gewidmet ist. Die Definition eines Triggers soll an einem Beispiel kurz dargestellt werden. Für intensivere Beschäftigung mit dieser Problematik siehe [Vos08] oder [KE09].

In unserer Uni-Miniwelt sollen alle Professoren, die (neuerdings) mehr als 5 SWS Vorlesungen halten, eine 5%-ige Gehaltserhöhung bekommen. Der Trigger, der dies leistet, ist schon recht kompliziert:

```
CREATE TRIGGER Viellehre  
  AFTER INSERT ON Vorlesungen  
  REFERENCING      NEW AS vo_neu  
  FOR EACH ROW  
    WHEN GROUP BY vo_neu.gelesenVon  
      HAVING SUM (Vorlesungen.SWS) > 5 AND  
              SUM (Vorlesungen.SWS) - vo_neu.SWS <= 5  
  UPDATE Professoren  
  SET    Professoren.Gehalt = Professoren.Gehalt * 1.05  
  WHERE  Professoren.PersNr = vo_neu.gelesenVon;
```

Die Semantik ist in etwa folgende: Nach einer Einfügung in Vorlesungen – das neue Tupel wird `vo_neu` genannt – wird die Relation Vorlesungen Reihe für Reihe untersucht nach der Erfüllung dieser Bedingung: Wenn die ganze Relation nach der Personalnummer des Professors im neuen Tupel (`vo_neu.gelesenVon`) gruppiert wird und *mit* dem neuen Tupel die Summe der dabei gruppierten SWS  $> 5$  und *ohne* die neuen SWS  $\leq 5$  ist, dann erneuere

die Relation Professoren und setze das Gehalt um 5% hoch für die Personalnummer, die im neuen Tupel steht.

Damit wollen wir das Kapitel über Integritätsbedingungen beenden. Wer mehr wissen möchte, kann sich durch stapelweise Bücher arbeiten. Für den Unterricht bleibt eines festzuhalten:

Es gibt bei weitem mehr als nur `SELECT ... FROM ... WHERE ...`.

## Literatur zu Kapitel 5

- [Eis05] EISENTRAUT, Peter: *PostgreSQL: ge-packt*. 1. Auflage. Bonn : mitp-Verlag, 2005 (Datenbanken). – ISBN 978-3-8266-1493-4
- [KE09] KEMPER, Alfons ; EICKLER, André: *Datenbanksysteme: Eine Einführung*. 7. Auflage. München : Oldenbourg Wissenschaftsverlag, 2009. – ISBN 978-3-486-59018-0
- [Vos08] VOSSEN, Gottfried: *Datenmodelle, Datenbanksprachen und Datenbankmanagementsysteme*. 5. Auflage. München : Oldenbourg, 2008. – ISBN 978-3-4862-7574-2

## 5.5. Übungszettel A-12

**Aufgabe 12.1.** *Im Anhang D finden Sie eine Übersicht zu den verwendeten Miniwelten, darunter auch die Universitäts-Miniwelt. Im Material zur Vorlesung finden Sie einen Ordner, in dem die Universitäts-Miniwelt in SQL-Dateien zur Verfügung steht. Erstellen Sie anhand des Anhangs D eine eigene Create-Datei, in der Sie Ihre eigene Auswahl an Datentypen und Integritätsbedingungen entwerfen. Vergleichen Sie Ihre Create-Datei mit der des Vorlesungsmaterials.*

*Achten Sie u. a. auf Folgendes:*

- (a) Ergänzen Sie Ihre Deklaration der Relationen um **ON DELETE**- und **ON UPDATE**-Anweisungen. Überlegen Sie sich genau, ob (und wo) Sie kaskadierende Löschoperationen erlauben und wo nicht.*
- (b) Ergänzen Sie die folgenden Integritätsbedingungen:*
  - (i) Studenten dürfen maximal 20 Semester studieren, der Default Wert für Semester sei 1.*
  - (ii) Die ID des Nachfolgers einer Vorlesung muss größer als die des zugehörigen Vorgängers sein.*
  - (iii) Die Matrikelnummer muss fünfstellig sein.*
- (c) Schöpfen Sie aus der Vielfalt statischer Integritätsbedingungen (siehe Vorlesung) und geben Sie Ihrem Uniwelt-Schema den letzten Schliff.*

## 5.6. Jenseits der Standard-Typen

### 5.6.1. Aufzähltypen

Mit der Deklaration

TYPE

```
CREATE TYPE Geschlecht AS ENUM ('m','w','d')
```

wird ein Aufzähltyp (ENUM-Typ) Geschlecht mit den Werten 'm', 'w' und 'd' für männlich/weiblich/divers erzeugt. Er kann dann wie ein eingebauter Standardtyp für Spaltendefinitionen in Relationen verwendet werden:

```
CREATE TABLE Angestellte (  
    PId      INTEGER,  
    Vorname  VARCHAR(20),  
    Nachname VARCHAR(30),  
    mwd      Geschlecht  
);
```

Daten werden wie gewohnt erzeugt:

```
INSERT INTO Angestellte VALUES  
    (1,'Hans','Maier','m'),  
    (2,'Lieschen','Müller','w'),  
    (3,'George','Schulze','d');
```

Sinn der Sache ist die Ablehnung jeglicher anderer Werte als 'm', 'w' oder 'd' durch das System:

```
INSERT INTO Angestellte VALUES(4,'Frieda','Schultze','f');
```

wird mit einer Fehlermeldung boykottiert. In SELECT-Ausdrücken nutzt man Aufzähltypen zum Beispiel so:

```
SELECT * FROM Angestellte WHERE mwd = 'w';
```

Auf Aufzähltypen liegt eine automatisch definierte Ordnung vor: die der Reihenfolge der Werte bei der Aufzählung in der Definition. Im Beispiel gilt also 'm' < 'w' < 'd', Vergleiche mit den üblichen Relationsoperatoren sind möglich. Auch die Standardfunktionen MIN und MAX sind anwendbar.

### 5.6.2. Zusammengesetzte Typen (Verbundtypen)

Mit der Typdeklaration

```
CREATE TYPE Namen AS (  
    Vorname VARCHAR(20),  
    Titel    VARCHAR(10),  
    Nachname VARCHAR(30)  
);
```

wird ein aus drei Komponenten zusammengesetzter Datentyp Namen erzeugt. Die Typen der Komponenten können natürlich verschieden sein. Wir erzeugen eine Relation Adressbuch mit den Attributen PID, Name, mw und Ort:

```
CREATE TABLE Adressbuch (  
    PID    INTEGER,  
    Name  Namen,  
    mw    Geschlecht,  
    Ort   VARCHAR(30)  
);
```

in die mit

```
INSERT INTO Adressbuch VALUES  
    (1,ROW('Hans',", 'Schulze'),'m','Berlin'),  
    (2,ROW('Lieschen','Prof. Dr.','Müller'),'w','Hannover');
```

Daten eingefügt werden. Komponenten werden in Anfragen mit der Punktnotation angesprochen, allerdings muss zur Unterscheidung von Tabellennamen der Verbundtyp-Name in runde Klammern eingeschlossen werden:

```
SELECT * FROM Adressbuch WHERE (Name).Vorname = 'Hans';
```

Interessant sind Verbunddatentypen als Rückgabewerte für serverseitige Funktionen.

### 5.6.3. Felder (ARRAY-Datentypen)

Als Beispiel kreieren wir eine Tabelle Kleidung mit den Attributen KNr, Typ und Farben.

```
CREATE TABLE Kleidung (  
    KNr    INTEGER,  
    Typ    VARCHAR(20),  
    Farben VARCHAR[]  
);
```

Die Deklaration der Farben als VARCHAR[], einem *Feld* von Strings, soll es uns ermöglichen, die Farben eines Kleidungsstücks zu notieren, auch wenn es nicht einfarbig ist. Soll die maximale Länge des Arrays offen bleiben, werden an den gewünschten Datentyp leere, eckige Klammern angefügt, ansonsten wird in die Klammern die maximale Länge eingefügt, z. B.: VARCHAR[10]. Man beachte aber: Der Feldindex beginnt nicht bei 0, sondern bei 1. Werte lassen sich auf zwei Weisen einfügen:

```
INSERT INTO Kleidung VALUES
  (1, 'Bluse', '{"blau","grün","gelb"}'),
  (2, 'Hose', '{"schwarz"}');
```

```
INSERT INTO Kleidung VALUES
  (3, 'Hemd', ARRAY['blau','gelb','schwarz']);
```

Ausschnitte des Farben-Feldes können zum Beispiel so erzeugt werden:

Farben[1]      erste Komponente  
Farben[1:3]    erste bis dritte Komponente

Operatoren sind unter anderem:

<expr> = ANY(<array-expr>)    wahr, wenn LS in <array-expr> enthalten ist  
                                  (z. B. 'grün' = ANY(Farben))

<expr> = ALL(<array-expr>)    wahr, wenn LS gleich allen Elementen von <array-expr>  
                                  (z. B. 'schwarz' = ALL(Farben))

@>                               ARRAY[1,3,4] @> ARRAY[3,1] „enthält“  
Farben @> '{"grün","gelb"}'

<@                                ARRAY[1,4] <@ ARRAY[3,1,7,4] „ist enthalten“  
                                  '{"grün","gelb"}' <@ Farben

Die Abfrage von Arrays ergibt einen Nullwert, wenn eine Indexposition nicht besetzt ist. Das kann zu Fehlern führen, z. B. bei einer Zählung.

SELECT Farben[2]	SELECT COUNT(*)	SELECT COUNT(Farben[2])
FROM Kleidung;	FROM (	FROM Kleidung;
	SELECT Farben[2]	
Farben	FROM Kleidung) AS xyz;	count
-----		-----
grün	count	2
	-----	
gelb	3	

Diese Ergebnistabelle im Beispiel ergäbe bei einer COUNT(\*)-Abfrage drei Farben anstatt zwei, da jede Zeile gezählt wird (Nullwert wird mitgezählt). Die Zählung der Werte eines konkreten Attributs dagegen ignoriert Nullwerte, COUNT(Farben[2]) liefert daher das Ergebnis zwei.

#### 5.6.4. Definition neuer Wertebereiche mit CREATE DOMAIN

Über CREATE DOMAIN-Deklarationen erreicht man, dass CHECK-Bedingungen nicht wiederholt werden müssen, wenn derselbe Wertebereich mehrmals in einem DB-Schema auftaucht. Die Syntax ist:

```
CREATE DOMAIN <name> [AS] <Datentyp> [DEFAULT <Ausdruck>] [<Bedingung>]
```

Beispiele:

```
CREATE DOMAIN PLZ      AS VARCHAR(5)
    CHECK(VALUE ~ '^d{5}$');
CREATE DOMAIN Adresse  AS VARCHAR(100)
    NOT NULL DEFAULT 'N.V.';
CREATE DOMAIN NatZahl  AS INTEGER
    CHECK(VALUE >= 0);
CREATE DOMAIN Wochentag AS VARCHAR(2)
    CHECK( VALUE IN ('Mo','Di','Mi','Do','Fr','Sa','So'));
```

Definierte Wertebereiche lassen sich bei der Erstellung von Typen verwenden, um hier diese Einschränkungen ebenfalls wirksam werden zu lassen.

```
CREATE TYPE Anschrift AS (
    Straße VARCHAR(20),
    PostLZ PLZ
);
```



## 5.7. Übungszettel A-13

**Aufgabe 13.1.** Definieren Sie einen Aufzähltyp *Wochentage* mit den Werten 'Mo' bis 'So'. Erzeugen Sie eine Tabelle *Kurse* mit den Attributen *KursId*, *Fach* und *Kurstag* unter Nutzung des eben definierten Aufzähltyps. Fügen Sie die Tupel (1, 'Mathematik', 'Mi'), (2, 'Mathematik', 'Do'), (19, 'Informatik', 'Di') und (22, 'Informatik', 'Mo') ein. Stellen Sie folgende Anfragen:

- (a) Welche Kurse finden nicht am Dienstag statt?
- (b) Alle Kurse zwischen Dienstag und Donnerstag.
- (c) Der Kurs, der als erster (letzter) in der Woche stattfindet.
- (d) Alle Kurse, geordnet nach Wochentagen.

**Aufgabe 13.2.** Erzeugen Sie einen Verbundtyp *Adresse*, der die Postleitzahl, den Straßennamen, die Hausnummer und den Ort enthält. Definieren Sie damit eine Variante *AdrBuch* des *Adressbuchs*, die die beiden Verbundtypen *Namen* und *Adresse* enthält. Erzeugen Sie eine dem *Adressbuch* entsprechende Tabelle mit drei geeignet konstruierten Tupeln. Gesucht sind folgende Anfragen:

- (a) Alle Adressen mit Titel im Namen.
- (b) Nachnamen aller Adressen mit Postleitzahlen in Sachsen (führende 0).

**Aufgabe 13.3.** Anfragen: Alle Kleidungsstücke

- (a) !
- (b) mit erster Komponente „blau“ .
- (c) mit den ersten beiden Farben.
- (d) die „schwarz“ enthalten.
- (e) die „blau“ und „gelb“ enthalten.

**Aufgabe 13.4.**

- (a) Testen Sie den Postleitzahlen-Wertebereich mit einer geeigneten Tabelle
- (b) Geben Sie als neuen Wertebereich an:
  - (i) Alle Gehälter zwischen 400 und 5000 Euro
  - (ii) Die Zahlungsart im Webshop (default Vorkasse, sonst Kreditkarte, Rechnung oder Nachnahme).

**Aufgabe 13.5.** Entwerfen Sie aus dem Schulkontext eine Relation *Kursliste* für Oberstufenkurse. Modellieren Sie dafür geeignete Attribute, Integritätsbedingungen und Datentypen. Testen Sie Ihre Ergebnisse durch entsprechende Abfragen.



## 6. Transaktionen, Serialisierbarkeit und Recovery

Ein DBMS muss für die Einhaltung vorgegebener Integritätsbedingungen sorgen. Des Weiteren spielt ein DBMS seine Stärken im Bereich des Mehrbenutzerbetriebes aus. Anfragen werden zu sogenannten Transaktionen gebündelt, die vom Transaktionsmanager verwaltet werden. Das Transaktionskonzept dient im Wesentlichen zwei Aufgaben:

- Synchronisation der Transaktionen (Scheduler)
- Recovery im Fehlerfall (Wiederherstellung)

Beiden Aspekten wird ein eigenes Kapitel gewidmet. Zuvor jedoch muss das Transaktionskonzept genauer untersucht werden. Mit diesen Fragestellungen begeben wir uns mehr und mehr in die Implementierungsebene eines DBMS – ein ziemlich komplexes Gebiet, das wir hier nur oberflächlich betrachten können. Jedoch finden sich viele Anknüpfungspunkte an bereits in vergangenen Semestern behandelte Themengebiete wie nebenläufige Programmierung oder verkettete Listen, so dass sich aus didaktischer Sicht auch ein kurzer Blick auf die Interna eines DBMS lohnt.

### 6.1. Transaktionen

Neben einem totalen Datenverlust ist der größte anzunehmende (Daten-)Unfall der Verlust der Datenintegrität. Soll beispielsweise ein Geldbetrag von Konto A auf Konto B überwiesen werden, dann muss der Geldbetrag bei Konto A abgebucht werden und anschließend Konto B gutgeschrieben werden. Wird – aus welchem Grund auch immer – zwar der erste aber nicht der zweite Schritt ausgeführt, verschwindet das überwiesene Geld quasi im Datennirwana – für den Kunden ein untragbarer Umstand. Ändert man die Reihenfolge – schreibt also zuerst gut und bucht dann ab, ergibt sich ein ähnlich unakzeptabler Zustand für die Bank.

Bei der Überlegung spielt es zunächst keine Rolle, aus welchem Grund die zweite Anweisung nicht ausgeführt wurde, ob Rechnerabsturz wegen Hauptspeicherverlust, Serverausfall wegen Plattenfehler oder kollidierende DB-Operationen anderer DB-Benutzer, in allen Fällen ergibt sich ein inkonsistenter Zustand der Datenbank. Schwierig ist das auch vor allem deswegen, weil ein solcher Fall oft nicht auffällt und man die Gelegenheit, den Datenbankzustand zu korrigieren, aus Unwissenheit verpasst. Um solche Zustände zu vermeiden, muss das DBMS in der Lage sein, die Fehler zu erkennen und ggf. selbsttätig darauf zu reagieren, ohne dass der DB-Anwender die Details zu sehen bekommt. Das DB-System muss die Datenbank immer von einem konsistenten in einen anderen konsistenten Zustand überführen. Sollte das nicht vollständig möglich sein, der Weg dorthin aber schon beschritten (dass z. B. das Geld bereits abgebucht wurde, aber nicht mehr gutgeschrieben), muss das System . den Weg zum ursprünglichen (konsistenten) Zustand zurückgehen können. Es gilt das *Alles-Oder-Nichts-Prinzip*

Alles-Oder-Nichts-  
Prinzip

**Definition 6.1.1** (Transaktion). *Eine Transaktion ist eine Folge von DB-Operationen, die entweder alle gemeinsam gültig werden oder keine von ihnen.*

*Als Sprachkonzept findet man die Klammerung einer Operationenfolge durch die Anweisungen <BOT> (Begin Of Transaction) und <EOT> (End Of Transaction).*

<BOT>

Abbuchung von 100 Euro von Kundenkonto A  
Gutschrift von 100 Euro auf Kundenkonto B  
Berechnung des Gesamtsaldos des Kunden

<EOT>

Transaktionen weisen die so genannten **ACID**-Eigenschaften auf:

**A: Atomicity (Atomizität).**

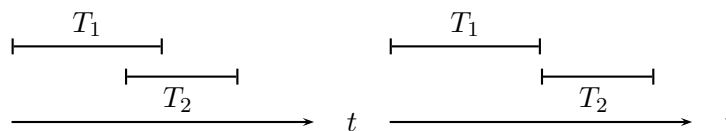
Eine Transaktion – bestehend aus einer beliebigen Anzahl von DB-Operationen – wird entweder ganz oder gar nicht ausgeführt (Alles-Oder-Nichts-Prinzip).

**C: Consistency (Konsistenz)**

Die Atomarität einer TA erzwingt, dass jede TA einen konsistenten DB-Zustand hinterlässt. Die Verantwortung dafür trägt allerdings das Applikationsprogramm bzw. der DB-Designer durch Hinzufügen und Einhalten von Integritätsbedingungen (Datentypen, CHECK, TRIGGER, ...).

**I: Isolation.**

Nebenläufig ausgeführte Transaktionen dürfen sich nicht beeinflussen. Der Transaktionsmanager sorgt dafür, dass zwei Transaktionen zeitlich entzerrt werden.



**D: Durability (Dauerhaftigkeit).**

- keine halben TAen auf der Platte
- die TA-Wirkung muss Abstürze, Plattenfehler, etc. überstehen

⇒ Hochverfügbarkeit: 99%, 99,5%, 99,9%, ...

**6.1.1. Einfaches Transaktions-Modell**

In einem einfachen Transaktionsmodell abstrahiert man zunächst einmal die komplexen Aktionen weg, die das DBMS ausführen muss, um die Daten auf der Platte oder einem anderen dauerhaft speichernden Medium so abzulegen, dass Schreib- und Lesezugriffe darauf effizient ausgeführt werden können.

Jede DB-Operation – und damit jede Transaktion – besteht aus einer Reihe von Schreib- und Leseoperationen auf dem DB-Speicher, bestehend aus dem dauerhaft speichernden Medium und einem DB-Puffer, der für einen höheren Durchsatz und eine größere Lebendigkeit des Systems sorgt.

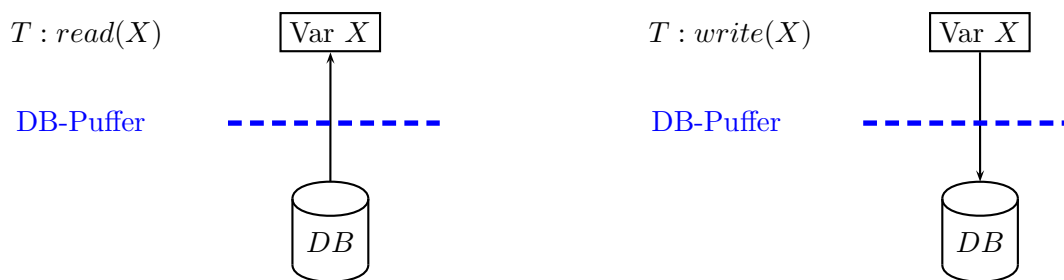


Abbildung 41: Ein einfaches TA-Modell, bestehend aus Lese- und Schreiboperationen einer Transaktion  $T$  auf einer Datenbank-Variablen  $X$ .

Das folgende Beispiel zeigt, welche ACID-Eigenschaft wie von einem Crash betroffen ist, der mitten in einer Transaktion auftritt. Es soll ein Betrag von 50 Euro von  $A$  nach  $B$  überwiesen werden. Nachdem der Betrag abgebucht wurde, tritt ein Systemfehler auf, der das Datenbanksystem betrifft.

```

T1:  read (A)
      DEC (A, 50)
      write (A)
      Crash
      read (B)
      INC (B, 50)
      write (B)
  
```

**Atomicity.** Abbruch an der Crash-Stelle führt zu inkonsistentem Zustand. Mit Hilfe von **Logs** kann der **Recovery**-Manager Konsistenz herstellen.<sup>66</sup>

**Consistency.** Den Wert  $A + B$  konstant zu halten, liegt in der Verantwortung der Anwendung.

**Isolation.** Nebenläufige  $A+B$ -Berechnung einer anderen Transaktion  $T_2$  führt unter Umständen zu einem so genannten **Dirty-Read**. Hier greift die **Nebenläufigkeitskontrolle**, die nur die zeitlich unverschränkten Abläufe  $T_1T_2$  oder  $T_2T_1$  zulässt.

**Durability.** Kein Systemfehler kann an der Überweisung etwas ändern.

⇒ Flush auf Platte vor EOT (**Recovery**-System)

### 6.1.2. Zustandsübergänge von Transaktionen

DB-Operationen werden logisch zu einer Transaktion zusammengefasst. Die konkurrierenden Transaktionen müssen, um die Integrität des Systems zu gewährleisten, vom Transaktionsmanager verwaltet werden. Dabei werden die einzelnen Transaktionen ähnlich den Prozessen in einem Von-Neumann-Rechner (vgl. NSP, Scheduler) in einer Art Prozessmodell von einem in einen anderen Zustand überführt (siehe Abbildung 42).

<sup>66</sup>Direkt nach dem Neuanlauf eines DBMS – vor Verarbeitung weiterer Anfragen – werden die Logs durchsucht, ob Transaktionen wegen eines Systemfehlers unterbrochen worden sind und die sich daraus ergebenden Inkonsistenzen beseitigt.

**potentiell.** Die Transaktion ist kodiert und wartet darauf, in den Zustand *aktiv* zu wechseln. Dieser Übergang heißt inkarnieren.

**aktiv.** Die aktiven (d. h. derzeit rechnenden) Transaktionen konkurrieren untereinander um die Betriebsmittel wie z. B. Hauptspeicher oder Rechnerkern zur Ausführung von Operationen.

**wartend.** Bei einer Überlast des Systems (z. B. trashing (Seitenflattern) des Puffers) kann die Transaktionsverwaltung einige aktive Transaktionen in den Zustand *warten* verdrängen. Nach Behebung der Überlast werden diese wartenden Transaktionen wieder eingebracht, d. h. wieder aktiviert.

**abgeschlossen.** Durch den COMMIT-Befehl wird eine aktive Transaktion beendet. Die Wirkung abgeschlossener Transaktionen kann aber nicht gleich in die Datenbank geschrieben werden, weil vorher eventuell verletzte Integritätsbedingungen geprüft werden müssen.

**persistent.** Die Wirkung abgeschlossener Transaktionen werden – wenn die Konsistenzerhaltung sichergestellt ist – durch festschreiben dauerhaft in die Datenbasis eingebracht. Damit ist die TA persistent. Dies ist einer von zwei möglichen Endzuständen einer Transaktionsverarbeitung.

**gescheitert.** TAen können aufgrund vielfältiger Ursachen scheitern. So kann der Benutzer selbst ein ABORT an das DBMS senden, es könne aber auch Systemfehler zum Scheitern aktiver oder wartender TAen führen. Selbst abgeschlossene TAen können – wenn sie Konsistenzbedingungen verletzen – in den Zustand *gescheitert* überführt werden.

**wiederholbar.** Einmal gescheiterte Transaktionen sind unter Umständen wiederholbar. Dazu muss deren Wirkung auf die Datenbasis zurückgesetzt werden. Danach können Sie durch neustarten wieder aktiviert werden.

**aufgegeben.** Eine Transaktion kann sich allerdings auch als hoffnungsloser Fall herausstellen. Dann muss ihre Wirkung zurückgerollt (ROLLBACK). Anschließend geht die TA in den Zustand *aufgegeben* über.

### 6.1.3. Transaktionsmanagement in SQL

In PostgreSQL beginnt standardmäßig jede Anweisung eine Transaktion, ohne dass ein explizites <BOT> nötig wäre (AUTOCOMMIT-Modus). Die Transaktion wird mit dem erfolgreichen Abarbeiten der Anweisung beendet. Gibt man einen BEGIN TRANSACTION-Befehl, werden nachfolgende Anweisungen erst durch einen der beiden folgenden Befehle abgeschlossen:

- COMMIT: Die in der TA vollzogenen Änderungen werden – sofern keine Probleme aufgetreten sind (z. B. Integritätsverletzungen) – festgeschrieben. Andernfalls wirkt COMMIT wie ein ROLLBACK.
- ROLLBACK: Alle Änderungen müssen zurückgesetzt werden. Anders als der COMMIT-Befehl ist die Ausführung eines ROLLBACK vom DBMS immer gesichert.

Das folgende Codebeispiel zeigt, wie eine Anweisungsfolge gekapselt werden kann, um die Datenintegrität zu gewährleisten – hier eine Umbuchung unter der zusätzlichen Bedingung, dass das Kundenkonto gedeckt ist.

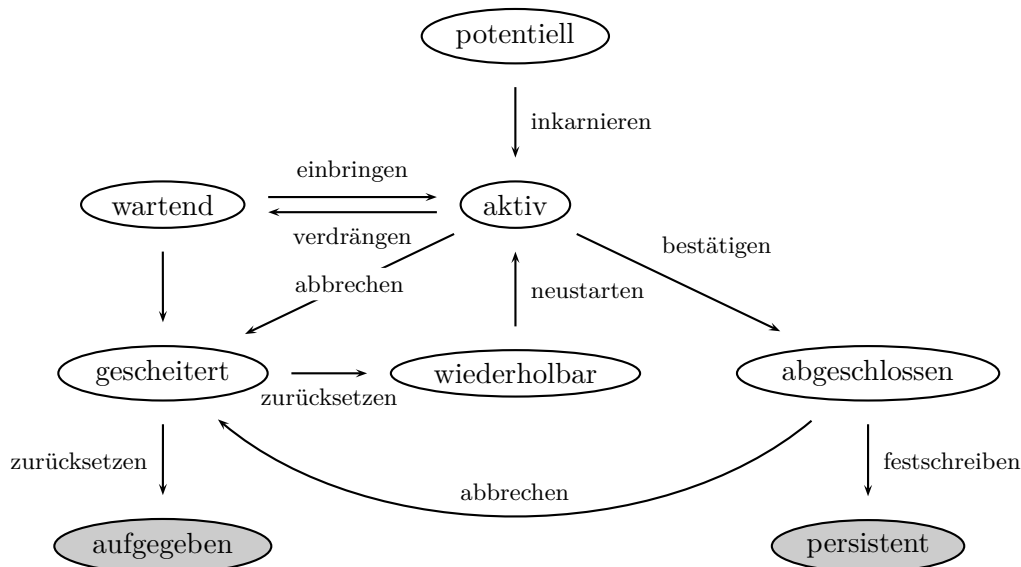


Abbildung 42: Zustandsübergangs-Diagramm für Transaktionen. Das Zustandsübergangsdiagramm ähnelt in Teilen dem Prozessmodell, das bereits aus der NSP bekannt ist.

```

1  -- Vor.: EXEC SQL CONNECT TO BANKDB
2  --      Enthaeelt Tabellen Girokonten, Kontoauszuege
3
4  PROCEDURE Abbuchung( :K: INTEGER; :Betrag: EURO);
5  BEGIN
6      EXEC SQL BEGIN DECLARE SECTION;           \
7          Kontostand, Dispo : EURO;              | Programmvariablen
8      EXEC SQL END DECLARE SECTION;             /
9
10     EXEC SQL
11         SELECT Saldo, Kredit INTO :Kontostand, :Dispo | Lesen
12         FROM   Girokonten
13         WHERE  KNr = :K;                             /
14
15     Kontostand := Kontostand - Betrag;             | Lokale Berechnung
16     IF Kontostand + Dispo >= 0
17     THEN
18         EXEC SQL
19             UPDATE Girokonten
20             SET   Saldo = :Kontostand
21             WHERE KNr = :K;                         /
22
23         EXEC SQL
24             INSERT INTO Kontoauszuege
25             VALUES (:KNr, :Betrag, :Kontostand);    /
26
27         EXEC SQL COMMIT WORK RELEASE;              | Commit
28     ELSE
29         WriteString("Konto überzogen");
30         EXEC SQL ROLLBACK;                         | Abort
31     END
32 END Abbuchung;

```

Die Transaktionsverwaltungs-Komponente sieht vom Ablauf dieser Prozedur nur wenig: Die

Programmvariablen sind lokal, die lokalen Berechnungen bleiben ihr verborgen. Das Lesen und Schreiben bedeutet Zugriff auf (Attribute) ein(es) Tupel(s) der Tabelle *Girokonten*, bzw. *Kontoauszuege*.

Es wird lesend zugegriffen auf die Objekte *Saldo* und *Dispo* und schreibend auf die Objekte *Saldo* und *Kontoauszuege*:  $r(\text{Saldo})$ ,  $r(\text{Dispo})$ ,  $w(\text{Saldo})$ ,  $w(\text{Kontoauszuege})$ .

Weiter abstrahiert sieht die Transaktionsverwaltung irgendwelche Datenbankobjekte  $a, b, c, \dots$  und Lese- und Schreibvorgänge hierauf,  $r(a)$ ,  $r(b)$ ,  $w(a)$ ,  $w(c)$  zum Beispiel, und die benutzergesteuert abgesetzten Operationen 'gültig machen' (COMMIT) und 'abbrechen' (ROLLBACK). Die Transaktionsverwaltung kann diese Operationen bei Bedarf auch in eigener Machtvollkommenheit in die Operationsfolge einstreuen.

#### 6.1.4. Lesende Transaktionen und Isolationsstufen

Nebenläufige Transaktionen können auch dann Probleme verursachen, wenn sie nur lesend auf die gleichen Daten zugreifen. Grund dafür ist, dass eine Transaktion die Datenbank zwar von einem konsistenten in einen anderen konsistenten Zustand überführt, zwischendurch aber inkonsistente Zustände existieren können. Erfolgt in einem solchen Zustand ein lesender Zugriff und werden die Informationen für weitere Auswertungen verwendet, können sich dadurch inkonsistente Zustände ergeben, die sogar außerhalb eines  $\langle \text{BOT} \rangle$ - $\langle \text{EOT} \rangle$ -Klammerpaares wirken.

**Definition 6.1.2** (Dirty Read). *Das nebenläufige Lesen von (nicht freigegebenen) Daten durch eine Transaktion  $T_1$ , auf die zeitgleich eine weitere Transaktion  $T_2$  innerhalb ihres  $\langle \text{BOT} \rangle$ - $\langle \text{EOT} \rangle$ -Blockes schreibend zugreift, nennt man **dirty read**. Die gelesenen Daten können einen nicht-konsistenten Zustand repräsentieren und führen u. U. außerhalb des Blockes zu Fehlern, die erst später ihre Wirkung zeigen.*

Ob ein *dirty read* hinnehmbar ist oder nicht, hängt vom konkreten Kontext ab, wie die folgenden beiden Beispiele zeigen.

#### Beispiel 1: Überweisung

Betrachte folgendes Programm, das die Überweisung von einem auf ein anderes Konto erledigen soll und dabei negative Kontostände verhindern soll.

Programm P1: Überweisung $A \rightarrow B$	
1.	Aufbuchen B
2.	Ist Konto A gedeckt?
2a.	NEIN: Abbuchen von Konto B
2b.	JA: Abbuchen von Konto A

Nun sollen zwei Überweisungen getätigt werden, wobei explizit negative Kontostände vermieden werden sollen (Überziehungszinsen!).



(a) Transaktion  $T_1$ :  $P1(K1, K2, 150)$

(b) Transaktion  $T_2$ :  $P2(K2, K3, 250)$

Es sollen also zum Einen 150€<sup>67</sup> von Konto 1 auf das Konto 2 gebucht werden und zum Anderen 250€ von Konto 2 auf das Konto 3. Die nachfolgende veschränkte Anweisungsfolge zeigt ein Beispiel, bei dem zwar kein Geld verloren geht, aber das Vorhaben, einen negativen Kontostand auf  $K2$  zu vermeiden, misslungen ist.

TA-Folge	K1	K2	K3	Bemerkung
BEGIN TA	100	200	300	
$T_2(1.)$			550	
$T_1(1.)$		350		
$T_2(2.)$				Ja, dirty!
$T_1(2.)$				Nein
$T_2(2b.)$		100		
$T_2(2b.)$	100	-50	550	$\Sigma$ 600

Will man dies auf jeden Fall vermeiden, muss der Transaktionsmanager ein Lesen unbestätigter Daten, wie hier an der Stelle  $T_2(2.)$ , verbieten. Dies muss der Anwender bei der Definition des <BOT>-<EOT>-Blockes festlegen.

## Beispiel 2: Flugplatzbuchung

Ein *dirty-read* muss nicht immer zu großen Problemen führen. Folgendes Programm soll in zwei Schritten zuerst den Status (*frei* oder *verfügbar*) eines Sitzplatzes eines Fluges prüfen und danach diesen bei Verfügbarkeit und eine Nachfrage buchen.

Programm P2: Sitzplatz-Buchung			
I Verfügbarkeit		II Bestätigung	
1.	Platz verfügbar?	2.	Platz buchen?
1a.	JA: Status := 'belegt'	2b.	JA: COMMIT
1b.	NEIN: END	2a.	NEIN: Status := 'frei'

Führen zwei Transaktionen  $T_1$  und  $T_2$  beide nebenläufig  $P2$  aus – beide anfänglich mit einem Platz 19A – kann es passieren, dass  $T_1$  Platz 19A belegt,  $T_2$  den Platz belegt sieht und einen neuen Platz wählt. Bestätigt nun  $T_1$  den Platz nicht, wird Platz 19A frei, und  $T_2$  konnte ihn trotz Verfügbarkeit nicht belegen. Diese Situation ist nicht gewollt, stellt aber im Vergleich zum vorigen Beispiel aber einen hinnehmbaren Zustand dar. Hier könnte der TA-Manager ein *dirty-read* erlauben.

<sup>67</sup>Eine Währung wurde natürlich nicht angegeben, sie spielt aber auch keine Rolle.

In SQL wird mit dem Befehl BEGIN eine Transaktion gestartet (entspricht <BOT>), die entweder mit COMMIT (erfolgreich) abgeschlossen oder mit ROLLBACK zurückgerollt werden kann.

```
BEGIN [ WORK | TRANSACTION ] [
    ISOLATION LEVEL {
        SERIALIZABLE | REPEATABLE READ |
        READ COMMITTED | READ UNCOMMITTED }
    READ WRITE | READ ONLY ]
```

Folgende Übersicht zeigt, wie strikt die einzelnen Isolationsstufen den Zugriff auf Daten vor dem Bestätigen durch ein COMMIT verbieten (nach [EN09, S. 440]).

Isolationsstufe	Dirty Read	Nonrepeatable Read	Phantom-Tupel
READ UNCOMMITTED	erlaubt	erlaubt	erlaubt
READ COMMITTED	nicht erlaubt	erlaubt	erlaubt
REPEATABLE READ	nicht erlaubt	nicht erlaubt	erlaubt
SERIALIZABLE	nicht erlaubt	nicht erlaubt	nicht erlaubt

READ COMMITTED erlaubt das Lesen von Daten erst, wenn diese durch ein COMMIT bestätigt wurden, bevor die Transaktion begann.

REPEATABLE READ sichert ab, dass bei einem wiederholten Lesen von Daten diese nicht in-between von einer anderen Transaktion verändert wurden.

SERIALIZABLE unterbindet Phantom-Tupel. Diese können entstehen, wenn mehrmals Tupel mittels einer WHERE-Klausel abgefragt werden. Fügt eine andere Transaktion zwischen durch zusätzliche Tupel ein, die diese Klausel erfüllen, ergibt eine wiederholte Abfrage vorher nicht-existente Tupel.

In PostgreSQL wird READ UNCOMMITTED ignoriert und hat die gleiche Auswirkung wie die Isolationsstufe READ COMMITTED.<sup>68</sup>

Für das erste Beispiel zur Überweisung ist daher die Setzung

```
SET TRANSACTION READ WRITE
```

sinnvoll. Im zweiten Beispiel, in dem die Buchung eines anderen Platzes hinnehmbar ist, genügt

```
SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED
```

<sup>68</sup>Zum gegenwärtigen Zeitpunkt (30.11.2020) verwendete Version 12.4, siehe <https://www.postgresql.org/docs/12/sql-set-transaction.html>.

## 6.2. Schedules

Obwohl einzelne Transaktionen für sich genommen die Integrität des Datenbank-Zustandes gewährleisten können, so muss dies bei (unvermeidbarer) nebenläufiger Ausführung mehrerer solcher TAen nicht der Fall sein, wie die Beispiel zu *lost update* oder *dirty read* bereits im letzten Kapitel gezeigt haben. Um die Korrektheit des Ablaufes mehrerer Transaktionen zu gewährleisten, setzt jedes DBMS einen **Datenbank-Scheduler** ein, der die Nebenläufigkeit steuert. Dessen Aufgabe ist es, die einzelnen Transaktionen zeitlich so anzuordnen, dass die Datenintegrität erhalten bleibt. Dazu wird mittels einer Serialisierbarkeitsanalyse festgestellt, inwieweit es möglich ist, gegebenenfalls verschränkte Transaktionen zeitlich zu entzerren, ohne die Wirkung der TAen zu beeinflussen. Falls dies möglich ist, verlaufen die TAen in jedem Fall wie (vom Anwender) gedacht, da die Atomarität der Transaktionen gesichert ist. Neben dem Scheduler sind noch andere Instanzen in einem DBMS am Werk, wie Abbildung

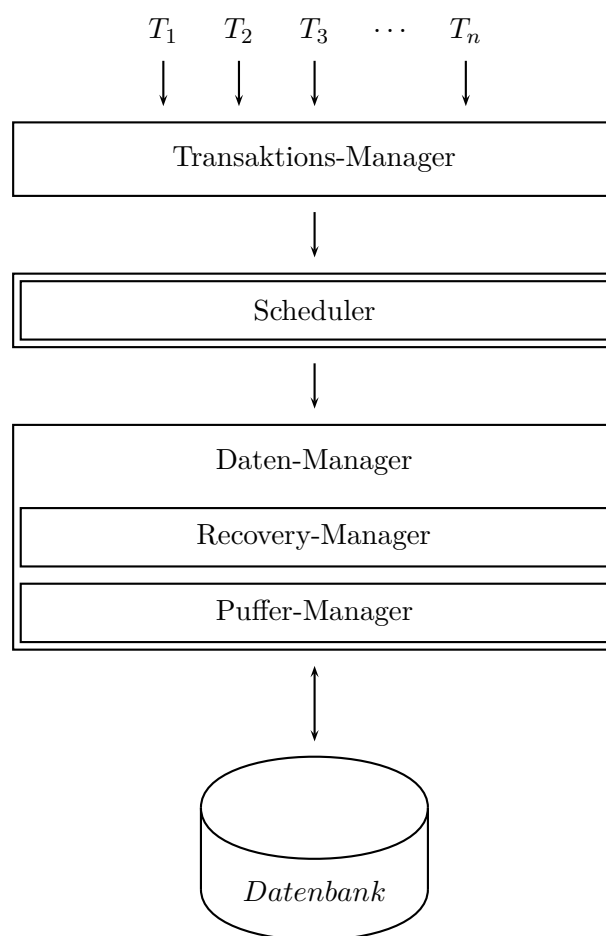


Abbildung 43: Die Stellung des *Schedulers* in der Datenbanksystem-Architektur (entnommen aus [KE09, S. 323]).

43 zeigt. Die Transaktionen werden vom Transaktionsmanager entgegengenommen, der sie verwaltet und an den Scheduler weiterreicht, der eine automatisierte Analyse der eingehenden Transaktionen vornimmt und feststellt, ob es sich um serialisierbare Ablaufpläne handelt und diese ggf. erzeugt. Der Scheduler stellt die Korrektheit des geplanten Ablaufes der Transaktionen mit Hilfe von **locks** (Sperrern) sicher, welche den Zugriff auf Datenbankelemente zeitweise

verhindern können. Die dabei häufig verwendete Methode zur Synchronisation nennt man **2-phase-locking** (zweiphasiges Sperren).<sup>69</sup> Der Scheduler verwaltet **Schedules** (Ablaufpläne). Ein Schedule ist eine zeitliche Abfolge (zentraler) Datenbankaktionen, die aus möglicherweise sehr vielen Transaktionen mit möglicherweise sehr vielen Einzeloperationen besteht. Die Aktionen finden auf dem **Datenbankpuffer** statt, geänderte Datenbankelemente werden asynchron durch die Pufferverwaltung persistent gemacht.

Folgendes Beispiel zeigt drei Transaktionen  $T_1$ ,  $T_2$  und  $T_3$ , die auf den Datenbankelementen  $A$  und  $B$  erbeiten.  $T_1$  will beide Werte um 100 erhöhen,  $T_2$  will beide verdoppeln und  $T_3$  erhöht beide Werte um 200.

$T_1$	$T_2$	$T_3$
Read (A, x)	Read (A, y)	Read (A, x)
Inc (x, 100)	$y := 2 * y$	Inc (x, 200)
Write (A, x)	Write (A, y)	Write (A, x)
Read (B, x)	Read (B, y)	Read (B, x)
Inc (x, 100)	$y := 2 * y$	Inc (x, 200)
Write (B, x)	Write (B, y)	Write (B, x)

Abbildung 44: Drei Transaktionen, aufgespalten in Elementaroperationen. Die Transaktionen  $T_1$ ,  $T_2$  und  $T_3$  werden in diesem Kapitel mehrfach verwendet.

Die Transaktionen können nun in verschiedener zeitlicher Verzahnung ablaufen, ihre Wirkung auf den Datenbankzustand ist (in der Regel) verschieden. Wichtige – und immer einzuhaltende – Randbedingung dabei ist jedoch, dass innerhalb eines Ablaufplanes die Einzeloperationen jeder Transaktion in der gleichen Abfolge auftreten, in der sie auch in ihrer Transaktion auftreten. Um z. B. den Wert von  $A$  zu verdoppeln, ist es notwendig, dass  $A$  zuerst gelesen wird, erst dann der Wert verdoppelt wird und erst danach der neue Wert wieder in  $A$  geschrieben wird. Vertauschungen innerhalb dieser Reihenfolge sind unzulässig, selbst wenn sie zufälligerweise das gleiche Ergebnis hätten.

Diese Mindestanforderung an einen Schedule ist die notwendige Bedingung dafür, dass die Semantik aller TAen sicher erhalten bleibt.

### 6.2.1. Serielle Schedules

Ein **serieller Schedule** besteht aus einer Aneinanderreihung von Transaktionen  $T_1, \dots, T_n$  in einer bestimmten Permutation, wobei die Einzelaktionen der Transaktion *nicht* mit denen anderer gemischt werden dürfen. Jede der  $n$  Transaktionen wird vom Scheduler als Block behandelt, so dass die Transaktionen in

$$N_{\text{seriell}} = n!$$

<sup>69</sup>Zur Methode des zweiphasigen Sperrens und dem Recovery später mehr.

Permutationen angeordnet werden können. Würde man die Forderung der Nichtvermischung fallen lassen, wäre eine viel größere Anzahl Permutationen möglich (siehe Kapitel 6.2.2).<sup>70</sup> Betrachtet man die möglichen seriellen Schedules, die die Transaktionen  $T_1$  und  $T_2$  umfasst, erhält man für das angegebene Beispiel (siehe Abbildung 44)  $N_{\text{seriell}} = 2! = 2$ . Abbildung 45 zeigt diese beiden möglichen seriellen Schedules  $S_1 = (T_1, T_2)$  und  $S_2 = (T_2, T_1)$  mit ihrer jeweiligen Wirkung auf den Datenbankelementen.

$S_1 = (T_1, T_2)$				$S_2 = (T_2, T_1)$			
$T_1$	$T_2$	A	B	$T_1$	$T_2$	A	B
<i>Startzustand</i>		25	25	<i>Startzustand</i>		25	25
Read (A, x)		125		Read (A, y)		50	
Inc (x, 100)				y := 2 * y			
Write (A, x)				Write (A, y)			
Read (B, x)				Read (B, y)			
Inc (x, 100)				y := 2 * y			
Write (B, x)			125	Write (B, y)			50
	Read (A, y)	250		Read (A, x)		150	
	y := 2 * y			Inc (x, 100)			
	Write (A, y)			Write (A, x)			
	Read (B, y)			Read (B, x)			
	y := 2 * y			Inc (x, 100)			
	Write (B, y)		250	Write (B, x)			150
<i>Endzustand</i>		250	250	<i>Endzustand</i>		150	150

Abbildung 45: Vergleich zweier serieller Schedules  $S_1 = (T_1, T_2)$  und  $S_2 = (T_2, T_1)$ . Die Endzustände der beiden Schedules sind zwar verschieden, der Endzustand der DB ist aber konsistent.

### 6.2.2. Serialisierbare Schedules

Ablaufpläne, in denen nur serielle Schedules auftreten, führen zwar immer zu einem konsistenten Datenbankzustand, können die Beweglichkeit eines Systems aber erheblich beeinträchtigen. So würde bei erzwungener serieller Ausführung eine Transaktion, die über mehrere Stunden läuft, das gesamte System lahmlegen.<sup>71</sup> Es gibt aber noch ein weiteres gewichtiges Argument, die parallele Ausführung von Transaktionen zu erlauben: Häufig können so Datenbankelemente aus dem Cache genutzt werden, die Ausführungsgeschwindigkeit steigt in diesem Fall um einige Größenordnungen. Um trotz verzahnter Ausführung von Transaktionen die Datenintegrität sicher zu stellen, erlaubt man nicht eine beliebige Verzahnung der Elementaroperationen der Transaktionen, sondern nur solche, die einem seriellen Schedule

<sup>70</sup>Einzig für den Fall, dass jede Transaktion aus genau einer Elementaroperation besteht, wären die Zahlen gleich. Die gleiche Problematik wurde bereits in der NSP behandelt, wo eine Transaktion einem Prozess entsprach.

<sup>71</sup>Hier zeigt sich wieder einmal, dass es größerer Beispiele bedarf, um die Notwendigkeit solcher theoretischen Überlegungen zu erkennen. Das zeigt sich auch in der Realität: Viele theoretische Überlegungen wurden erst angestellt, als konkrete produktive Anwendungen 'gegen den Baum gefahren' wurden.

in ihrer Wirkung gleichen. Es ließe sich also stattdessen (trotz seiner möglichen Nachteile) auch ein serieller Schedule verwenden. Einen solchen Schedule nennt man **serialisierbar**.

**Definition 6.2.1.** Ein Schedule  $S$  heißt **serialisierbar**, wenn es einen seriellen Schedule  $S'$  gibt, so dass für alle Datenbank-Anfangszustände die Wirkungen von  $S$  und  $S'$  gleich sind.

Ob ein gegebener Schedule serialisierbar ist oder nicht, ist nicht ohne Weiteres entscheidbar, weil dazu eine Allaussage zu prüfen wäre: "...für alle Datenbank-Anfangszustände ...". Im Kapitel über (Konflikt-)Serialisierbarkeit wird ein effizientes Verfahren vorgestellt, das geeignet ist, diese Frage im Allgemeinen zu beantworten.

Die Anzahl der möglichen Verzahnungen, die man erhält, wenn man keine seriellen Schedules fordert, ist erheblich größer als im seriellen Fall. Besteht die Transaktion  $T_i$  aus  $N_i$  Elementaroperationen, ergeben sich

$$N = \frac{(N_1 + N_2 + \dots + N_n)!}{N_1! \cdot N_2! \cdot \dots \cdot N_n!}$$

Permutationsmöglichkeiten. Betrachtet man wieder das Beispiel aus  $T_1$  und  $T_2$  ( $N_1 = N_2 = 4$ ), erhält man  $N = 70$  Möglichkeiten.

Die folgenden beiden Schedules  $S_3$  und  $S_4$  sind nicht seriell, die Einzeloperationen der beiden Transaktionen greifen verzahnt ineinander. Während  $S_3$  jedoch serialisierbar ist, gilt dies für  $S_4$  nicht.  $S_4$  lässt zudem die Datenbank in einem inkonsistenten Zustand zurück, da für den Anwender die Transaktionen als atomar zu betrachten sind und daher der gleiche Endwert zu erwarten wäre, wenn die Anfangswerte gleich sind.

$S_3$			
$T_1$	$T_2$	A	B
Startzustand		25	25
Read (A, x)		125	
Inc (x, 100)			
Write (A, x)			
	Read (A, y)		
	y := 2 * y		
	Write (A, y)	250	
Read (B, x)			
Inc (x, 100)			
Write (B, x)			125
	Read (B, y)		
	y := 2 * y		
	Write (B, y)		250
Endzustand		250	250

$S_4$			
$T_1$	$T_2$	A	B
Startzustand		25	25
Read (A, x)		125	
Inc (x, 100)			
Write (A, x)			
	Read (A, y)		
	y := 2 * y		
	Write (A, y)	250	
	Read (B, y)		
	y := 2 * y		
	Write (B, y)		50
Read (B, x)			
Inc (x, 100)			
Write (B, x)			150
Endzustand		250	150

In einigen speziellen Fällen kann es sein, dass eine ähnliche Historie wie die von  $S_4$  zu einem serialisierbaren Schedule führt. Ersetzt man in  $S_4$  die TA  $T_2$  durch  $T_3$ , ergibt sich aus arithmetischen Gründen ein serialisierbarer Schedule:

$S_5$			
$T_1$	$T_3$	$A$	$B$
<i>Startzustand</i>		25	25
Read (A, x)		125	
Inc (x, 100)			
Write (A, x)			
	Read (A, y)	325	
	Inc (x, 200)		
	Write (A, y)		
	Read (B, y)		
	Inx (y, 200)		
	Write (B, y)		225
Read (B, x)			
Inc (x, 100)			
Write (B, x)			325
<i>Endzustand</i>		325	325

Die Ursache für den Umstand, dass dieser Schedule serialisierbar ist, liegt in der Kommutativität der Addition begründet. Wegen  $x + 100 + 200 = x + 200 + 100$  folgt aus  $A = B$  auch  $A + 100 + 200 = B + 200 + 100$ . Eine solche Form der (zufälligen) Serialisierbarkeit soll im Folgenden nicht weiter beachtet werden; dieses Phänomen zeigt aber, dass es eines abstrakten Verfahrens bedarf, eine Transaktionsfolge auf Serialisierbarkeit zu untersuchen, denn der Scheduler kennt die Semantik der Operationen nicht und ist daher auch nicht in der Lage, auf der Basis semantischer Überlegungen einen serialisierbaren Schedule zu erstellen.

### 6.2.3. Das RW-Modell für Schedules

Um effizient Schedules – sowohl serielle als auch andere – zu notieren, wird im Folgenden noch weiter abstrahiert (vgl. 6.1.1)

- (a) Datenbankelemente werden mit großen Buchstaben ( $A, \dots, Z$ ) bezeichnet.
- (b) Leseoperationen werden mit  $r$  (read) bezeichnet.
- (c) Schreiboperationen werden mit  $w$  (write) bezeichnet.
- (d) Ein Index kennzeichnet die Zugehörigkeit der Elementaroperationen  $r$  und  $w$  zu den durchnummerierten Transaktionen.
- (e) Die Rechenoperationen im Hauptspeicher werden unterdrückt.

So bezeichnet  $r_3(A)$  den lesenden Zugriff auf das Datenbankelement  $A$ , ausgelöst von der Transaktion mit der Nummer 3.

Wegen der Unterdrückung der Hauptspeicheroperationen ist von der Transaktionssemantik nichts mehr zu sehen. Um die Transaktionen  $T_1$ ,  $T_2$  und  $T_3$  vollumfassend zu kennzeichnen, muss man die Transaktion erläutern (was natürlich nur bei unseren 'Spielbeispielen' effizient möglich ist).

- $T_1$  erhöht den Wert von  $A$  und  $B$  jeweils um 100.  
 $T_1 = (r_1(A), w_1(A), r_1(B), w_1(B))$

- $T_2$  verdoppelt den Wert von  $A$  und  $B$ .  
 $T_2 = (r_2(A), w_2(A), r_2(B), w_2(B))$
- $T_3$  erhöht den Wert von  $A$  und  $B$  jeweils um 200.  
 $T_3 = (r_3(A), w_3(A), r_3(B), w_3(B))$

Alle drei Transaktionen weisen die gleichen abstrakten Folgen von Elementaroperationen auf; ihre Semantik erschließt sich erst aus dem begleitenden Text.

Damit lassen sich beispielsweise die Schedules  $S_1$ ,  $S_3$  oder  $S_4$  folgendermaßen notieren:

- $S_1 = (r_1(A), w_1(A), r_1(B), w_1(B), r_2(A), w_2(A), r_2(B), w_2(B))$
- $S_3 = (r_1(A), w_1(A), r_2(A), w_2(A), r_1(B), w_1(B), r_2(B), w_2(B))$
- $S_4 = (r_1(A), w_1(A), r_2(A), w_2(A), r_2(B), w_2(B), r_1(B), w_1(B))$

Wie bereits erläutert, ist  $S_1$  ein serieller Schedule,  $S_3$  serialisierbar und  $S_4$  nicht serialisierbar.

### 6.3. Übungszettel A-14

#### Aufgabe 14.1. (*lost update*)

Gegeben sind zwei Transaktionen  $T_1$  und  $T_2$  mit den gleichen abstrakten Aktionsfolgen, aber unterschiedlicher Semantik:  $T_i = (w_i(A), w_i(B))$ .  $A$  und  $B$  sind dabei Gehaltskonten zweier Mitarbeiter,  $T_1$  setzt die Konten auf 1000 €,  $T_2$  auf 2000 €. Berechnen Sie die Historien der Kontostände für die folgenden Schedules:

- (a)  $S_1 = (w_1(A), w_1(B), w_2(A), w_2(B))$
- (b)  $S_2 = (w_2(A), w_2(B), w_1(A), w_1(B))$
- (c)  $S_3 = (w_1(A), w_2(A), w_2(B), w_1(B))$
- (d)  $S_4 = (w_1(A), w_2(A), w_1(B), w_2(B))$

Untersuchen Sie jeweils, ob die Integritätsbedingung, dass die Gehaltskonten der Mitarbeiter gleiche Konstostände aufweisen müssen, eingehalten wird. Welche Schedules sind seriell, welche serialisierbar, welche nicht serialisierbar?



**Aufgabe 14.2. (*dirty read*)**

Gegeben sind zwei Transaktionen  $T_1$  und  $T_2$  mit den gleichen abstrakten Aktionsfolgen, aber unterschiedlicher Semantik:  $T_i = (r_i(A), w_i(A), r_i(B), w_i(B))$ .  $A$  und  $B$  sind dabei verschiedene Girokonten, zwischen denen Überweisungen getätigt werden.  $T_1$  überweist 100€ von  $A$  nach  $B$ ,  $T_2$  bucht 5% Zinsen auf jedes Konto. Zu Beginn sei  $A = 200$  und  $B = 400$ . Berechnen Sie die Historien der Kontostände für die folgenden Schedules:

- (a)  $S_1 = (r_1(A), w_1(A), r_1(B), w_1(B), r_2(A), w_2(A), r_2(B), w_2(B))$
- (b)  $S_2 = (r_2(A), w_2(A), r_2(B), w_2(B), r_1(A), w_1(A), r_1(B), w_1(B))$
- (c)  $S_3 = (r_1(A), w_1(A), r_2(A), w_2(A), r_2(B), w_2(B), r_1(B), w_1(B))$
- (d)  $S_4 = (r_1(A), w_1(A), r_2(A), w_2(A), r_1(B), w_1(B), r_2(B), w_2(B))$

Analysieren Sie die Ergebnisse. Welche Schedules sind seriell, welche serialisierbar, welche nicht serialisierbar?

**Aufgabe 14.3. (*inconsistent read*)**

Gegeben sind zwei Transaktionen  $T_1$  und  $T_2$  mit den abstrakten Aktionsfolgen

$$T_1 = (r_1(A), r_1(B), r_1(C)) \quad \text{und} \quad T_2 = (r_2(C), w_2(C), r_2(A), w_2(A))$$

$A$ ,  $B$  und  $C$  seien drei Konten.  $T_1$  summiert in einer lokalen Variablen `sum` die drei Kontostände auf (und gibt die Summe aus).  $T_2$  überweist 100€ von Konto  $C$  auf Konto  $A$ . Die Anfangszustände der drei Konten seien  $A = 400$ ,  $B = 500$  und  $C = 300$ . Berechnen Sie die Historien der Kontostände für die folgenden Schedules:

- (a)  $S_1 = (r_1(A), r_1(B), r_1(C), r_2(C), w_2(C), r_2(A), w_2(A))$
- (b)  $S_2 = (r_2(C), w_2(C), r_2(A), w_2(A), r_1(A), r_1(B), r_1(C))$
- (c)  $S_3 = (r_1(A), r_1(B), r_2(C), w_2(C), r_2(A), w_2(A), r_1(C))$

Analysieren Sie die Ergebnisse. Welche Schedules sind seriell, welche serialisierbar, welche nicht serialisierbar?

**Aufgabe 14.4. (*unrepeatable read*)**

Gegeben sind zwei Transaktionen  $T_1$  und  $T_2$  mit den abstrakten Aktionsfolgen

$$T_1 = (r_1(A), r_1(A)) \quad \text{und} \quad T_2 = (r_2(A), w_2(A))$$

$A$  ist eine Kontentabelle,  $T_1$  liest zweimal und liefert einen statistischen Wert (`SUM` und `AVG`),  $T_2$  aktualisiert die Kontotabelle (z. B.  $A := A + 1000$ ).

- (a) Kommentieren Sie die zu erwartenden Ergebnisse für die beiden möglichen, seriellen Schedules  $S_1 = (T_1, T_2)$  und  $S_2 = (T_2, T_1)$ .
- (b) Konstruieren Sie einen nicht-seriellen Schedule, dessen Ausgabe das Phänomen des sogenannten **unrepeatable read** verdeutlicht.

## 6.4. Konfliktserialisierbarkeit

Im letzten Kapitel haben wir bereits die Frage nach der Serialisierbarkeit aufgeworfen. An ausgewählten Beispielen konnte die Frage beantwortet werden, indem ein serieller Schedule mit gleicher Wirkung angeführt wurde, der *offensichtlich* äquivalent war. Das gelang uns deswegen leicht, weil wir die jeweilige Semantik des Schedules kannten. Diesen Vorteil hat ein Datenbank-Scheduler nicht. Er muss allein anhand der Folge der Elementaroperationen  $r_i(X)$  bzw.  $w_i(X)$  entscheiden, ob ein gegebener Schedule serialisierbar ist.

Serielle Schedules – also solche, bei denen die beteiligten Transaktionen nicht verzahnt sind – werden wegen der sie charakterisierenden Eigenschaft der atomaren Ausführung als korrekt angesehen, serialisierbare Schedules aus diesem Grunde ebenfalls. Die Schwierigkeit besteht nun darin zu entscheiden, ob ein gegebener Schedule serialisierbar ist, ob es also einen äquivalenten seriellen Schedule gibt. Was heißt *äquivalent*? Die einfachste, aber leider wenig zufrieden stellende Definition der Äquivalenz ist die über die Wirkungsgleichheit auf der Datenbank. Dummerweise gibt es nämlich nicht-serialisierbare Schedules, die auf bestimmten Datenbankzuständen die gleiche Wirkung wie ein serieller Schedule haben. Das Beispiel auf Seite 6–12 zeigt bereits einen Fall, in dem die Serialisierbarkeit von der Semantik der Operationen abhängt. Ein Beispiel, in dem die Serialisierbarkeit von den Eingangswerten abhängt, zeigt Abbildung 46: Statt die Wirkungsgleichheit als Kriterium der

$S_1$			
$T_1$	$T_2$	A	B
<i>Startzustand</i>		3	3
Read (A, x)		9	
$x := x * x$			
Write (A, x)			
Read (B, x)			
$x := x * x$			
Write (B, x)			9
	Read (A, y)		
	$y := y - 5$		
	Write (A, y)	4	
	Read (B, y)		
	$y := y - 5$		
	Write (B, y)		4
<i>Endzustand</i>		4	4

$S_2$			
$T_1$	$T_2$	A	B
<i>Startzustand</i>		3	3
Read (A, x)		9	
$x := x * x$			
Write (A, x)			
	Read (A, y)		
	$y := y - 5$		
	Write (A, y)	4	
	Read (B, y)		
	$y := y - 5$		
	Write (B, y)		-2
Read (B, x)			
$x := x * x$			
Write (B, x)			4
<i>Endzustand</i>		4	4

Abbildung 46: Der rechts abgebildete Schedule  $S_2$  liefert nur durch Zufall das gleiche Ergebnis wie der links abgebildete serielle Schedule  $S_1$ , weil die Reihenfolge der Operationen *quadrieren* und *5 subtrahieren* vertauschbar ist, wenn der Eingangswert 3 ist. Das ist für keinen anderen Eingangswert so, weil  $x^2 - 5 = (x - 5)^2$  nur die Lösung  $x = 3$  besitzt.

Äquivalenz zweier Schedules zu verwenden, hat sich als wichtigstes Kriterium der Äquivalenz die sogenannte **Konfliktserialisierbarkeit** herausgestellt. Zudem lässt sich die Zugehörigkeit eines Schedules zur Klasse der konfliktserialisierbaren Schedules in polynomieller Zeit lösen,

was für die Effizienz eines Datenbankmanagementsystems von großer Bedeutung ist.<sup>72</sup>

### 6.4.1. Konflikte

$T_i$  und  $T_j$  seien zwei verschiedene Transaktionen ( $i \neq j$ ) eines Schedules  $S$ . In Konflikt stehende – kurz auch nur **konfligierende** – Operationen sind solche, bei der die Ausführungsreihenfolge Einfluss auf den Nachzustand des Systems hat.

Folgende Operationsfolgen stehen *nicht* in Konflikt, dürfen also vertauscht werden:

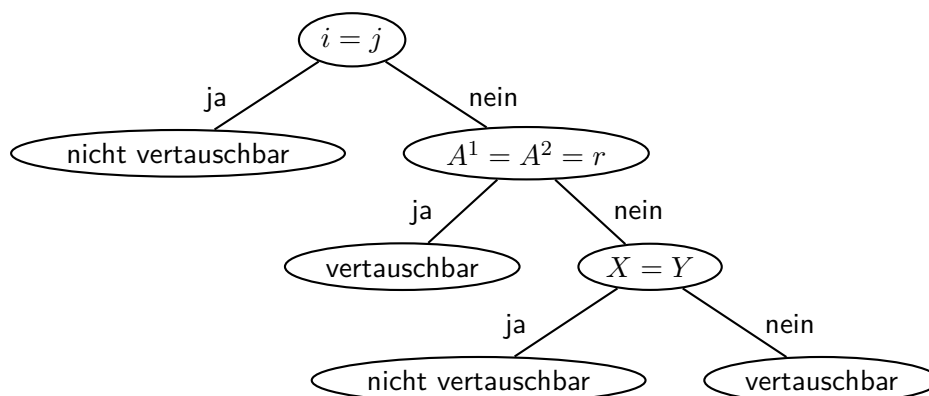
- (a) Leseoperationen  $r_i(X)$  und  $r_j(Y)$ , auch wenn  $X = Y$  ist.
- (b) Beliebige Operationen ( $r/w$ ), wenn sie auf unterschiedlichen DB-Objekten agieren:
  - $r_i(X)$  und nachfolgende Schreiboperation  $w_j(Y)$  für  $X \neq Y$
  - $w_i(X)$  und nachfolgende Leseoperation  $r_j(Y)$  für  $X \neq Y$
  - $w_i(X)$  und nachfolgende Schreiboperation  $w_j(Y)$  für  $X \neq Y$

Konfligierende und damit nicht tauschbare Operationenfolgen sind:

- (a) Aktionen einer Transaktion:  $r_i$  und  $w_i$ ,  $w_i$  und  $w_i$  sowie  $r_i$  und  $r_i$  dürfen nicht vertauscht werden, da ihre Reihenfolge wegen der Anwendungssemantik fixiert ist.
- (b) Operationen mit mindestens einem Schreiber auf gleichem DB-Objekt:
  - $w_i(X)$  und nachfolgendes  $w_j(X)$  für  $i \neq j$  (lost update)
  - $rw$ -Konflikt:  $r_i(X)$  mit nachfolgendem  $w_j(X)$  für  $i \neq j$
  - $wr$ -Konflikt:  $w_i(X)$  mit nachfolgendem  $r_j(X)$  für  $i \neq j$

Folgendes Baumdiagramm veranschaulicht, unter welchen Bedingungen benachbarte DB-Operationen  $A_i^1(X)$  und  $A_j^2(Y)$  vertauschbar bzw. nicht vertauschbar sind:

Entscheidungsbaum  
zur Vertauschung von  
Elementaroperationen



<sup>72</sup>Es gibt weitere Definitionen der Äquivalenz zweier Schedules. Eine davon ist die **View-Serialisierbarkeit**, die weniger restriktiv ist als die Konfliktserialisierbarkeit. Wesentliches Problem dabei ist, dass der Test auf View-Serialisierbarkeit NP-vollständig ist, eine effiziente Lösung also vermutlich nicht existiert.

**Satz 6.4.1** (Vertauschbarkeit von DB-Operationen).

- *Zwei Aktionen verschiedener Transaktionen können nicht vertauscht werden, wenn sie das gleiche Element betreffen und mindestens eine von ihnen schreibt.*
- *Zwei Aktionen der gleichen Transaktion können nie vertauscht werden.*
- *Alle anderen sind vertauschbar.*

**Definition 6.4.1** (Konfliktserialisierbarkeit).

- *Ein Schedule  $S_1$  heißt konfliktäquivalent zu  $S_2$ , wenn  $S_1$  mit Hilfe von Vertauschungen nicht konfligierender Elemente in  $S_2$  überführt werden kann.*
- *Ein Schedule  $S_1$  heißt konfliktserialisierbar wenn er konfliktäquivalent zu einem seriellen Schedule ist.*

**Beispiel 1**

Gegeben ist der Schedule zweier Transaktionen  $T_1$  und  $T_2$ :

$$S_1 = (r_1(A), w_1(A), r_2(A), w_2(A), r_1(B), w_1(B), r_2(B), w_2(B))$$

Durch Vertauschen benachbarter nicht konfligierender Operationen erhält man (die nächsten zu vertauschenden sind jeweils unterstrichen):

$$S_1 = (r_1(A), w_1(A), r_2(A), \underline{w_2(A)}, \underline{r_1(B)}, w_1(B), r_2(B), w_2(B))$$

$$S_1 = (r_1(A), w_1(A), r_2(A), r_1(B), \underline{w_2(A)}, w_1(B), r_2(B), w_2(B))$$

$$S_1 = (r_1(A), w_1(A), \underline{r_2(A)}, \underline{r_1(B)}, w_2(A), w_1(B), r_2(B), w_2(B))$$

$$S_1 = (r_1(A), w_1(A), r_1(B), r_2(A), w_2(A), w_1(B), r_2(B), w_2(B))$$

$$S_1 = (r_1(A), w_1(A), r_1(B), r_2(A), \underline{w_2(A)}, \underline{w_1(B)}, r_2(B), w_2(B))$$

$$S_1 = (r_1(A), w_1(A), r_1(B), r_2(A), w_1(B), \underline{w_2(A)}, r_2(B), w_2(B))$$

$$S_1 = (r_1(A), w_1(A), r_1(B), \underline{r_2(A)}, \underline{w_1(B)}, w_2(A), r_2(B), w_2(B))$$

$$S_1 = (r_1(A), w_1(A), r_1(B), w_1(B), r_2(A), w_2(A), r_2(B), w_2(B))$$

$S_1$  ist also konfliktserialisierbar.

**Beispiel 2**

Gegeben ist der Schedule  $S_2$  von Seite 6–16, der im R/W-Modell geschrieben werden kann als:

$$S_2 = (r_1(A), w_1(A), r_2(A), w_2(A), r_2(B), w_2(B), r_1(B), w_1(B))$$

Dieser Schedule ist nicht konfliktserialisierbar. Beim Versuch,  $T_1$  „nach vorne“ zu holen, konfligieren gleich zu Beginn zwei Operationen verschiedener Transaktionen  $T_1$  und  $T_2$  auf dem gleichen Datenbankelement  $B$ :

$$S_2 = (r_1(A), w_1(A), r_2(A), w_2(A), r_2(B), \underline{w_2(B)}, \underline{r_1(B)}, w_1(B))$$

Es genügt allerdings nicht zu prüfen, ob Transaktion  $T_1$  nach vorne zu holen ist. Es könnte ja auch  $T_2$  vorne stehen:

$$S_2 = (r_1(A), \underline{w_1(A)}, \underline{r_2(A)}, w_2(A), r_2(B), w_2(B), r_1(B), w_1(B))$$

Die Entflechtung der Elementaroperationen scheitert auch in diesem Fall, allerdings wegen Operationen verschiedener Transaktionen auf dem Datenbankelement  $A$ .

## Begriffsübersicht

- *kein Schedule*: Wenn im Scheduleentwurf nicht alle Einzeloperationen der enthaltenen Transaktionen aufgelistet werden oder diese Operationen nicht in der korrekten Reihenfolge erscheinen, in der sie in der Transaktion aufgeführt werden, dann liegt kein gültiger Schedule vor.
- *Seriell*: Einen Schedule nennt man seriell, wenn die enthaltenen Transaktionen mit allen jeweils zugehörigen Einzeloperationen nacheinander und nicht ineinander verzahnt abgearbeitet werden. Die Reihenfolge der Transaktionen spielt keine Rolle. Die Integrität der Datenbank ist nicht durch diesen Schedule bzw. dessen Struktur gefährdet.
- *Serialisierbar*: Einen Schedule nennt man serialisierbar, wenn die enthaltenen Transaktionen ineinander verzahnt abgearbeitet werden, die Ergebnisse des Schedules aber die gleichen sind, wie bei einem seriellen Schedule. Nachteile dieser Betrachtungsweise: Zum Einen können selbst problematische Schedules zu zufällig gleichen Ergebnissen führen. Zum Anderen erfordert der Vergleich der Ergebnisse einen Blick auf den Inhalt der Operationen bzw. der Datenbankelemente.
- *Äquivalent*: Die Äquivalenz zweier Schedules lässt sich auf zwei Ebenen untersuchen. Zum Einen können sie äquivalent sein, wenn sie zu gleichen Ergebnissen führen (s. serialisierbar). Hierfür muss die Semantik der Operationen bekannt sein. Zum Anderen können sie äquivalent sein, wenn sie durch Vertauschungen der Einzeloperationen einander angeglichen bzw. ineinander überführt werden können (s. konfliktäquivalent). Hierfür muss lediglich bekannt sein, ob die Operationen der Transaktionen lesend oder schreibend sind, ihr Inhalt spielt keine Rolle.
- *Konfliktäquivalent*: Einen Schedule nennt man konfliktäquivalent zu einem anderen Schedule, wenn man ihn mittels (erlaubter) Vertauschungen so lange ändern kann, bis er dem anderen Schedule gleicht. konfligierende Operationen sind in beiden Schedules in der gleichen Reihenfolge angeordnet. Existieren Konflikte innerhalb des einen Schedules, die die Integrität der Datenbank gefährden könnten, so existieren sie in gleicher Form auch im anderen Schedule. Ist ein Schedule konfliktserialisierbar, also konfliktfrei, dann ist es der andere auch.
- *Konfliktserialisierbar*: Einen verzahnten Schedule nennt man konfliktserialisierbar, wenn man ihn mittels (erlaubter) Vertauschungen so lange ändern kann, bis er einem seriellen Schedule gleicht. Das bedeutet, dass keine datenbankgefährdenden Konflikte innerhalb des Schedules vorliegen.

### 6.4.2. Präzedenzgraphen

Es ist aufwändig, die Konfliktserialisierbarkeit eines Schedules  $S$  wie in Beispiel 2 zu testen. Sind mehr als zwei Transaktionen beteiligt, muss prinzipiell für jede Permutation der  $n$  Transaktionen – und davon gibt es  $n!$  Stück – geprüft werden, ob sich  $S$  durch Vertauschen nicht konfligierender DB-Operationen in diesen seriellen Schedule überführen lässt. Zwar liefert das Verfahren im Gelingensfall auch gleich einen äquivalenten Schedule  $S'$ , jedoch ist es im konkreten Fall des DBMS gar nicht von Interesse, wie der serielle Schedule aussieht, sondern nur, ob es überhaupt einen gibt.

Woran scheitert die Serialisierung im Einzelfall? Immer dann, wenn zwei konfligierende DB-Operationen aufeinander treffen, die man nicht vertauschen kann, wird dadurch eine Ordnung der Menge der Transaktionen impliziert. Gibt es im Schedule beispielsweise die Operationenfolge  $w_1(A), w_2(A)$ , ist dadurch festgelegt, dass  $T_1$  vor  $T_2$  auch in einem eventuell existierenden konfliktäquivalenten seriellen Schedule  $T_1$  vor  $T_2$  kommt. Solche Operationsfolgen stellen Zwangsbedingungen für die Anordnung der Transaktionen in einem Schedule dar. Ein Schedule kann mehrere solcher impliziten Zwangsbedingungen enthalten, die sich u. U. auch widersprechen können. Weist ein Schedule neben der Teilfolge  $w_1(A), w_2(A)$  noch die Teilfolge  $w_2(B), w_1(B)$  auf, kann es keinen konfliktserialisierbaren seriellen Schedule geben, weil in ihm sowohl  $T_1$  vor  $T_2$  als auch  $T_2$  vor  $T_1$  auftreten müsste.

**Definition 6.4.2** (Präzedenzen). Sei  $\mathcal{T} = \{T_1, \dots, T_n\}$  eine Menge von Transaktionen und  $S$  ein Schedule über  $\mathcal{T}$ . Wir sagen  $T_i$  kommt vor  $T_j$ , geschrieben  $T_i <_S T_j$  ( $i \neq j$ ), falls es Aktionen  $A_i$  in  $T_i$  und  $A_j$  in  $T_j$  gibt ( $A \in \{r, w\}$ ), so dass gilt:

- $A_i$  steht in  $S$  vor  $A_j$ ,
- $A_i$  und  $A_j$  agieren auf demselben Datenbankelement,
- wenigstens eine Aktion schreibt.

$T_i <_S T_j$  nennt man eine **Präzedenz**. Sie bringt die Transaktionen  $T_i$  und  $T_j$  in eine Ordnung auf  $S$ .

Als Präzedenzgraphen bezeichnet man denjenigen gerichteten Graphen  $G_S = (V, E)$ , dessen Knotenmenge  $V$  aus den Transaktionen  $T_i$  von  $S$  und dessen Kantenmenge  $E$  aus gerichteten Kanten  $(T_i, T_j)$  besteht, wobei  $T_i <_S T_j$  eine Präzedenz des Schedule ist.

**Definition 6.4.3** (Präzedenzgraph). Der Präzedenzgraph  $G_S = (V, E)$  einer Schedule  $S$  zu einer Transaktionsmenge  $\mathcal{T} = \{T_1, \dots, T_n\}$  hat die Eckenmenge  $V = \{T_1, \dots, T_n\}$  und als Kanten die Menge der Paare  $E = \{(T_i, T_j) \mid T_i <_S T_j, i, j \in \{1, \dots, n\}\}$ .

#### Beispiel 1

Gegeben ist ein Schedule dreier Transaktionen  $T_1, T_2$  und  $T_3$ :

$$S = (r_2(A), r_1(B), w_2(A), r_3(A), w_1(B), w_3(A), r_2(B), w_2(B))$$

Es gilt:

- $T_2 <_S T_3$ , da  $r_2(A) <_S w_3(A)$ ,  $w_2(A) <_S r_3(A)$ ,  $w_2(A) <_S w_3(A)$
- $T_1 <_S T_2$ , da  $r_1(B) <_S w_2(B)$ ,  $w_1(B) <_S r_2(B)$ ,  $w_1(B) <_S w_2(B)$ .

Daher ist  $G_S = (\{T_1, T_2, T_3\}, \{(T_1, T_2), (T_2, T_3)\})$ .

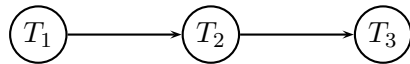


Abbildung 47: Präzedenzgraph zu  $G_S = (\{T_1, T_2, T_3\}, \{(T_1, T_2), (T_2, T_3)\})$ .

## Beispiel 2

Betrachten wir noch einmal den Schedule

$$S' = (r_1(A), w_1(A), r_2(A), w_2(A), r_2(B), w_2(B), r_1(B), w_1(B))$$

Für diesen gilt:

- $T_1 <_S T_2$ , da  $w_1(A) <_S r_2(A)$
- $T_2 <_S T_1$ , da  $r_2(B) <_S w_1(B)$

Der Schedule enthält also zwei sich widersprechende Präzedenzen. Im Präzedenzgraphen äußert sich das in einem Zyklus:



Abbildung 48: Präzedenzgraph zu  $G_S = (\{T_1, T_2\}, \{(T_1, T_2), (T_2, T_1)\})$ .

Anhand des Präzedenzgraphen kann nun leicht entschieden werden, ob ein Schedule konfliktserialisierbar ist oder nicht. Es gilt der Satz:<sup>73</sup>

**Satz 6.4.2.** *Ein Schedule  $S$  ist genau dann konfliktserialisierbar, wenn der zugehörige Präzedenzgraph  $G_S$  keinen Zyklus enthält.*

Der Sachverhalt ist unmittelbar klar: Ein Zyklus, in dem zwei Transaktionen  $T_i$  und  $T_j$  Knoten sind, impliziert zwei widersprechende Präzedenzen  $T_i <_S T_j$  und  $T_j <_S T_i$ .

<sup>73</sup>Den Beweis findet man z. B. in [Vos08, S. 655].

### 6.4.3. Topologische Sortierung

Der Präzedenzgraph liefert eine topologische Ordnung der Transaktionen. Eine topologische Ordnung ist eine strenge Halbordnung, die ähnliche Eigenschaften hat wie eine totale Ordnung, wobei allerdings im Gegensatz dazu nicht jede zwei Elemente zueinander in Relation stehen. Folgendes Beispiel soll dies veranschaulichen:<sup>74</sup>

Beim Anziehen von Kleidungsstücken ist eine bestimmte Reihenfolge einzuhalten, wobei nicht zwischen allen zwei Kleidungsstücken eine Reihenfolge zwingend vorgeschrieben ist. Hat man zum Beispiel eine Hose, ein Unterhemd, Hemd, Pullover, Mantel, Socken, eine Unterhose und ein Paar Schuhe, so kann man die folgenden Beziehungen für das Anziehen angeben.

- Unterhemd vor dem Hemd
- Hose vor dem Mantel
- Hemd vor dem Pullover
- Hose vor den Schuhen
- Unterhose vor der Hose
- Socken vor den Schuhen
- Pullover vor dem Mantel

Damit sind z. B. folgende Reihenfolgen möglich:

- (a) Unterhose, Socken, Hose, Unterhemd, Hemd, Pullover, Mantel, Schuhe
- (b) Unterhemd, Unterhose, Hemd, Pullover, Socken, Hose, Schuhe, Mantel

Es ist nicht möglich, mit dem Pullover anzufangen, weil vorher das Unterhemd und das Hemd angezogen werden muss.

**Satz 6.4.3** (Topologische Ordnung). *Sei  $M$  eine Menge,  $R$  eine Relation auf  $M \times M$ ,  $R \subseteq M \times M$ , und  $a, b, c \in M$ . Dann ist das Paar  $(M, R)$  eine topologische Ordnung (strenge Halbordnung), wenn gilt:*

- (a)  $\neg(aRa)$  (Irreflexivität) <sup>a</sup>
- (b)  $aRb \wedge bRc \Rightarrow aRc$  (Transitivität)

*Man beachte: Es wird für beliebige  $a, b \in M$  nicht gefordert, dass entweder  $aRb$  oder  $bRa$  gefordert wird.*

<sup>a</sup>Man kann die Hose nicht vor der Hose anziehen.

Mit dieser Definition kann nun leicht ermittelt werden, welcher serielle Schedule konfliktäquivalent zu einem gegebenen konfliktserialisierbaren Schedule  $S$  ist.<sup>75</sup>

**Satz 6.4.4** (konfliktäquivalente serielle Schedules). *Jede topologische Ordnung der Eckenmenge  $V(G_S)$  ist eine konfliktäquivalente serielle Anordnung der Transaktionen von  $S$ .*

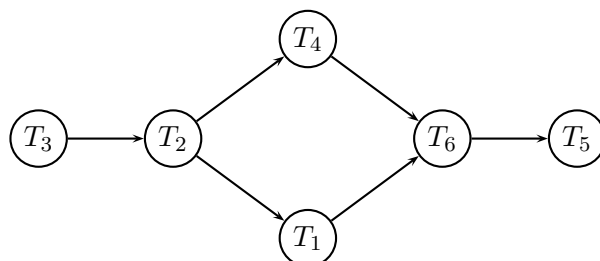
<sup>74</sup>Entnommen aus [http://de.wikipedia.org/wiki/Topologische\\_Sortierung](http://de.wikipedia.org/wiki/Topologische_Sortierung) (22.02.2015).

<sup>75</sup>Siehe auch [SKS11].



## Beispiel

Der Präzedenzgraph



hat die topologischen Ordnungen  $(T_3, T_2, T_4, T_1, T_6, T_5)$  und  $(T_3, T_2, T_1, T_4, T_6, T_5)$ .

Will man anhand eines gegebenen Schedules ermitteln, ob er konfliktserialisierbar ist und anschließend auch noch einen möglichen konfliktäquivalenten seriellen Schedule gewinnen, muss also zuerst ein Präzedenzgraph erstellt werden und dieser topologisch sortiert werden.

Den zweiten Schritt kann das Linux-Tool `tsort` übernehmen. Es führt eine topologische Sortierung durch und liegt unserer Distribution bereits bei, weil es früher benutzt wurde, um die richtige Einordnung von untereinander abhängigen Objektdateien in eine Programmbibliothek zu ermöglichen. Die folgenden beiden Beispiele zeigen die Wirkung von `tsort`.

## Beispiel mit existierender Totalordnung

In der Datei `Kleidung` ist die Halbordnung beschrieben.

1	Unterhemd	Pullover
2	Unterhose	Hose
3	Pullover	Mantel
4	Hose	Mantel
5	Hose	Schuhe
6	Socken	Schuhe

Das Tool `tsort` liefert eine mögliche Totalordnung.

```
$ tsort Kleidung
Socken
Unterhemd
Unterhose
Pullover
Hose
Schuhe
Mantel
```

## Beispiel mit Schleife

Die in der Datei `Schleife` beschriebene Halbordnung enthält einen Zyklus.

1	T1	T2
2	T2	T1

In diesem Fall existiert keine totale Ordnung. Entsprechend gibt `tsort` eine Fehlermeldung aus.

```
$ tsort Schleife
tsort: Schleife: Eingabe enthält eine Schleife:
tsort: T1
tsort: T2
```

#### 6.4.4. Topologisches Sortieren mit Miranda

Das Linux-Tool `tsort` soll in diesem Abschnitt mit Miranda nachgebaut werden. Wir gehen dazu davon aus, dass wir eine Liste von Präzedenzen  $T_i <_S T_j$  in der Form `[(T_i,T_j)]` bzw. allgemeiner `[*,*]` vorliegen haben. Daraus soll – sofern möglich – eine topologisch sortierte Liste `[*]` ermittelt werden. Der Algorithmus, der dies leistet, arbeitet folgendermaßen:

- 1. Schritt** Erstelle aus der Liste der Präzedenzen einen gerichteten Graphen, dargestellt durch das Paar `([*],[(*,*)])` aus Knoten- und Kantenliste.

```
tgraph :: [(*,*)] -> ([*],[(*,*)])
tgraph m = ((mkset . concat) [[ni,nj] | (ni,nj) <- m], m)
```

- 2. Schritt** Entferne aus dem Graphen einen Knoten, der keinen Vorgänger hat.
- 3. Schritt** Wiederhole Schritt 2 solange, bis der Graph keine Kanten mehr enthält.
- 4. Schritt** Entferne nun noch die verbleibenden Knoten aus dem Graphen.
- 5. Schritt** Die Liste der entfernten Knoten in der Reihenfolge ihrer Entfernung ist eine (mögliche) topologische Sortierung der Präzedenzliste.

Die Schritte 2-5 können in Miranda rekursiv implementiert werden. Dabei arbeitet die Funktion `tsort` mit Akkutechnik, damit im Fall eines detektierten Zyklus – der anzeigt, dass die Präzedenzliste keine topologische Sortierung erlaubt – eine leere Liste geliefert werden kann.

```
tsort :: ([*],[(*,*)]) -> [*]
tsort (ns,es)
  = tsort' (ns,es) []
  where
    tsort' (ns,es) akk
      = akk ++ ns,          if es = []
      = [],                if fs = []
      = tsort' (ns',es') (akk ++ [f]), otherwise
      where
        f  = hd fs
        fs = [n | n <- (mkset . map fst) es;
              (filter (==n) . map snd) es = []]
        ns' = filter (/=f) ns
        es' = filter p es
        where
          p (ni,nj) = (ni /= f) & (nj /= f)
```

Die topologische Sortierung erreicht man durch Hintereinanderausführung der Funktionen `tgraph` und `tsort`:

```
topsort :: [(*,*)] -> [*]
topsort = tsort . tgraph
```

'Füttert' man topsort mit der Liste

```
kleidung = [("Unterhemd", "Pullover"),  
            ("Unterhose", "Hose"),  
            ("Pullover", "Mantel"),  
            ("Hose", "Mantel"),  
            ("Hose", "Schuhe"),  
            ("Socken", "Schuhe")]
```

liefert der Aufruf topsort kleidung das Ergebnis

```
["Unterhemd", "Unterhose", "Pullover", "Hose", "Socken", "Mantel", "Schuhe"]
```

Das ist – wie man leicht nachprüft – eine mögliche Reihenfolge, die genannten Kleidungsstücke anzuziehen. Das Ergebnis unterscheidet sich allerdings von dem, das das Linux-Tool liefert. Es gibt eben mehrere Möglichkeiten, die Kleidung anzuziehen.

Wendet man topsort dagegen auf eine zyklenbehafteten Präzedenzliste an, erhält man keine topologisch sortierte Liste. Der Aufruf von topsort schleife mit

```
transaktionen ::= T1 | T2  
schleife = [(T1,T2),(T2,T1)]
```

liefert die leere Liste [].

## 6.5. Übungszettel A-15

### Aufgabe 15.1. Zu den Transaktionen

$$\begin{aligned}T_1 &= (r_1(Z), r_1(Y), w_1(Z)) \\T_2 &= (r_2(U), r_2(Z), r_2(Y), w_2(X), w_2(Y)) \\T_3 &= (r_3(T), w_3(Y)) \\T_4 &= (w_4(X), w_4(Y)) \\T_5 &= (r_5(V), w_5(U)) \\T_6 &= (r_6(X), w_6(U))\end{aligned}$$

seien folgende Ausführungsfolgen gegeben:

$$\begin{aligned}S_1 &= (r_1(Z), r_6(X), r_5(V), r_3(T), w_4(X), w_6(U), r_2(U), w_5(U), \\&\quad w_4(Y), w_3(Y), r_2(Z), r_2(Y), w_2(X), w_2(Y), r_1(Y), w_1(Z)) \\S_2 &= (w_4(X), w_4(Y), r_6(X), w_6(U), r_2(U), r_2(Y), r_2(Z), w_2(X), \\&\quad w_2(Y), r_3(T), w_3(Y), r_5(V), w_5(U), r_1(Z), r_1(Y), w_1(Z)) \\S_3 &= (r_2(U), r_1(Z), r_1(Y), r_5(V), w_1(Z), r_3(T), r_2(Z), w_4(X), \\&\quad w_5(U), r_6(X), r_2(Y), w_2(X), w_6(U), w_2(Y), w_4(Y), w_3(Y)) \\S_4 &= (r_2(U), r_1(Z), r_1(Y), r_5(V), w_1(Z), r_3(T), r_2(Z), w_4(X), \\&\quad w_5(U), r_2(Y), w_2(X), r_6(X), w_6(U), w_2(Y), w_4(Y), w_3(Y))\end{aligned}$$

Welche der Ablauffolgen sind (mit Begründung)

- (a) legale Schedules,
- (b) serielle Schedules,
- (c) konfliktserialisierbar (Nachweis über Vertauschungen und Präzedenzgraphen, Angabe der topologischen Ordnung eines äquivalenten seriellen Schedules),
- (d) nicht konfliktserialisierbar (Nachweis über Vertauschungen und Präzedenzgraphen)?

Prüfen Sie bei konfliktserialisierbaren Schedules Ihre topologische Ordnung mit dem Linux-Tool *tsort*.

**Aufgabe 15.2.** Untersuchen Sie, ob der Schedule  $S$  konfliktserialisierbar ist. Geben Sie hierfür die Präzedenzen und den Präzedenzgraphen an:

$$\begin{aligned}S &= (r_1(X), r_2(X), w_2(X), r_2(Y), r_2(U), r_3(X), \\&\quad r_4(U), w_3(V), w_4(U), r_5(V), r_5(Y), w_2(Y))\end{aligned}$$

**Aufgabe 15.3.** Zu dem folgenden konfliktserialisierbaren Schedule  $S$  sollen mindestens zwei verschiedene konfliktäquivalente serielle Schedules als topologische Ordnung angegeben werden. Prüfen Sie, welche Ordnung sich mittels  $tsort$  ergibt.

$$S = (r_1(X), r_2(X), r_2(Y), w_3(Z), w_3(T), r_1(Z), r_4(T), w_1(Z), \\ r_4(Y), w_4(T), w_4(U), r_5(Z), r_5(V), r_6(U), r_6(T), r_6(V), w_7(V))$$

**Aufgabe 15.4.** Gegeben sind die beiden folgenden Transaktionen:

T0: read(A);	T1: read(B);
read(B);	read(A);
if A=0 then INC(B);	if B=0 then INC(A);
write(B).	write(A).

Die Konsistenzbedingung sei  $A=0 \vee B=0$  mit den Anfangswerten  $A=B=0$ .

- (a) Zeigen Sie, dass jede serielle Ausführungsfolge mit diesen beiden Transaktionen die Konsistenz der Datenbank erhält.
- (b) Demonstrieren Sie eine nebenläufige Ausführung von T0 und T1, die einen nicht-serialisierbaren Ausführungsplan erzeugt.
- (c) Gibt es eine nebenläufige Ausführung von T0 und T1, die einen serialisierbaren Ausführungsplan erzeugt? Wenn ja, geben Sie eine an, wenn nein, begründen Sie dies.

## 6.6. Übungszettel A-16 – Querschnittsübung

### Aufgabe 16.1.

- (a) Erläutern Sie die Semantik der drei Armstrong-Axiome  
 $Y \subseteq X \Rightarrow X \rightarrow Y$ ,  
 $X \rightarrow Y \wedge Y \rightarrow Z \Rightarrow X \rightarrow Z$ .  
 $X \rightarrow Y \Rightarrow XZ \rightarrow YZ$ ,
- (b) Leiten Sie unter Angabe der verwendeten Axiome die Gültigkeit der Schlussregeln  
(i)  $X \rightarrow Y \wedge WY \rightarrow Z \Rightarrow XW \rightarrow Z$  (Pseudotransitivität)  
(ii)  $X \rightarrow YZ \Rightarrow X \rightarrow Y \wedge X \rightarrow Z$  (Zerlegung)  
ab.
- (c) Gegeben seien die funktionalen Abhängigkeiten  
(1)  $A \rightarrow B$ , (2)  $C \rightarrow B$ , (3)  $D \rightarrow ABC$ , (4)  $AC \rightarrow D$ . Leiten Sie systematisch unter Angabe der verwendeten Gesetze die Gültigkeit der funktionalen Abhängigkeiten  
(i)  $D \rightarrow ABCD$ , (ii)  $AC \rightarrow BD$ , (iii)  $AC \rightarrow ABCD$  her.

**Aufgabe 16.2.** Gegeben sind die zwei Mengen funktionaler Abhängigkeiten  $\mathcal{F} = \{AC \rightarrow B, A \rightarrow C, D \rightarrow A\}$  und  $\mathcal{G} = \{A \rightarrow BC, D \rightarrow AB\}$ . Zeigen Sie mit Hilfe der Armstrong-Axiome, dass  $\mathcal{F} \equiv \mathcal{G}$  ist.

**Aufgabe 16.3.** Für die Relation  $R$  mit  $\mathcal{A}(R) = ABCDEFG$  gelten die funktionalen Abhängigkeiten  $\mathcal{F} = \{B \rightarrow E, C \rightarrow F, BD \rightarrow G\}$ .

- (a) Berechnen Sie  $ABC^+$ .
- (b) Ist  $ABC \rightarrow D$  ableitbar (d.h. aus  $\mathcal{F}^+$ )?  
(schlüssige Erklärung oder Beispielpopulation)

**Aufgabe 16.4.** Untersuchen Sie für die Menge funktionaler Abhängigkeiten  $\mathcal{F} = \{AB \rightarrow C, D \rightarrow E, AE \rightarrow G, GD \rightarrow H, ID \rightarrow J\}$  für die Relation  $R$  mit  $\mathcal{A}(R) = ABCDEFGHIJ$ , ob

- (a)  $ABD \rightarrow GH \in \mathcal{F}^+$   
(b)  $ABD \rightarrow HJ \in \mathcal{F}^+$

gilt.

**Aufgabe 16.5.** Gegeben sind zwei Relationen  $R$  und  $S$  mit Beispielausprägungen:

$R$				$S$		
$A$	$B$	$C$	$D$	$A$	$B$	$E$
0	a	z	1	1	c	k
0	a	t	4	1	a	n
1	a	u	1	0	a	k
0	b	t	1	0	a	n

Anfragen an diese Datenbasis können als Relationenalgebra-, Tupelkalkül-, Domain-Kalkül-Ausdruck bzw. als Datalog-Programm formuliert werden. Zu

(a)  $\pi_{AC}(R \bowtie S)$

(b)  $\{ \langle c, e \rangle \mid \exists a \exists b \exists d \exists f \exists g (R(a, b, c, d) \wedge S(f, g, e) \wedge a < f \wedge b \neq g) \}$

sollen das Anfrageergebnis und die jeweils 3 nicht angegebenen äquivalenten Formulierungen ermittelt werden.

**Aufgabe 16.6.** Gegeben sind die Relationen  $R$ ,  $S$  und  $T$  mit den Relationsschemata  $R(A,B)$ ,  $S(A,C,D)$  und  $T(C,E)$  und folgenden Extensionen:

$R$		$S$			$T$	
$A$	$B$	$A$	$C$	$D$	$C$	$E$
2	c	3	x	1	v	4
2	b	7	z	1	u	5
3	c	2	u	8	z	4
5	d	3	z	9	w	6
7	b				x	5

(a) Geben Sie die Antwortmengen für folgende Anfragen im Tupelkalkül über  $R$ ,  $S$  und  $T$  an:

- (i)  $\{s \mid S(s) \wedge (s[A] = 3 \vee s[D] = 1)\}$
- (ii)  $\{t \mid \exists r, s (R(r) \wedge S(s) \wedge t[A] = r[A] \wedge t[D] = s[D] \wedge r[A] = s[A])\}$
- (iii)  $\{t \mid T(t) \wedge \exists s (S(s) \wedge t[C] = s[C])\}$
- (iv)  $\{t \mid \exists r, s (R(r) \wedge S(s) \wedge t[B] = r[B] \wedge t[D] = s[D] \wedge r[A] \neq s[A] \wedge r[B] = 'c')\}$

(b) Schreiben Sie die Anfragen als Ausdrücke der relationalen Algebra.

**Aufgabe 16.7.** *Ein IT-Konzern bietet hausinterne Fortbildung an. Die Fortbildungskurse decken Themengebiete ab, die für die Entwicklung des Konzerns wichtig sind. Die Fortbildungsorganisation wird datenbankgestützt organisiert. Die Angestellten können Fortbildungsveranstaltungen bestimmter Dauer zu einem festgelegten Termin besuchen. Für die Angestellten sind neben dem Namen die Personalnummer, die Funktion bzw. der Aufgabenbereich und das Gehalt repräsentiert. Die Fortbildungen sind bestimmten Gebieten eindeutig zugeordnet. Unter den Angestellten gibt es welche, die mit der Leitung eines (maximal eines) Fortbildungsgebiets betraut sind. Sie betreuen zu ihm eine Webseite im Intranet des Konzerns. Es wird erwartet, dass die Fortbildungen erfolgreich verlaufen. Mehrfacher Besuch der gleichen Fortbildung ist nicht erlaubt.*

- (a) *Entwerfen Sie zu dieser Beschreibung ein ER-Modell mit Angabe der Komplexitäten. Formulieren Sie ggf. zusätzliche Annahmen durch begleitenden Text.*
- (b) *Geben Sie eine einschränkende Bedingung an, die im Modell nicht repräsentiert werden kann.*
- (c) *Transformieren Sie das Modell in einen normalisierten relationalen Tabellentwurf mit Kennzeichnung der Schlüssel. Wo liegen Fremdschlüsselbeziehungen vor?*

**Aufgabe 16.8.** *Formulieren Sie die folgenden Anfragen in SQL.*

- (a) *Welche Fortbildungen werden im Gebiet ‘Datenbanken’ angeboten?*
- (b) *Welche Angestellten haben einen Fortbildungskurs aus dem Gebiet ‘Datenbanken’ belegt?*
- (c) *Welche Angestellten haben überhaupt keine Fortbildungskurse belegt?*
- (d) *Die Anzahl der Angestellten, die eine ‘Datenbank’-Fortbildung belegt haben.*
- (e) *Das minimale und maximale Gehalt der Angestellten, aufgeschlüsselt nach Aufgabenbereich.*

**Aufgabe 16.9.** *Bearbeiten Sie von den letzten beiden Übungszetteln die Aufgaben 14.1-14.3 und 15.1-15.3. Auch diese dienen als Vorbereitung für die DB-Klausur.*



## 6.7. 2-Phasen-Sperrprotokoll (2PL)

Die Untersuchung eines Schedules auf Serialisierbarkeit anhand des Präzedenzgraphen liefert zwar die Information, ob ein Schedule serialisierbar ist, kann aber im Fall einer Nichtserialisierbarkeit nicht mehr eingreifen; schlimmstenfalls muss eine Transaktion zurückgewiesen werden, weil ihre Ausführung die Datenintegrität der Datenbank gefährden würde.

### 6.7.1. Sperrbasierter Scheduler

Um im laufenden Betrieb das Zurückrollen eines Ablaufplanes wegen nicht gegebener Serialität eines Schedules zu vermeiden, wurden mehrere Strategien zu ihrer Vermeidung entwickelt. Eine ist die **sperrbasierte Ablaufplanung**, bei der gemeinsamer Zugriff auf gleiche Datenbankvariablen von unterschiedlichen Transaktionen durch das Setzen von Sperren auf Datenbankvariablen verhindert wird. Der Scheduler verwaltet der Einfachheit halber zuerst nur eine Sorte von Sperren; das Bild wird später verfeinert.

#### Erweiterung des RW-Modells

Das RW-Modell, in dem bisher nur die Operationen  $r_i(X)$  und  $w_i(X)$  zugelassen waren, wird um Operationen zum Setzen von Sperren und Entfernen von Sperren auf einzelnen Datenbankobjekten erweitert.

RWL-Modell statt  
RW-Modell

$l_i(X)$ .  $T_i$  fordert eine Sperre (lock) auf der DB-Variablen  $X$  an.

$u_i(X)$ .  $T_i$  gibt eine Sperre (unlock) auf der DB-Variablen  $X$  frei.

Dieses Modell nennt man das RWL-Modell (*read-write-lock-Modell*). Dabei muss ein sperrbasierter Scheduler mindestens zwei Anforderungen erfüllen:

- (a) *Transaktionskonsistenz*: Aktionen und Sperren müssen zusammenspielen:
  - (i) Eine Transaktion  $T$  liest oder schreibt ein Datenbankelement  $X \iff T$  hat vorher eine Sperre auf  $X$  erhalten und hat diese Sperre noch nicht zurückgegeben.
  - (ii) Erhält eine Transaktion  $T$  eine Sperre auf  $X$ , so *muss* sie diese später zurückgeben.

Das bedeutet: Führt  $T_i$  eine der Aktionen  $r_i(X)$  oder  $w_i(X)$  aus, so gab es vorher eine Sperranforderung  $l_i(x)$  ohne zwischenzeitliche Rückgabe  $u_i(X)$ . Außerdem folgt auf diese Aktion ein Unlock  $u_i(X)$ .

- (b) *Schedule-Korrektheit*: Keine zwei Transaktionen dürfen dasselbe Element gesperrt haben. Eine Transaktion kann eine Sperre auf  $X$  erst erwerben, wenn sie von einer anderen Transaktion freigegeben wurde.

Das bedeutet: Gibt es Aktionen  $l_i(X)$  gefolgt von  $l_j(X)$  in einem erweiterten Schedule  $S$ , so muss zwischen diesen beiden Sperren eine Aktion  $u_i(X)$  erfolgt sein.

Das Verfahren klingt einleuchtend, aber es löst das Serialisierbarkeitsproblem nicht. Es kann zwar verhindern, dass zwei Schreiboperationen nebenläufig auf ein Datenbankelement zugreifen (und z. B. das lost-update-Problem auftritt), nicht verhindern kann es jedoch, dass zeitlich verzahnte Schedules entstehen, die nicht serialisierbar sind, wie folgendes Beispiel zeigt.

Gegeben sind die Transaktionen  $T_1$  und  $T_2$  sowie  $T_1'$  und  $T_2'$ , die – angelehnt an das Beispiel von Seite 6–12 – nun um Sperren und Freigaben der Datenbankobjekte  $A$  und  $B$  angereichert werden.

$T_1$	$T_1'$	$T_2$	$T_2'$
$l_1(A)$	$l_1(A)$	$l_1(A)$	$l_1(A)$
$r_1(A) \rightarrow x$	$r_1(A) \rightarrow x$	$r_1(A) \rightarrow y$	$r_1(A) \rightarrow y$
$x := x + 100$	$x := x + 100$	$y := 2y$	$y := 2y$
$x \rightarrow w_1(A)$	$x \rightarrow w_1(A)$	$y \rightarrow w_1(A)$	$y \rightarrow w_1(A)$
$u_1(A)$	$l_1(B)$	$u_1(A)$	$l_1(B)$
$l_1(B)$	$u_1(A)$	$l_1(B)$	$u_1(A)$
$r_1(B) \rightarrow x$	$r_1(B) \rightarrow x$	$r_1(B) \rightarrow y$	$r_1(B) \rightarrow y$
$x := x + 100$	$x := x + 100$	$y := 2y$	$y := 2y$
$x \rightarrow w_1(B)$	$x \rightarrow w_1(B)$	$y \rightarrow w_1(B)$	$y \rightarrow w_1(B)$
$u_1(B)$	$u_1(B)$	$u_1(B)$	$u_1(B)$

Abbildung 49: Transaktionen im RWL-Modell.  $T_1$  und  $T_1'$  addieren 100 zu  $A$  und  $B$ ,  $T_2$  und  $T_2'$  multiplizieren  $A$  und  $B$  jeweils mit 2. Die Lock- und Unlock-Operationen  $l_i(X)$  und  $u_i(X)$  werden jeweils vom Scheduler eingefügt.  $T_1'$  und  $T_2'$  werden nach dem 2PL-Protokoll erzeugt.

Der Schedule  $S_1$  in Abbildung 51 lehnt sich an das Beispiel von Seite 6–12 an. Er zeigt einen legalen, aber nicht serialisierbaren Schedule. Das Sperren der Datenbankobjekte genügt nicht, um die Serialisierbarkeit zu gewährleisten.

Dieses Problem löst das Zwei-Phasen-Sperrprotokoll (2PL), bei dem innerhalb einer Transaktion zuerst alle Sperren angefordert werden und erst danach die erste angeforderte Sperre wieder freigegeben werden darf.

**Definition 6.7.1** (Zweiphasiges Sperren (2-Phase-Lock, 2PL)). *Gehen innerhalb eines Ablaufplanes bei allen Transaktionen alle Sperraktionen den Entsperraktionen voraus, nennt man dieses Sperrprotokoll **zweiphasiges Sperren (2-Phase-Lock, 2PL)**. Das Protokoll zeichnet die Existenz einer Auf- und einer Abbauphase aus.*

Im Einzelnen gilt:

- Der Scheduler darf nur Sperren bedienen, die zu einem legalen Schedule führen.
- Können angeforderte Sperren nicht zugeteilt werden, wird die entsprechende Transaktion verzögert, d. h. in eine Warteschlange eingereiht.
- Der Scheduler verwaltet eine (Element,TA-Nummer)-Tabelle.

Schedules, die nach diesem Prinzip aufgebaut sind, sind konfliktserialisierbar. Das zeigt z. B.  $S_2$  in Abbildung 51. Das Anfordern der Sperre  $l_2(B)$  führt dazu, dass die Operationen von  $T_2$  zuerst zurückgestellt werden – solange, bis  $T_1$  ein unlock auf dieser Datenbankvariable abgeschickt hat.

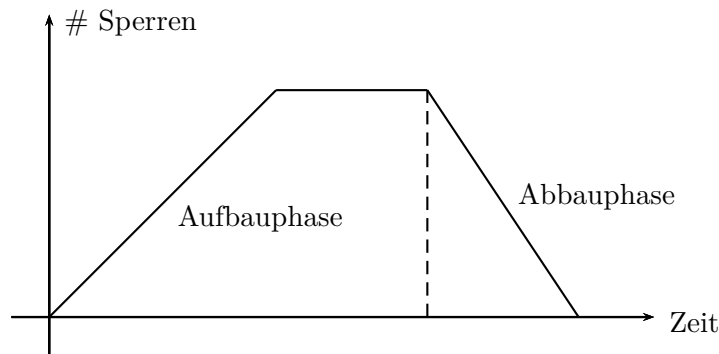


Abbildung 50: 2PL: Für jede Transaktion gibt es eine Aufbauphase, in der die Sperren angefordert werden, und eine Abbauphase, in der sie wieder freigegeben werden.

**Satz 6.7.1** (2PL und Konfliktserialisierbarkeit). *Ein nach dem Zweiphasenprotokoll aufgebauter Schedule ist konfliktserialisierbar.*

### Protokollanforderungen für 2PL

Der Scheduler muss beim Erzeugen eines Schedules gewisse Regeln einhalten, um ein zwei-Phasen-Sperrprotokoll zu erzeugen.<sup>76</sup>

#### Transaktionskonsistenz.

Lese und Schreibzugriffe werden durch passende Locks geschützt:

- Vor jedem  $r_i(X)$  gibt es ein  $rl_i(X)$  oder  $wl_i(X)$ , ohne dass dazwischen ein  $u_i(X)$  auftritt.
- Vor jedem  $w_i(X)$  gibt es ein  $wl_i(X)$  ohne dass dazwischen ein  $u_i(X)$  auftritt.
- Auf alle Sperren folgen entsprechende Entsperroperationen.

#### Zweiphasiges Sperren.

Erwerb von Sperren vor dem ersten Unlock: Kein  $rl_i(X)$  oder  $wl_i(X)$  hat vor sich eine Aktion  $u_i(Y)$  ( $\forall T_i$ )

#### Legale Schedules.

- Gibt es  $wl_i(X)$  in  $S$ , so kann kein  $wl_j(X)$  oder  $rl_j(X)$  folgen, ohne dass dazwischen ein  $u_i(X)$  kommt.
- Gibt es  $rl_i(X)$  in  $S$ , so kann es für  $i \neq j$  kein  $wl_j(X)$  geben, ohne dass ein  $u_i(X)$  dazwischen liegt.

<sup>76</sup>Das Protokoll berücksichtigt bereits feiner granulierte Sperren (siehe S. 6–36).

$S_1$				$S_2$			
$T_1$	$T_2$	$A$	$B$	$T_1'$	$T_2'$	$A$	$B$
<i>Startzustand</i>		25	25	<i>Startzustand</i>		25	25
$l_1(A)$ $r_1(A) \rightarrow x$ $x := x + 100$ $x \rightarrow w_1(A)$ $u_1(A)$		125		$l_1(A)$ $r_1(A) \rightarrow x$ $x := x + 100$ $x \rightarrow w_1(A)$ $l_1(B)$ $u_1(A)$		125	
	$l_2(A)$ $r_2(A) \rightarrow y$ $y := 2y$ $y \rightarrow w_2(A)$ $u_2(A)$	250			$l_2(A)$ $r_2(A) \rightarrow y$ $y := 2y$ $y \rightarrow w_2(A)$ $l_2(B)$	250	$T_2$ muss warten
	$l_2(B)$ $r_2(B) \rightarrow y$ $y := 2y$ $y \rightarrow w_2(B)$ $u_2(B)$		50	$r_1(B) \rightarrow x$ $x := x + 100$ $x \rightarrow w_1(B)$ $u_1(B)$		125	$T_2$ freigegeben
$l_1(B)$ $r_1(B) \rightarrow x$ $x := x + 100$ $x \rightarrow w_1(B)$ $u_1(B)$		150			$u_2(A)$ $r_2(B) \rightarrow y$ $y := 2y$ $y \rightarrow w_2(B)$ $u_2(B)$	250	
<i>Endzustand</i>		250	150	<i>Endzustand</i>		250	250

Abbildung 51: Der Schedule  $S_1$  ist legal, aber nicht serialisierbar. Schedule  $S_2$  dagegen ist serialisierbar; in diesem Schedule gehen alle Sperraktionen einer Transaktion den Entsperraktionen voraus.  $S_2$  folgt daher dem Zweiphasen-Sperrprotokoll.

### 6.7.2. Verklemmungen

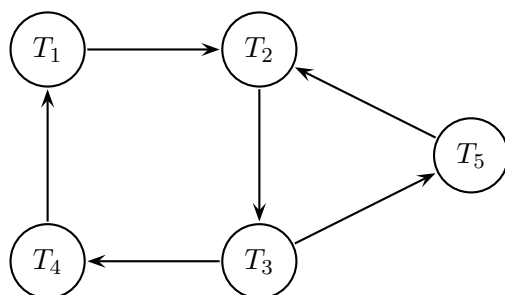
Mit Sperren einher geht allerdings auch die Gefahr der Verklemmung (deadlocks) einher. Eine Verklemmung tritt dann auf, wenn zwei nebenläufige Transaktionen wechselseitig auf die Freigabe einer Sperre warten und daher nicht weiterarbeiten können. Treten solche Verklemmungen auf, muss der Scheduler den Schedule vollständig zurückfahren (ROLLBACK). Das Erkennen von Verklemmungen kann auf unterschiedliche Arten geschehen ([KE09, S. 327]):

**Time-out-Strategie.** Reagiert eine bestimmte Transaktion innerhalb einer bestimmten Zeit  $T_W$  (z. B.  $T_W = 1s$ ) nicht, wird davon ausgegangen, dass die Transaktion verklemmt ist. In diesem Fall wird die betreffende TA zurückgefahren. Problematisch ist diese Methode, weil die Wartezeit das Systemverhalten entscheidend beeinflusst. Ist  $T_W$  zu klein gewählt, werden viele TAen abgebrochen, die gar nicht verklemmt waren, sondern nur auf Ressourcen gewartet haben (CPU, Speicher, Sperren, ...). Ist  $T_W$  zu groß, werden tatsächlich vorhandene Verklemmungen zu lange geduldet, was sich negativ auf die Performanz auswirkt.

$S_3$			
$T_1'$	$T_2'$	$A$	$B$
Startzustand		25	25
$l_1(A)$			
$r_1(A) \rightarrow x$			
	$l_2(B)$		
	$r_2(B) \rightarrow y$		
$x := x + 100$	$y := 2y$		
$x \rightarrow w_1(A)$	$y \rightarrow w_2(B)$	125	50
$l_1(B)$			
$T_1$ wartet auf $u_2(B)$			
	$l_2(A)$		
$T_2$ wartet auf $u_1(A)$			
...			

Abbildung 52: Der Schedule ist durch die verschränkte Anforderung von Sperren verklemmt. Dieses systemimmanente Problem lässt sich nicht ausräumen. In einem solchen Fall bleibt nur das ROLLBACK.

**Waits-For-Graph.** Auf der Basis von Sperranforderungen wird ein Wartegraph (Waits-For-Graph) erstellt, in dem die Knoten den derzeit im System aktiven Transaktionen entsprechen und die (gerichteten) Kanten eine Wartebeziehung darstellen. So wird die Kante  $(T_i, T_j)$  dann eingefügt, wenn  $T_i$  auf die Sperrfreigabe durch  $T_j$  wartet. Eine Verklemmung liegt genau dann vor, wenn der Wartegraph (mindestens) einen Zykel aufweist, wie der folgende Wartegraph mit zwei Zyklen zeigt.



Wird eine Verklemmung erkannt, muss eine TA aus dem Wartegraphenzyklus entfernt werden. Auch hierfür gibt es mehrere Strategien ([KE09, S. 328]). So kann man diejenige TA entfernen, bei der der Rücksetzaufwand minimal ist, oder diejenige, die die meisten Ressourcen belegt. Sind Transaktionen an mehreren Wartezyklen beteiligt (z. B.  $T_2$  oder  $T_3$ ), kann es sinnvoll sein, eine solche zu entfernen. Wichtig ist, eine vielseitige Strategie zu haben, damit nicht immer die gleiche Transaktion zurückgesetzt wird, um ein 'Verhungern' zu vermeiden.

### 6.7.3. Erweiterte Konzepte im 2PL

#### Lese- und Schreibsperrern

Exklusive Sperren sind ein ziemlich scharfes Geschütz. Gerade in Systemen, in denen häufiger gelesen als geschrieben wird, steigt durch exklusive Sperren die Gefahr einer Verklemmung deutlich an. Analog zum Fall zweier lesender Zugriffe auf DB-Objekte, den man im Rahmen der Konfliktserialisierbarkeit als Nicht-Konflikt identifiziert hat, wird das RWL-Modell weiter verfeinert. Man führt nun Lese- und Schreibsperrern ein: Das RWL-Modell wird dahingehend verfeinert, dass Lese- und Schreibsperrern –  $rl_i(X)$  und  $wl_i(X)$  – verwendet werden. Für jedes DB-Objekt kann es beliebig viele Lesesperren, aber *nur eine* Schreibsperre geben. Daher nennt man die Lesesperre oft auch **shared lock** und die Schreibsperre **exclusive lock**.

$rl_i(X)$  und  $wl_i(X)$

#### Striktes 2PL

Das 2PL erzeugt in jedem Fall serialisierbare Schedules. Allerdings kann es in manchen Fällen ein erzwungenes, kaskadierendes Rollback nicht verhindern. So muss im Beispiel des serialisierbaren Schedule  $S_2$  im Fall, dass  $T_1$  zurückgesetzt werden muss, auch  $T_2$  zurückgesetzt werden, weil es ein *dirty read* von  $A$  ausgeführt hat. Dies lässt sich vermeiden, wenn man zu 2PL noch die Forderung hinzufügt, dass es keine Schrumpfungsphase geben darf, sondern dass alle Sperren erst am Ende der Transaktion freigegeben werden.

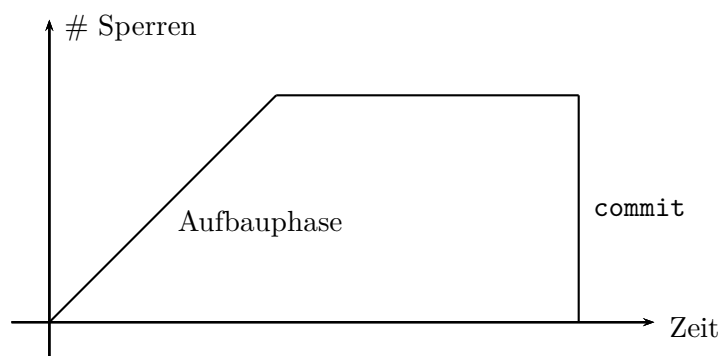


Abbildung 53: striktes 2PL: Es gibt nur noch eine Aufbauphase, die Sperren werden zusammen am TA-Ende freigegeben.

## 6.8. Übungszettel A-17

**Aufgabe 17.1.** Zeigen Sie, ob die nicht-serialisierbaren Schedules der Übungszettel 14 und 15 serialisierbar sind, wenn sie nach dem 2PL-Sperrprotokoll durchgeführt werden.

Erstellen Sie aus dem Schedule der Aufgabe 15.3 durch konfligierende Vertauschungen einen nicht-konfliktserialisierbaren Schedule und zeigen Sie, ob dieser mithilfe des 2PL-Sperrprotokolls serialisierbar wird oder in einem Deadlock endet.

## 6.9. Logging und Recovery

Der immense Wert, den ein DBMS für ein Unternehmen darstellt, kann in vielen Fällen gar nicht hoch genug eingeschätzt werden. Banken, Online-Shops, Handy-Netze oder die Bahn bzw. andere Reisebuchungsunternehmen wären im Fall eines (Total-)Ausfalles ihrer zugrundeliegenden Datenbasis handlungsunfähig. Auch Warenhäuser, die mittlerweile nahezu ausschließlich ihren Warenbestand, die Preise und das Kassensystem elektronisch verwalten, wären bei einem Verlust dieses Systems kaum in der Lage, weiter Waren an Kunden zu verkaufen. Selbst die Erfassung der Verkäufe zur Berechnung der Steuern erfolgt elektronisch. Das hat zur Folge, dass es oberstes Ziel sein muss, den Datenbestand und das die Daten verwaltende System zu sichern und im Fehlerfall schnellstmöglich und im Idealfall ohne Datenverlust wieder anlaufen lassen zu können.

Der Recovery-Vorgang ist recht komplex und erfordert das Zusammenspiel unterschiedlicher Komponenten. Der Transaktionsmanager nimmt die TAen entgegen, der Query-Prozessor optimiert die Anfragen, der LOG-Manager schreibt die Transaktions-LOGs, die für den Recovery-Prozess essentiell sind. Der Buffer-Manager ist das Bindeglied zwischen primärem Speicher (Hauptspeicher) und sekundärem Speicher (persistentem Speicher), das kurze Zugriffszeiten garantiert. Die Recovery-Komponente sorgt im Fehlerfall dafür, dass das System nach dem Wiederanlauf in einen konsistenten Zustand mündet.

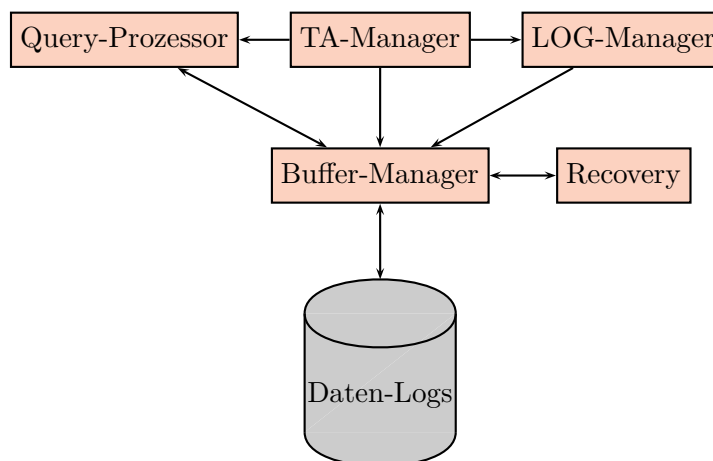


Abbildung 54: Die Systemarchitektur der Puffer- und LOG-Verwaltung

### 6.9.1. Fehlerklassifikation

Von welchen Fehlern soll die Rede sein? Es geht um Fehler, bei deren Eintreten das DBMS selbsttätig in der Lage sein soll, den letzten konsistenten Datenbankzustand wiederherzustellen. Man unterscheidet dabei im Wesentlichen drei Fehlerkategorien ([KE09]):

Recovery  
=  
Wiederherstellung

#### Lokale Fehler einer Transaktion.

Lokale Fehler sind solche, die zum Scheitern einer einzelnen Transaktion führen, den Rest des Systems hinsichtlich der Konsistenz aber nicht beeinflussen. Typische Vertreter sind:

- Fehler im Anwendungsprogramm (Absturz oder logischer Fehler),

- benutzergesteuerter Abbruch, weil z. B. das gewünschte Ergebnis nicht zustande kommt,
- systemgesteuerter Abbruch, weil z. B. eine Verklemmung aufgetreten ist.

Fehler dieser Art treten relativ häufig auf und müssen daher in sehr kurzer Zeit (typischerweise Millisekunden) behoben werden können. Das System macht dazu alle Änderungen an der Datenbasis, die von der aktiven TA verursacht wurden, rückgängig (siehe Kapitel *UNDO-Log*). Diesen Vorgang bezeichnet man als **lokales UNDO**.

#### Fehler mit Primärspeicherverlust.

Datenbanksysteme arbeiten nicht direkt auf persistentem (Sekundär-)Speicher sondern auf dem sogenannten Datenbankpuffer, der ein Teil des (primären) Hauptspeichers ist. Fällt dieser aus – wie das u. a. bei einem Stromausfall geschieht – geht der Inhalt dieses Puffers verloren. Änderungen, die bislang nur im DB-Puffer vorhanden sind, sind damit vernichtet. Das Transaktionsparadigma verlangt, dass

- alle bereits im sekundären Speicher vorhandenen Änderungen von nicht-abgeschlossenen TAen rückgängig gemacht werden (**globales UNDO**),
- alle noch nicht im sekundären Speicher vorhandenen Änderungen bereits abgeschlossener TAen nachvollzogen werden (**globales REDO**).

Diese Fehlerart geht von einem intakten Sekundärspeicher aus, der sich allerdings in einem inkonsistenten Zustand befindet, der mit Hilfe von LOG-Informationen in einen konsistenten Zustand gebracht werden soll.

Fehler dieser Art treten in mittelmäßig großen Zeitabständen auf (typischerweise einigen Tagen). Eine Wiederherstellung sollte in der Größenordnung weniger Minuten liegen.

#### Fehler mit Sekundärspeicherverlust.

Der Sekundärspeicher wird in der Regel durch eine (einzelne) Festplatte oder einen Festplattenverbund realisiert. Handelt es sich um ein RAID-System<sup>77</sup> mit Redundanz, also z. B. RAID 1 oder RAID 5, lassen sich bereits viele Hardware-Fehler abfangen. Prominente Ursachen für Fehler im sekundären Speicher sind

- *head crash* der Platte(n),
- Feuer, Erdbeben, wodurch die Platte(n) zerstört wird(werden),
- Fehler in Systemprogrammen (z. B. im Plattentreiber), die zu einem Datenverlust führen.

Solche Fehler treten in der Regel selten auf (typischerweise Monate oder Jahre), dennoch müssen unbedingt Vorkehrungen getroffen werden, um die Datenbasis in einem solchen Fall wieder in einen konsistenten Zustand zu bringen. Ein DBMS kann dies nicht mehr aus eigenen Ressourcen heraus leisten. Hierfür wird eine Datenbasis-Archivkopie und ein LOG-Archiv benötigt, das das DBMS einlesen kann; beides sollte räumlich getrennt vom Original aufbewahrt werden, um nicht sämtliche Informationen zu verlieren.

<sup>77</sup>RAID: Redundant Array Of Independent Disks



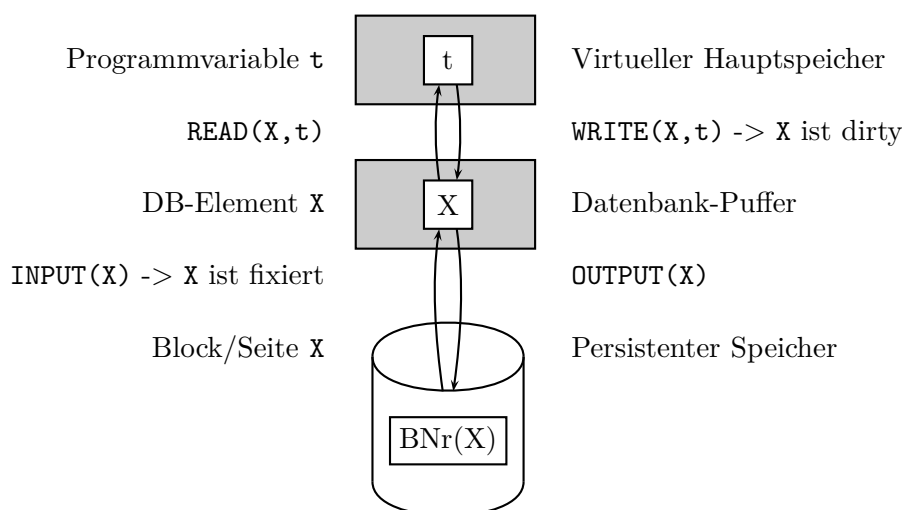


Abbildung 55: Speicherhierarchie der Logverwaltung

### 6.9.2. Speicherhierarchie und Einbringstrategien

Durch die Trennung von Primär- und Sekundärspeicher wirken sich Änderungen an der Datenbasis nicht sofort in der materialisierten Datenbasis aus. Die Transaktionen arbeiten auf den Elementen im Puffer, die typischerweise in sog. Datenseiten (von üblicherweise 8KB Größe) organisiert sind. Bei einem TA-Zugriff sind meist mehrere solcher Seiten involviert. Sobald ein Zugriff stattfindet, wird die Seite vom Puffer-Manager fixiert, so dass diese nicht aus dem Puffer verdrängt werden kann. Werden Daten geändert, wird die betreffende Seite als modifiziert (*dirty*) gekennzeichnet und die Fixierung aufgehoben; der Puffer-Manager weiß nun, dass diese Seite nicht mehr mit der entsprechenden Seite des Sekundärspeichers übereinstimmt, und sorgt beizeiten dafür, dass sie festgeschrieben wird. Eine dauerhafte Fixierung der Seiten würde dazu führen, dass der Primärspeicher irgendwann voll ist. Neben den bereits bekannten Speicheroperationen `READ` und `WRITE`, die Datenobjekte aus dem Pufferspeicher des DBMS in den (für das Anwendungsprogramm relevanten) Hauptspeicher holen, existieren nun noch Operationen, die vom Puffer-Manager ausgelöst werden (siehe Abbildung 55). Eine schreibt Seiten des Puffers in den Sekundärspeicher (`OUTPUT`), eine andere holt Seiten bei

Daten werden  
seitenweise organisiert  
und kopiert

Syntax	Semantik
<code>INPUT(X)</code>	<code>= get(BlockNr(X), X)</code>
<code>READ(X, t)</code>	<code>= IF X ∉ DB-Puffer THEN INPUT(X)</code> <code>memcpy(X, t)</code>
<code>WRITE(X, t)</code>	<code>= IF X ∉ DB-Puffer THEN INPUT(X)</code> <code>memcpy(t, X)</code>
<code>OUTPUT(X)</code>	<code>= flush(X, BlockNr(X))</code>

Abbildung 56: Operationen der Puffer- und Logverwaltung und ihre Semantik.

Bedarf aus dem Sekundärspeicher und schreibt sie in den Puffer (INPUT).



### Beispiel zur LOG-Verwaltung

Nebenstehendes Beispiel zeigt den Ablauf einer Transaktion, die zwei Datenbank-Variablen X und Y verdoppelt. Dabei wird neben dem Anwendungs-Hauptspeicher der DB-Puffer und der Zustand des Sekundärspeichers notiert. Erst beim Kommando OUTPUT wird der Inhalt einer Datenbank-Variable auf den Sekundärspeicher propagiert.

Operation	t	X	Y	B(X)	B(Y)
READ(X,t)	4	4		4	5
$t := 2 * t$	8	4		4	5
WRITE(X,t)	8	8		4	5
READ(Y,t)	5	8	5	4	5
$t := 2 * t$	10	8	5	4	5
WRITE(Y,t)	10	8	10	4	5
OUTPUT(X)	10	8	10	8	5
OUTPUT(Y)	10	8	10	8	10

### Einbringstrategien

Wann der Puffer-Manager eine Pufferseite auf den Sekundärspeicher schreibt, hängt von der verwendeten Strategie ab. Hierbei wird betrachtet, ob und wann eine Seite im Puffer durch eine neu zu ladende Datenseite ersetzt wird. Bevor eine Seite im Puffer ersetzt wird, wird sie in den Sekundärspeicher geschrieben und ihr Speicherplatz im Primärspeicher freigegeben.

### Einbringstrategien für nicht abgeschlossene Transaktionen.

In Bezug auf aktive, also noch nicht abgeschlossene (commit) Transaktionen gibt es zwei Strategien:

- $\neg steal$ : Die Ersetzung von Seiten, die von nicht abgeschlossenen Transaktionen verändert wurden, bleibt ausgeschlossen bis zum Abschluss der Transaktion. So lange blockieren sie den Primärspeicher.
- *steal*: Jede nicht fixierte Seite ist prinzipiell ein Kandidat für die Ersetzung durch neu einzulagernde Seiten. Damit können Veränderungen schneller im Sekundärspeicher eingetragen werden und der Primärspeicher wird schneller frei.

Bei der  $\neg steal$ -Strategie kann es nie vorkommen, dass Änderungen einer nicht abgeschlossenen TA auf den Sekundärspeicher geschrieben werden. Bei einem Rollback (dieser aktiven TA) muss der Sekundärspeicher nicht betrachtet werden, da dort Änderungen erst nach einem COMMIT festgeschrieben werden können. Bei der *steal*-Strategie muss dagegen auch in einem solchen Fall der Zustand der Datenbasis auf dem Sekundärspeicher betrachtet werden und ggf. ein UNDO auf den materialisierten Seiten durchgeführt werden, um wieder einen konsistenten Zustand zu erhalten.

*steal* erfordert  
UNDO-Logging

### Einbringstrategien für abgeschlossene Transaktionen.

Für bereits abgeschlossene Transaktionen gibt es ebenfalls zwei Strategien:

- *force*: Alle von einer TA modifizierten Seiten werden beim Eintreffen eines COMMIT-Befehls auf den Sekundärspeicher geschrieben und machen den Primärspeicher frei.
- $\neg force$ : Selbst nach einem TA-COMMIT werden die modifizierten Seiten nicht notwendigerweise propagiert, sondern erst, wenn sie ohnehin ersetzt werden müssten. Damit müssen oft genutzte Datenseiten nicht jedes Mal neu in den Sekundärspeicher geschrieben werden ("Flaschenhals").

Die  $\neg force$ -Strategie erfordert andere Protokolleinträge aus einer separaten Protokoll-datei, um die noch nicht propagierten Änderungen dort nachvollziehen zu können (REDO). Bei der *force*-Strategie ist das nicht nötig, weil die Änderungen aller abgeschlossenen Transaktionen auf dem Sekundärspeicher bereits festgeschrieben wurden.

Die verschiedenen Kombinationen der Einbring-Strategien liefern Konsequenzen für die Art des Loggings, wie folgende Übersicht zeigt:

	<i>force</i>	$\neg force$
$\neg steal$	<ul style="list-style-type: none"> <li>• kein REDO</li> <li>• kein UNDO</li> </ul>	<ul style="list-style-type: none"> <li>• REDO</li> <li>• kein UNDO</li> </ul>
<i>steal</i>	<ul style="list-style-type: none"> <li>• kein REDO</li> <li>• UNDO</li> </ul>	<ul style="list-style-type: none"> <li>• REDO</li> <li>• UNDO</li> </ul>

Es scheint auf den ersten Blick sehr verlockend, die Strategien  $\neg steal$  und *force* zu kombinieren. Dagegen spricht:



- Viele Seiten werden von mehreren TAen verwendet und residieren oft lange im Puffer. Die erzwungene Propagierung verhindert ihre sofortige Wiederverwendung.
- Die Speicherung muss atomar erfolgen ("ganz oder gar nicht"). Ein Systemabsturz während der Festschreibung würde die Datenbasis ohne weitere Mechanismen in einem inkonsistenten Zustand hinterlassen.
- Die Umsetzung dieser Kombination ist nicht möglich, wenn zwei TAen nebenläufig auf **unterschiedlichen Elementen einer Seite** arbeiten.

### 6.9.3. UNDO-Logging

Die materielle Datenbasis auf dem Sekundärspeicher enthält meist nicht den jüngsten konsistenten Zustand der Datenbasis, u. U. nicht einmal einen konsistenten Zustand. Deshalb benötigt man Zusatzinformationen, die an anderer Stelle gespeichert werden, in der sogenannten **LOG-Datei** – auch Protokolldatei genannt.

Eine LOG-Datei besteht aus zeilenweisen Einträgen von LOG-Datensätzen (log records) in einem Transaktions-LOG, einer (Binär-)Datei, die oft aus Effizienz- aber auch Sicherheitsgründen auf einer anderen Platte als die Daten liegt und als Ringpuffer implementiert ist.<sup>78</sup> Der LOG-Manager kennt dabei folgende Operationen:

<sup>78</sup>Siehe dazu [KE09, S. 295]. In der Scientific-Linux Distribution (Version 6.5) liegen die standardmäßig 16MB großen Transaktions-LOGs unterhalb von `/var/lib/pgsql/data/pg_xlog/`.

<BOT T>	Beginn der Transaktion $T$
<COMMIT T>	Die Transaktion $T$ ist erfolgreich beendet. Vorher muss der LOG-Manager die zugehörigen Puffer geschrieben haben.
<ABORT T>	Die Transaktion $T$ war nicht erfolgreich. Der Transaktionsmanager muss sicherstellen, dass keine Änderungen auf dem Sekundärspeicher landen bzw. bleiben.
<T,X,w>	Protokollierung des Wertes $w$ des DB-Elementes $X$ . Die Änderung fand nur im Primärspeicher statt.

Für das UNDO-Logging gibt es zwei Regeln:

**UNDO-Logging-Regel 1:** Ändert eine Transaktion  $T$  den Wert eines DB-Elementes  $X$ , dann muss der Update-Record  $\langle T, X, \text{alt} \rangle$  **vor** der Änderung von  $X$  auf den Sekundärspeicher festgeschrieben werden.  $\text{alt}$  ist dabei der Wert der Variablen, den  $X$  **vor** der Änderung hatte.

Da während der Transaktion der Sekundärspeicher **mehrfach** geändert werden kann (Verdrängung), ist jedes Mal die Festschreibung des Logs davor notwendig.

**UNDO-Logging-Regel 2:** Wird eine Transaktion  $T$  gültig (COMMIT), so muss der COMMIT-Record **nach** dem Zurückschreiben aller Datenbankelemente von  $T$  – natürlich möglichst schnell – in das LOG geschrieben werden.

Auch das Schreiben auf dem LOG geschieht zuerst nur auf dem Primärspeicher und muss noch persistent gemacht werden. Der LOG-Manager benötigt dafür die Anweisung FLUSHLOG, mit der der Puffermanager angewiesen wird, die Änderungen im LOG persistent zu machen.



### Beispiel zum UNDO-Logging

Das Beispiel von oben wird nun um LOG-Daten ergänzt. Nach der UNDO-Logging-Regel 1 muss das LOG festgeschrieben werden, **bevor** die Datensätze auf der Platte landen. Die LOG-Einträge protokollieren dabei den alten Wert des Datenelementes. Nach der UNDO-Logging-Regel 2 muss nach dem Festschreiben der Daten der COMMIT-Eintrag in das LOG geschrieben werden.

Nr	Operation	t	X	Y	B(X)	B(Y)	UNDO-Log
1							<BOT T>
2	READ(X,t)	4	4		4	5	
3	$t := 2 * t$	8	4		4	5	
4	WRITE(X,t)	8	8		4	5	<T,X,4>
5	READ(Y,t)	5	8	5	4	5	
6	$t := 2 * t$	10	8	5	4	5	
7	WRITE(Y,t)	10	8	10	4	5	<T,Y,5>
8	FLUSHLOG						
9	OUTPUT(X)	10	8	10	8	5	
10	OUTPUT(Y)	10	8	10	8	10	
11							<COMMIT T>
12	FLUSHLOG						

In diesem Beispiel werden die Daten in den Schritten 9 und 10 auf dem Sekundärspeicher gültig gemacht. Zu diesem Zeitpunkt sind alle LOG-Einträge (wegen des FLUSHLOG) bereits auf der Platte vorhanden. Weil die Variablen zeilenweise festgeschrieben werden, muss die ganze Seite durch Setzen von Sperren vor dem Zugriff durch andere Transaktionen gesperrt werden, was leider die Performanz verringert.

### Recovery eines UNDO-LOGs

Um im Fehlerfall die Aktionen des UNDO-LOGs rückgängig zu machen, geht der Recovery-Manager nach folgendem Muster vor:

- (a) Durchmustern des Transaktions-LOG **von hinten** nach Transaktionen (COMMITTED oder UNCOMMITTED).
- (b) Gibt es ein <COMMIT T>, sind alle Datenbankelemente nach UNDO-Regel 2 bereits auf die Platte geschrieben. Folglich ist für T **nichts zu tun**.
- (c) Gibt es ein <BOT T> ohne zugehöriges <COMMIT T>, ist unklar, ob die Daten bereits gültig gemacht wurden. In diesem Fall werden alle <T,X,w> blind rückgängig gemacht. Dadurch werden alle Datenbankelemente auf den alten Stand gebracht.

Einen Algorithmus findet man in Abbildung 57. Um den Algorithmus besser zu verstehen,

```
// UNDO-Algorithmus
LOG von *hinten* durchsuchen
FORALL T DO
  IF exists <T,X,w> THEN
    IF not exists <COMMIT T> THEN
      // alten Wert zurückschreiben
      WRITE(X,w)
      <ABORT T> ins LOG
    ELSE
      // nichts zu tun
    END
  END
END
FLUSHLOG
```

Abbildung 57: UNDO-Algorithmus für das Recovery nicht abgeschlossener Transaktionen

betrachten wir Beispiel 1 genauer. Es soll analysiert werden, was beim Recovery passiert, wenn ein Fehler auftritt, der den Primärspeicher betrifft. Wir versetzen uns in die Lage des Recovery-Managers während des Recoverys:

- (a) Der Fehler tritt nach Zeile 12 auf.  
Im LOG ist ein <COMMIT T> zu finden, also ist nichts zu tun.
- (b) Der Fehler tritt zwischen Zeile 11 und 12 auf.

Der Recovery-Manager findet kein `<COMMIT T>` im LOG (auf der Platte!) und macht daher blind alle Operationen `<T,X,w>` rückgängig. In diesem Fall wäre das nicht nötig gewesen, weil sich die materielle Datenbasis in einem konsistenten Zustand befand.

- (c) Der Fehler tritt zwischen Zeile 10 und 11 auf.  
gleiche Situation wie in (b); hier wurde das `<COMMIT T>` erst gar nicht geschrieben.
- (d) Der Fehler tritt zwischen Zeile 8 und 10 auf.  
Der Recovery-Manager findet kein `<COMMIT T>` im LOG und macht alle Operationen `<T,X,w>` rückgängig. Das ist in jedem Fall nötig, weil nicht klar ist, welche Datenbank-Variable bereits auf den Sekundärspeicher propagiert wurde und welche nicht.
- (e) Der Fehler tritt vor Zeile 8 auf.  
Das LOG wurde nicht festgeschrieben, also gibt es auch keine Einträge `<T,X,w>`. Es fand aber auch kein OUTPUT statt. Es ist nichts zu tun, da keine Daten auf die Platte geschrieben wurden.

#### 6.9.4. REDO-Logging



Ein großes Problem des UNDO-Loggings ist das erzwungene Propagieren von Daten auf den Sekundärspeicher **vor** einem COMMIT. Hier stellt die Seitenverwaltung des Puffermanager einen erheblichen Flaschenhals dar.

Eine andere Herangehensweise stellt das sogenannte **REDO-Logging** dar. Während man beim UNDO-Logging nur die Operationen nicht abgeschlossener Transaktionen rückgängig macht und abgeschlossene Transaktionen ignoriert, geht man beim REDO-Logging genau anders herum vor: Nicht abgeschlossene Transaktionen werden vollständig ignoriert, abgeschlossene Transaktionen werden beim Wiederanlauf wiederholt.

**REDO-Logging-Regel 1:** Ändert eine Transaktion  $T$  den Wert eines DB-Elementes  $X$ , dann muss der Update-Record `<T,X,neu>` **vor** der Änderung von  $X$  auf den Sekundärspeicher geschrieben werden. `neu` ist dabei der Wert der Variablen, den  $X$  **nach** der Änderung hat.

**REDO-Logging-Regel 2:** Wird eine Transaktion  $T$  gültig (COMMIT), so muss der COMMIT-Record **vor** dem Zurückschreiben aller Datenbankelemente von  $T$  in das LOG geschrieben werden.

Das bedeutet, dass eine Transaktion erst dann enden kann, wenn alle Änderungen im LOG protokolliert wurden. Dieses als **deferred update** bezeichnete Vorgehen erfordert einen großen Cache, weil die gesamte Transaktion erst dann auf die Platte geschrieben werden kann, wenn ein COMMIT erreicht ist.<sup>79</sup>

#### Recovery eines REDO-LOGs

Um im Fehlerfall die Aktionen des REDO-LOGs wirksam werden zu lassen, geht der Recovery-Manager nach folgendem Muster vor:

- (a) Durchmustern des Transaktions-LOG **von vorne** nach Transaktionen (COMMITTED oder UNCOMMITTED).

---

<sup>79</sup>Siehe auch [EN07].

- (b) Gibt es ein  $\langle \text{BOT } T \rangle$  ohne zugehöriges  $\langle \text{COMMIT } T \rangle$ , sind keine Änderungen von  $T$  persistent. Folglich ist **nichts zu tun**.
- (c) Gibt es ein  $\langle \text{COMMIT } T \rangle$ , ist unklar, ob die Daten bereits gültig gemacht wurden. In diesem Fall werden alle  $\langle T, X, w \rangle$  erneut geschrieben (und persistent gemacht). Dadurch werden alle Datenbankelemente auf den ursprünglich gewünschten Stand gebracht.

Den REDO-Algorithmus findet man in Abbildung 58.

### Beispiel zum REDO-Logging

Wir betrachten wieder das gleiche Beispiel wie beim UNDO-Logging, erstellen aber jetzt ein REDO-Log. Nach der REDO-Logging-Regel 2 dürfen die Daten auf dem Sekundärspeicher erst dann festgeschrieben werden, wenn das COMMIT im LOG festgeschrieben ist. Die LOG-Einträge protokollieren den neuen Wert des Datenelementes.



Nr	Operation	t	X	Y	B(X)	B(Y)	REDO-Log
1							$\langle \text{BOT } T \rangle$
2	READ(X,t)	4	4		4	5	
3	$t := 2 * t$	8	4		4	5	
4	WRITE(X,t)	8	8		4	5	$\langle T, X, 8 \rangle$
5	READ(Y,t)	5	8	5	4	5	
6	$t := 2 * t$	10	8	5	4	5	
7	WRITE(Y,t)	10	8	10	4	5	$\langle T, Y, 10 \rangle$
8							$\langle \text{COMMIT } T \rangle$
9	FLUSHLOG						
10	OUTPUT(X)	10	8	10	8	5	
11	OUTPUT(Y)	10	8	10	8	10	

### 6.9.5. UNDO/REDO-Logging

In den meisten Umgebungen wird wegen der großen Flexibilität das sogenannte **UNDO/REDO-Logging** auf der Basis der Einbringstrategien *steal*/ $\neg$ *force* angewendet. Dabei geschieht das Recovery nach dem Wiederanlauf in drei Phasen:

**Analyse.** Die LOG-Datei wird von Anfang bis Ende analysiert, um die Menge der abgeschlossenen Transaktionen (engl. *winner*) und die Menge der nicht-abgeschlossenen Transaktionen (engl. *loser*) zu ermitteln.

**REDO-Phase.** Es werden alle protokollierten Änderungen – sowohl die der *winner* als auch die der *loser* – in der Reihenfolge ihrer Ausführung in die Datenbasis eingebracht.

**UNDO-Phase.** Die Änderungsoperationen der *loser*-Transaktionen werden in umgekehrter Reihenfolge ihrer ursprünglichen Ausführung rückgängig gemacht.

Das REDO aller Operationen ist nötig, weil das Wiederherstellen der Datenbasis idempotent sein muss.<sup>80</sup> Würde nämlich ein Crash während des Recovery-Vorganges auftreten,

<sup>80</sup>Eine Funktion  $f$  heißt idempotent, falls  $f \circ f = f$  gilt.

müsste der dieser wiederholt werden. Dabei muss aber das Ergebnis der wiederhergestellten Datenbasis das gleiche sein, wie nach (erfolgreichem) einmaligem Recovery.<sup>81</sup>

WAL  
=  
Write Ahead Log

Bei dieser Systemkonfiguration (*steal/¬force*) ist es unabdingbar, dass das sogenannte **WAL-Prinzip** eingehalten wird (*Write-Ahead-Log*). Die folgenden beiden Regeln müssen dabei zwingend befolgt werden:

1. Bevor eine Transaktion festgeschrieben wird, müssen alle zu ihr gehörenden LOG-Einträge festgeschrieben werden.
2. Bevor eine modifizierte Seite ausgelagert werden darf, müssen alle LOG-Einträge, die zu ihr gehören, in das LOG geschrieben werden.

Für das grundlegende Verständnis von Logging und Recovery soll es uns aber ausreichen, UNDO- und REDO-Protokolle isoliert zu verstehen. In der am Ende des Kapitels erwähnten Fachliteratur findet man ausführliche Erläuterungen zu diesem Thema, insbesondere auch zu den Feinheiten des Performance-Tunings, das sich mit Themen wie verketteten Listen beschäftigt, um effizient eine LOG-Datei rückwärts zu durchlaufen.

#### 6.9.6. Sicherungspunkte

Das Transaktions-LOG enthält viele (verschränkt eingetragene) LOG-Records von vielen Transaktionen. Ein Recovery kann daher das System für längere Zeit zum Stillstand bringen. Um das zu verhindern, gibt es das Konzept der Sicherungspunkte, sogenannte **checkpoints**. Das System lehnt für kurze Zeit die Annahme von Transaktionen ab, bearbeitet aktive Transaktionen noch bis zu ihrem Ende (COMMIT oder ABORT), vermerkt den Sicherungspunkt im Protokoll, schreibt dieses fest und fährt mit dem normalen Datenbankbetrieb fort.

- (1) STOP der Annahme weiterer Transaktionen
- (2) Warte das Ende aller aktiven TAen ab
- (3) FLUSHLOG
- (4) <CHKPT> ins LOG
- (5) FLUSHLOG
- (6) RESUME der Annahme von Transaktionen

Während dieser Zeit ist das DBMS für Anfragen nicht verfügbar!<sup>82</sup>

<sup>81</sup>Siehe [KE09, S. 297] sowie Kapitel 10.5 auf Seite 298 ebenda.

<sup>82</sup>Sicherungspunkte werden von den meisten DBMSen in regelmäßigen Abständen automatisch erstellt. In **postgresql** geschieht dies in der Standardeinstellung spätestens alle 5 Minuten ([Sch09, S. 475]). Alternativ kann ein Checkpoint (vom DBA) auch manuell an der **psql**-Konsole durch das Kommando **CHECKPOINT** ausgelöst werden.



## Literatur zu Kapitel 6

- [EN07] ELMASRI, Ramez A. ; NAVATHE, Shamkant B.: *Fundamentals of Database Systems*. 5. Auflage. Boston etc : Pearson Intl. Edition, 2007. – ISBN 978-0-32-141506-6
- [EN09] ELMASRI, Ramez A. ; NAVATHE, Shamkant B.: *Grundlagen von Datenbanksystemen*. 3. Auflage. München : Pearson-Studium, 2009. – ISBN 978-3-8689-4012-1
- [KE09] KEMPER, Alfons ; EICKLER, André: *Datenbanksysteme: Eine Einführung*. 7. Auflage. München : Oldenbourg Wissenschaftsverlag, 2009. – ISBN 978-3-486-59018-0
- [Sch09] SCHERBAUM, Andreas: *PostgreSQL. Datenbankpraxis für Anwender, Administratoren und Entwickler*. 1. Auflage. München : Open Source Press, 2009. – ISBN 978-3-937514-69-7
- [SKS11] SILBERSCHATZ, Abraham ; KORTH, Henry F. ; SUDARSHAN, S.: *Database system concepts*. 6. Auflage. New York NY : McGraw-Hill, 2011 (McGraw-Hill international editions. Computer science series). – ISBN 978-007-128959-7
- [Vos08] VOSSEN, Gottfried: *Datenmodelle, Datenbanksprachen und Datenbankmanagementsysteme*. 5. Auflage. München : Oldenbourg, 2008. – ISBN 978-3-4862-7574-2

```
// REDO-Algorithmus
LOG von *vorne* durchsuchen
FORALL <T,X,w> DO
  IF exists <COMMIT T> THEN
    // neuen Wert erneut schreiben
    WRITE(X,w)
  ELSE
    // nichts zu tun
  END
END
FLUSHBUFFER
FORALL T DO
  IF not exists <COMMIT T> THEN
    <ABORT T> ins LOG
  END
END
FLUSHLOG
```

Abbildung 58: REDO-Algorithmus für das Recovery bereits abgeschlossener Transaktionen

## 6.10. Übungszettel A-18

**Aufgabe 18.1.** In einem UNDO-Log mit zwei Transaktionen  $T_1$  und  $T_2$  steht folgendes:

1	$\langle BOT\ T_1 \rangle$	4	$\langle T_2, Y, 20 \rangle$	7	$\langle COMMIT\ T_2 \rangle$
2	$\langle T_1, X, 10 \rangle$	5	$\langle T_1, Z, 30 \rangle$	8	$\langle T_1, V, 50 \rangle$
3	$\langle BOT\ T_2 \rangle$	6	$\langle T_2, U, 40 \rangle$	9	$\langle COMMIT\ T_1 \rangle$

Welche Aktionen führt der Recovery-Manager durch (incl. der Änderungen auf Platte und im LOG), falls es einen Crash gibt mit dem letzten Log-Eintrag

- (a)  $\langle BOT\ T_2 \rangle$
- (b)  $\langle COMMIT\ T_2 \rangle$
- (c)  $\langle T_1, V, 50 \rangle$
- (d)  $\langle COMMIT\ T_1 \rangle$

**Aufgabe 18.2.** Lösen Sie die vorangegangene Aufgabe unter der Voraussetzung, dass es sich um ein REDO-Log handelt.

### Aufgabe 18.3.

- (a) Analysieren Sie den Schedule

$$S = (r_2(A), w_2(A), w_1(A), w_3(A), r_2(B), w_2(B))$$

Begründen Sie im einzelnen, ob Schedule  $S$  von einem zweiphasig sperrenden Scheduler ausgeführt werden könnte.

- (b) Bei der Datenbank-Pufferverwaltung gibt es für die Verdrängung von Seiten die Strategien *steal* und  $\neg$ *steal* und für das Verhalten zum *COMMIT*-Zeitpunkt die Strategien *force* und  $\neg$ *force*. Erläutern Sie die 4 möglichen Kombinationen der Strategien und deren Folgen.
- (c) Demonstrieren Sie anhand eines Beispiels, dass man die Strategien *force* und  $\neg$ *steal* nicht kombinieren kann, wenn TAs nebenläufig auf unterschiedliche Elemente einer Pufferseite zugreifen. Entwerfen Sie eine verzahnte Ausführung zweier Transaktionen, bei denen die Kombination *force*/ $\neg$ *steal* ausgeschlossen ist.



## 7. Anwendungsprogramme

Nachdem wir – bis auf Details der Implementierung – Datenbankmanagementsysteme gründlich kennen gelernt haben, wollen wir uns nun der Anwendungsschicht zuwenden. Hier bietet nur SQL wegen seiner hohen Verbreitung genügend Softwareprodukte, die eine Kommunikation mit einem darunter liegenden DBMS ermöglichen. Wir wollen uns in diesem Kapitel mit zwei Zugängen genauer befassen: Die Programmierung einer Anwendung mit GO, die auf ein darunter liegendes DBMS zugreift, und eine PHP-gesteuerte Web-Anwendung, die auf einem fernen (oder lokalen) Webserver liegt, der über eine Anbindung an ein darunter liegendes DBMS verfügt. In beiden Fällen müssen wir uns auch um technische Details kümmern, damit eine solche Kombination (idealerweise nicht nur unter Linux) läuft; da hier oft der Teufel im Detail steckt, beschränken wir uns im Wesentlichen auf das DBMS

postgresql.<sup>83</sup>

Um auch ein bisschen Praxis auf Administrationsebene zu gewinnen – das wird der eine oder die andere im Schulalltag ohnehin bewerkstelligen müssen – runden wir die Vorarbeit zum unterrichtsbezogenen Datenbankpraktikum (UDBP) ab durch die Installation eines postgresQL-Servers auf einem Raspberry-Pi ab, den wir gleich im Praktikum benutzen werden.

Und nicht zuletzt soll die Arbeit am UDBP – die in Gruppen durchgeführt werden soll – unterstützt werden durch ein Versionsverwaltungssystem (wir verwenden Git), das auch dem Softwarepraktikum zugute kommen wird.

### 7.1. postgresQL und GO

postgresQL kann über viele verschiedene Sprachen angesprochen werden: C, C#, JAVA, Delphi, Perl, Python, aber auch über Go. Für letztgenannte stehen verschiedene Bibliotheken zur Verfügung, von denen viele allerdings einen proprietären Zugang darstellen. Erfährt Go ein Update, ist nicht klar, ob die Bibliothek daran angepasst wird oder bereits angepasst wurde oder ob überhaupt noch eine Anpassung durchgeführt wird. Der sichere Weg ist, die Go-eigene Bibliothek `database/sql` zu verwenden und nur den Treiber zum Datenbanksystem, das angesprochen werden soll, anzupassen. Diesen Ansatz verfolgt das Modul SQL, dessen Installation im Zusammenspiel mit dem DBMS postgresQL im Folgenden genau beschrieben wird.

#### 7.1.1. Installation

Die Installation gestaltet sich recht einfach und besteht aus folgenden Schritten:

1. Installation des postgresQL-Treibers für Go

Der angegebenen Treiber kann durch einen Kommandozeilenbefehl installiert werden.

```
go get github.com/lib/pq
```

<sup>83</sup>Insbesondere im Zusammenhang mit PHP sind wir damit ein Stück abseits des Mainstreams. Hier dominiert die Kombination PHP/MySQL, zu der es eine breite Palette an Literatur gibt. Das Zusammenspiel zwischen postgresQL und PHP ist nur in wenigen Büchern zu finden ([GS02], [DBB06], [Boe03],[MS05]). Der Einstieg ist allerdings nicht so schwierig, wie es zunächst scheint. Auch hier laufen die Installationen in der Regel ohne größere Probleme ab.

Dadurch wird das angegebene github-Repository nach `~/go/src` geklont. Es stellt lediglich den Treiber für den Zugriff auf das PostgreSQL-System dar und sollte nicht direkt verwendet werden.

## 2. Installation des Wrappers SQL

Der Wrapper stellt eine schmale Schnittstelle zur Verfügung, wie sie im unterrichtlichen Kontext zum Einsatz kommen könnte. Die Datei `SQL.tgz` wird von der Materialiensite zum Kurs heruntergeladen und **in das Heimatverzeichnis** entpackt.

Alternativ zu 1. und 2. kann auch die Datei `SQL-komplett.tgz` von der Materialiensite heruntergeladen und entpackt werden. Die Datei enthält bereits das oben angegebene github-Repository.

Hinweis: Unter Windows kann es sein, dass der Zugriff auf die lokale Datenbank abgewiesen wird. In diesem Fall fügt man der Datei `pg_hba.conf` den Eintrag

```
host all all 127.0.0.1 trust
```

hinzu.

### 7.1.2. Erste Schritte

Nach erfolgreicher Installation kann man mit Hilfe eines ersten Programmes prüfen, ob eine Verbindung zur Datenbank aufgebaut werden kann. Kern des Programmes ist die Anweisung

```
SQL.PgSQL (<Verbindungsdaten>)
```

in der festgelegt wird, auf welchem Weg auf eine Datenbank zugegriffen wird. Dabei ist *Verbindungsdaten* eine Zeichenkette, die aus Schlüsselwort-Wert-Paaren besteht, die durch Leerzeichen getrennt werden und die Form `<Schlüsselwort>=<Wert>` haben. In Tabelle 59 sind die derzeit unterstützten Schlüsselwörter zusammen mit einem Beispiel und dem Standardwert angegeben. Der Standardwert wird verwendet, wenn das jeweilige Schlüsselwort in der Verbindungsdaten-Zeichenkette nicht auftritt. Das nachfolgende erste Beispiel zeigt, wie eine Verbindung als Benutzer `lewein` zur Datenbank `lwb` auf dem lokalen Datenbankserver über den Standardport 5432 aufgebaut wird. Beachtenswert ist auch die `defer`-Anweisung, die sicherstellt, dass die Datenbankverbindung in jedem Fall wieder geschlossen wird – eine Aktion, die gerne vergessen wird.

```
// Autor:  Oliver Schäfer
// Datum:  01.04.2015
// Zweck:  Verbindungstest mit neuem SQL-Modul

package main
import "SQL"

func main() {
    var conn SQL.Verbindung

    conn = SQL.PgSQL ("user=lewein dbname=lwb")
    defer conn.Beenden ()
}
```

Schlüsselwort	Bedeutung	Standardwert	Beispiel
<b>host</b>	Name/IP-Adresse des Datenbankservers	localhost	host=192.168.0.51
<b>port</b>	Portnummer der DB-Verbindung	5432	port=55432
<b>dbname</b>	Name der Datenbank	<Benutzer>	dbname=lwb
<b>user</b>	Name des Benutzers		user=lewein
<b>password</b>	Passwort der Verbindung		password=niewel
<b>timeout</b>	Timeout in Sekunden, 0 = deaktiviert	0	timeout=60

Abbildung 59: Schlüsselwörter und ihre Standardwerte, die für den Aufbau einer Verbindung über SQL zur Verfügung stehen.

```
if conn == nil {  
    println ("Verbindung fehlgeschlagen")  
} else {  
    println ("Verbindung erfolgreich")  
}  
}
```

Nachdem eine Verbindung mit dem Datenbankserver aufgebaut wurde, lassen sich SQL-Anweisungen an den Server senden. Dabei sind zwei Arten zu unterscheiden:

(a) Anweisungen ohne Rückgabewerte

Anweisungen, die nur auf dem Datenbankserver Aktionen hinterlassen und keine Informationen aus der Datenbank holen, werden mit der Methode `Ausfuehren` an den DB-Server geschickt. Beispiele für solche Kommandos sind `INSERT` oder `CREATE DATABASE`.

```
conn.Ausfuehren ("CREATE TABLE Test ( a INTEGER, b INTEGER );")
```

Handelt es sich dabei um ein `INSERT`-, `UPDATE` oder `DELETE`-Kommando, liefert der Befehl die Anzahl der betroffenen Tupel zurück.

```
numRows = conn.Ausfuehren ("INSERT INTO Test VALUES (1, 3);")
```

Um längere Anfragen übersichtlich notieren zu können, bietet Go die Möglichkeit, RAW-Strings einzusetzen. Ein RAW-String wird von *Gravis* bzw. *Backticks* oder *Back-quotes* ` eingeschlossen. Innerhalb eines RAW-Strings darf jedes Zeichen (außer dem Backtick) verwendet werden, insbesondere hat der Backslash \ keine besondere Bedeutung.

```
conn.Ausfuehren (`CREATE TABLE Test (  
                a INTEGER,  
                b INTEGER  
            `);`)
```

(b) SELECT-Anweisungen

Eine SELECT-Anweisung liefert als Antwort eine Ergebnismenge (engl. *result-set*), die vom Programm weiterverarbeitet werden soll. Um eine SELECT-Anweisung an den DB-Server zu schicken und die Ergebnismenge in Empfang zu nehmen, benötigt man den Befehl

```
conn.Anfrage (<SELECT-Statement>)
```

Als Rückgabewert erhält man eine Variable vom Typ `SQL.Ergebnismenge`, die wiederum weitere Methoden zu ihrer Verarbeitung mitbringt.

### 7.1.3. Arbeiten mit Ergebnismengen

Für das Arbeiten mit Ergebnismengen bringt das Paket `SQL` die notwendigen Methoden mit. Konzeptionell liefert `Anfrage` eine Ergebnismenge, die man mit Hilfe eines `Cursor` von vorne beginnend durchgehen kann. Dazu dient die Methode `GibtTupel`, die – sofern vorhanden – den `Cursor` auf das nächste ungelesene `Tupel` setzt. War das `Tupel` bereits das letzte, bleibt der `Cursor` auf diesem `Tupel` stehen; die Funktion liefert in diesem Fall `false`, um anzuzeigen, dass es keine (weiteren) `Tupel` gibt.

Ist der `Cursor` durch Anwendung von `GibtTupel` auf ein gültiges `Tupel` gesetzt worden, können die darin enthaltenen Attributwerte mit Hilfe der Methode `LeseTupel` ausgelesen werden. Dazu übergibt man der Funktion einfach die Adressen der lokalen Variablen, in die der Wert gespeichert werden soll.

## 7.2. `mysql` und `GO`

Mehr eine Machbarkeitsstudie als eine systematische Anleitung zum Umgang mit `MySQL`-Datenbanken stellt das vorliegende Kapitel dar. Es arbeitet noch mit der letzten, aktuellen Version von `Scientific Linux (SL 7)`, eine Distribution, die nicht mehr weiterentwickelt, aber bis mind. 2027 noch unterstützt wird. Die Unterschiede zur `Fedora`-Distribution sind für unsere Zwecke aber minimal. Es zeigt, wie – praktisch mit gleicher Syntax – der Zugriff auf eine `MySQL`-Datenbank erfolgen kann und auch, was unter der `SL`-Distribution noch zu tun ist, damit der `MySQL`-Datenbankserver läuft.<sup>84</sup>

Unter `SL` muss zuerst der `MySQL`-Server installiert werden. Seit `SL 7` wird nicht mehr `MySQL` sondern `MariaDB` verwendet, daher haben die benötigten Pakete eventuell unerwartete Namen.

Der erste Schritt ist die Installation des Servers und der Client-Programme für lokalen DB-Zugriff:

<sup>84</sup>Für eine Installation und Einrichtung unter `Windows` (oder `Mac`) mag man sich mit Hilfe einschlägiger Literatur und der Hilfe aus Foren selbst zurechtfinden.



```
# yum install mariadb-server  
# yum install mariadb
```

Um den Datenbank-Dämon zu starten und dafür zu sorgen, dass dies bei jedem Systemstart automatisch geschieht, muss root die Kommandos

```
# systemctl start mariadb  
# systemctl enable mariadb
```

an der Konsole absetzen. Anschließend müssen noch Benutzer und Datenbank jeweils mit Namen lewein erstellt werden. Anders als bei PostgreSQL ist unter MySQL der Systemadministrator root gleichzeitig Datenbankadministrator. Ruft man als root den Befehl

```
# mysql
```

auf, wird mit den MySQL-Kommandos

```
MariaDB [(none)]> CREATE DATABASE lewein;  
MariaDB [(none)]> CREATE USER lewein@localhost;  
MariaDB [(none)]> GRANT ALL PRIVILEGES ON lewein.* TO 'lewein'@'%';  
MariaDB [(none)]> FLUSH PRIVILEGES;  
MariaDB [(none)]> \q
```

die Datenbank und der Benutzer eingerichtet. Anschließend kann man sich als lewein mittels

```
$ mysql -D lewein
```

an der Datenbank lewein anmelden und z. B. mit \. <datei.sql> SQL-Kommandos einer externen Datei einlesen.

Um auf eine MySQL-Datenbank mit Go zugreifen zu können, muss nur noch der MySQL-Datenbanktreiber installiert werden. Im weltweiten Netz findet man dazu unter der Adresse <https://github.com/golang/go/wiki/SQLDrivers> eine Liste verfügbarer Treiber. Mit

```
go get github.com/go-sql-driver/mysql
```

wird der notwendige MySQL-Treiber installiert.<sup>85</sup> Anschließend kann man – die Existenz entsprechender Tabellen in der Datenbank vorausgesetzt – mit nahezu dem gleichen Code wie im Fall von PostgreSQL mit der MySQL-Datenbank kommunizieren. Einziger Unterschied ist die Funktion, mit der die Verbindung zur Datenbank aufgebaut wird. Das Beispiel `connect.go` unterscheidet sich nur in Zeile 11:

```
conn = SQL.MySQL ("lewein@lwb")
```

<sup>85</sup>Dieser Treiber ist bereits im Archiv `SQL-komplett.tgz` enthalten, das man auf der Materialiensite finden kann. Nebenbei bemerkt: Wird eine neue Version von Go installiert, müssen alle per `go get <paketname>` installierten Pakete mittels `go build -a <paketname>` neu kompiliert werden.

# Anhang

## A. Planung

Stand:  
20. Februar 2023

### Datenbanksysteme a (WiSe 2022/2023)

Termin	Datum	Thema
1	29.08.2022	Literatur, Überblick, Dateisystem vs. DBS, DBMS
2	05.09.	Konzeptuelle Modellierung: ER-Modell I
3	12.09.	ER-Modell II, Varianten und Alternativen
4	19.09.	Logische Datenmodellierung: Relationenmodell I
5	26.09.	Relationenmodell II, Transformation ER → Rel
6	10.10.	Relationenalgebra
7	17.10.	Relationale Anfragesprachen: TRC und DRC
Herbstferien		
8	07.11.	Prolog, Datalog, Deduktive DBS
9	14.11.	Entwurfstheorie I: Funktionale Abhängigkeiten
10	21.11.	Entwurfstheorie II: Normalformen
11	28.11.	SQL I
12	05.12.	SQL II
13	12.12.	Komplexere SQL-Anfragen, Varianten
14	19.12.	Zusammenfassende Übungen zur Klausur
Weihnachtsferien		
15	09.01.2023	<b>Klausur Datenbanksysteme</b>
16	16.01.	Transaktionsmanagement, Schedules, Serialisierbarkeit
17	23.01.	Konfliktserialisierbarkeit, topologisches Sortieren (1. <b>Nachklausur DBS</b> )
Winterferien		
18	06.02.	Transaktionen – Zwei-Phasen-Sperrprotokoll (2PL)

Neben dem regulären Klausurtermin am 09.01.2023 gibt es einen ersten Nachschreibtermin am 23.01.2023 und einen zweiten am 13.02.2023.

**Datenbanksysteme b (SoSe 2023)**

Termin	Datum	Thema
1	13.02.	Logging und Recovery I (2. Nachklausur DBS)
2	20.02.	Logging und Recovery II
3	27.02.	GO und PostgreSQL I
4	06.03.	GO und PostgreSQL II / PHP und PostgreSQL I
5	13.03.	PHP und PostgreSQL II
6	20.03.	Formulare in PHP I
7	27.03.	Formulare in PHP II + Einstieg in das UDBP
Osterferien		
8-9	17.04.-24.04.	Arbeit am UDBP
internat. Tag der Arbeit		
10-12	08.05.-22.05.	Arbeit am UDBP
Pfingstferien		
13-14	05.06.-12.06.	Arbeit am UDBP
15	19.06.	Abgabe der schriftlichen Ausarbeitung
16	26.06.	Präsentationen und Praxis
17	03.07.	Präsentationen und Praxis
18	10.07.	Präsentationen und Praxis

## B. Literaturverzeichnis

### Empfehlungen

- [EN09] ELMASRI, Ramez A. ; NAVATHE, Shamkant B.: *Grundlagen von Datenbanksystemen*. 3. Auflage. München : Pearson-Studium, 2009. – ISBN 978-3-8689-4012-1
- [Jar10] JAROSCH, Helmut: *Datenbankentwurf: Eine beispielorientierte Einführung für Studenten und Praktiker*. 3. Auflage. Braunschweig : Vieweg, 2010 (Vieweg Ausbildung und Studium). – ISBN 978-3-8348-0955-1
- [KE09] KEMPER, Alfons ; EICKLER, André: *Datenbanksysteme: Eine Einführung*. 7. Auflage. München : Oldenbourg Wissenschaftsverlag, 2009. – ISBN 978-3-486-59018-0
- [SKS11] SILBERSCHATZ, Abraham ; KORTH, Henry F. ; SUDARSHAN, S.: *Database system concepts*. 6. Auflage. New York NY : McGraw-Hill, 2011 (McGraw-Hill international editions. Computer science series). – ISBN 978-007-128959-7
- [SSH10] SAAKE, Gunter ; SATTLER, Kai-Uwe ; HEUER, Andreas: *Datenbanken: Konzepte und Sprachen*. 4. Auflage. Heidelberg : mitp, 2010. – ISBN 978-3-8266-9057-0
- [Vos08] VOSSEN, Gottfried: *Datenmodelle, Datenbanksprachen und Datenbankmanagementsysteme*. 5. Auflage. München : Oldenbourg, 2008. – ISBN 978-3-4862-7574-2

### Übersichtsdarstellungen

- [Atz99] ATZENI, Paolo: *Database systems: Concepts, languages & architectures*. London : McGraw-Hill, 1999. – ISBN 0-07709-500-6
- [Bis95] BISKUP, Joachim: *Grundlagen von Informationssystemen*. Braunschweig : Vieweg, 1995 (Vieweg-Lehrbuch Informatik). – ISBN 3-52805-494-8
- [CB10] CONOLLY, Thomas ; BEGG, Carolyn: *Database Systems*. 5. Auflage. Boston : Pearson Education, 2010. – ISBN 978-0-321-52306-8
- [Dat04] DATE, Chris J.: *An introduction to database systems*. 8. Auflage. Boston, Mass. : Pearson Addison-Wesley, 2004. – ISBN 0-32118-956-6
- [EN09] ELMASRI, Ramez A. ; NAVATHE, Shamkant B.: *Grundlagen von Datenbanksystemen*. 3. Auflage. München : Pearson-Studium, 2009. – ISBN 978-3-8689-4012-1
- [Gei09] GEISLER, Frank: *Datenbanken: Grundlagen und Design*. 3. Auflage. Heidelberg : mitp-Verlag, 2009. – ISBN 978-3-8266-5529-6
- [GMUW09] GARCIA-MOLINA, Hector ; ULLMAN, Jeffrey D. ; WIDOM, Jennifer: *Database systems: The complete book*. 2. Auflage. Upper Saddle River, NJ : Pearson Education International/Prentice Hall, 2009. – ISBN 978-0-1313-5428-9

- [Joh97] JOHNSON, James L.: *Database: Models, languages, design*. New York : Oxford Univ. Press, 1997. – ISBN 0-19510-783-7
- [KBL06] KIFER, Michael ; BERNSTEIN, Arthur ; LEWIS, Philip M.: *Database systems: An application-oriented approach*. 2. Auflage. Boston : Pearson/Addison Wesley, 2006. – ISBN 978-0-3213-1256-3
- [KE09] KEMPER, Alfons ; EICKLER, André: *Datenbanksysteme: Eine Einführung*. 7. Auflage. München : Oldenbourg Wissenschaftsverlag, 2009. – ISBN 978-3-486-59018-0
- [KK98] KIESSLING, Werner ; KÖSTLER, Gerhard: *Multimedia-Kurs Datenbanksysteme*. Berlin : Springer, 1998. – ISBN 3-54063-836-9
- [Kro95] KROENKE, David M.: *Database processing: Fundamentals, design, and implementation*. 5. Auflage. Englewood Cliffs, NJ : Prentice-Hall, 1995 (Prentice Hall international editions). – ISBN 0-13320-128-7
- [Kud07] KUDRASS, Thomas (Hrsg.): *Taschenbuch Datenbanken*. Leipzig : Hanser Verlag, 2007. – ISBN 978-3-446-40944-6
- [KW06] KEMPER, Alfons ; WIMMER, Martin: *Übungsbuch Datenbanksysteme*. München : Oldenbourg, 2006. – ISBN 978-3-4865-7967-3
- [LS87] LOCKEMANN, Peter C. (Hrsg.) ; SCHMIDT, Jocachim W. (Hrsg.): *Datenbank-Handbuch*. Berlin : Springer Verlag, 1987. – ISBN 3-530-10741-X
- [MU99] MATTHIESSEN, Günter ; UNTERSTEIN, Michael: *Relationale Datenbanken und SQL: Konzepte der Entwicklung und Anwendung*. 4. Auflage. München : Addison-Wesley, 1999. – ISBN 3-82731-167-5
- [Ram98] RAMAKRISHNAN, Raghu: *Database management systems*. Boston, Mass. : WCB/McGraw-Hill, 1998. – ISBN 0-07115-508-2
- [SE07] SUMATHI, S. ; ESAKKIRAJAN, S.: *Fundamentals of Relational Database Management Systems*. Berlin, Heidelberg : Springer-Verlag Berlin Heidelberg, 2007 (Springer-11647 /Dig. Serial])
- [SKS11] SILBERSCHATZ, Abraham ; KORTH, Henry F. ; SUDARSHAN, S.: *Database system concepts*. 6. Auflage. New York NY : McGraw-Hill, 2011 (McGraw-Hill international editions. Computer science series). – ISBN 978-007-128959-7
- [SSH10] SAAKE, Gunter ; SATTLER, Kai-Uwe ; HEUER, Andreas: *Datenbanken: Konzepte und Sprachen*. 4. Auflage. Heidelberg : mitp, 2010. – ISBN 978-3-8266-9057-0
- [Ull88] ULLMAN, Jeffrey D.: *Principles of Computer Science Series*. Bd. Vol. I: *Principles of Database and Knowledge-Base Systems*. Rockville, Md. : Computer Science Press, 1988. – ISBN 0-71678-158-1
- [Vos08] VOSSEN, Gottfried: *Datenmodelle, Datenbanksprachen und Datenbankmanagementsysteme*. 5. Auflage. München : Oldenbourg, 2008. – ISBN 978-3-4862-7574-2

## Konzeptuelle Modellierung

- [BCN98] BATINI, Carlo ; CERI, Stefano ; NAVATHE, Shamkant B.: *Conceptual database design: An entity-relationship approach*. 6. Auflage. Redwood City, Calif. : Benjamin/Cummings, 1998 (The Benjamin/Cummings series on database systems and applications). – ISBN 0-80530-244-1
- [Bek92] BEKKE, Johan H. t.: *Semantic data modeling*. New York : Prentice Hall, 1992. – ISBN 0-13806-050-9
- [Bue08] BUENO, Gérard: *Conception méthodique des bases de données*. Paris : Ellipses, DL 2008 (Technosup). – ISBN 978-2-7298-3870-6
- [Deb98] DEBENHAM, John K.: *Knowledge engineering: Unifying knowledge base and database design*. Berlin : Springer, 1998 (Artificial intelligence). – ISBN 3-54063-765-6
- [HM08] HALPIN, Terry ; MORGAN, Tony: *Information modeling and relational database*. 2. Auflage. Amsterdam : Morgan Kaufmann, 2008. – ISBN 978-0-1237-3568-3
- [Hoh93] HOHENSTEIN, Uwe: *Teubner-Texte zur Informatik*. Bd. 4: *Formale Semantik eines erweiterten Entity-Relationship-Modells*. Stuttgart : Teubner, 1993. – ISBN 3-81542-052-0
- [Jar10] JAROSCH, Helmut: *Datenbankentwurf: Eine beispielorientierte Einführung für Studenten und Praktiker*. 3. Auflage. Braunschweig : Vieweg, 2010 (Vieweg Ausbildung und Studium). – ISBN 978-3-8348-0955-1
- [Kud88] KUDLICH, Hermann: *Datenbank-Design*. Wien : Springer, 1988 (Springers angewandte Informatik). – ISBN 0-38782-018-3
- [Oli07] OLIVÉ, Antoni: *Conceptual Modeling of Information Systems*. Berlin, Heidelberg : Springer-Verlag Berlin Heidelberg, 2007 (Springer Verlag)
- [RR] ROLAND, Gabriel ; RÖHRS, Heinz-Peter
- [Sta05] STAUD, Josef L.: *Datenmodellierung und Datenbankentwurf*. Berlin etc. : Springer, 2005. – ISBN 978-3-540-20577-7
- [Teo96] TEOREY, Toby J.: *Database modeling & design: The fundamental principles*. 2. Auflage. San Francisco, Calif : Morgan Kaufmann Publishers, 1996. – ISBN 1-55860-294-1
- [Teo09] TEORY, Toby J.: *Database Design. Know it All*. Burlington : Morgan Kaufmann, 2009. – ISBN 978-0-12-374630-6
- [Tha00] THALHEIM, Bernhard: *Entity-relationship modeling: Foundations of database technology*. Berlin : Springer, 2000. – ISBN 3-54065-470-4

## Relationale Datenbanksysteme

- [AD93] ATZENI, Paolo ; DEANTONELLIS, Valeria: *Relational database theory*. Redwood City, Calif. : Benjamin/Cummings Publ., 1993. – ISBN 0–80530–249–2
- [AHV96] ABITEBOUL, Serge ; HULL, Richard ; VIANU, Victor: *Foundations of Databases*. Reading, Mass. : Addison-Wesley, 1996. – ISBN 0–20153–771–0
- [Cod91] CODD, Edgar F.: *The relational model for database management: Version 2*. Reading, Mass. : Addison-Wesley, 1991. – ISBN 0–20114–192–2
- [Dat05] DATE, Chris J.: *Database in depth: Relational theory for practitioners*. 1. Auflage. Beijing : O'Reilly, 2005 (Theory in practice). – ISBN 978–0–5961–0012–4
- [Die01] DIETRICH, Suzanne W.: *Understanding relational database query languages*. Upper Saddle River, N.J. : Prentice Hall, 2001. – ISBN 0–13028–652–4
- [KK93] KANDZIA, Peter ; KLEIN, Hans-Joachim: *Reihe Informatik*. Bd. Bd. 79: *Theoretische Grundlagen relationaler Datenbanksysteme*. Mannheim : B.I. Wissenschaftsverlag, 1993. – ISBN 3–41114–891–8
- [Léo92] LÉONARD, Michel: *Database design theory*. Basingstoke : Macmillan, 1992 (MacMillan computer science series). – ISBN 0–33353–813–7
- [Mai83] MAIER, David: *The theory of relational databases*. Rockville, Maryland : Computer Science Press, 1983. – ISBN 0–91489–442–0
- [Mau98] MAURER, Hermann: *From databases to hypermedia: With 26 CAI lessons*. Berlin : Springer, 1998. – ISBN 3–54063–754–0
- [MR94] MANNILA, Heikki ; RÄIHÄ, Kari-Jouko: *The design of relational databases*. Wokingham : Addison-Wesley, 1994. – ISBN 978–0–2015–6523–2
- [MTC07] MATA-TOLEDO, Ramon A. ; CUSHMAN, Pauline K.: *Schaum's Repetitorien*. Bd. 8373: *Relationale Datenbanken: Das Übungsbuch. Mit 153 Tabellen*. 1. Auflage. Bonn : mitp, 2007. – ISBN 978–3–8252–8373–5
- [Par89] PAREDAENS, Jan: *EATCS monographs on theoretical computer science*. Bd. 17: *The structure of the relational database model*. Berlin : Springer, 1989. – ISBN 0–38713–714–9
- [Pre07] PREISS, Nikolai: *Entwurf und Verarbeitung relationaler Datenbanken: Eine durchgängige und praxisorientierte Vorgehensweise*. München : Oldenbourg, 2007 (Wirtschaftsinformatik kompakt). – ISBN 978–3–4865–8369–4
- [Sch04] SCHUBERT, Matthias: *Datenbanken: Theorie, Entwurf und Programmierung relationaler Datenbanken*. 1. Auflage. Stuttgart : Teubner, 2004. – ISBN 3–51900–505–0

## Deduktive Datenbanksysteme

- [CGH94] CREMERS, Armin B. ; GRIEFAHN, Ulrike ; HINZE, Ralf: *Deduktive Datenbanken: Eine Einführung aus der Sicht der logischen Programmierung*. Braunschweig : Vieweg, 1994 (Künstliche Intelligenz). – ISBN 978-3-5280-4700-9
- [CGT90] CERI, Stefano ; GOTTLOB, Georg ; TANCA, Letizia: *Logic Programming and Databases*. Berlin : Springer Verlag, 1990. – ISBN 3-540-51728-6
- [Das92] DAS, Subrata K.: *Deductive databases and logic programming*. Wokingham : Addison-Wesley, 1992 (International series in logic programming). – ISBN 0-20156-897-7
- [Ull89] ULLMAN, Jeffrey D.: *Principles of Computer Science Series*. Bd. Vol. II, The New Technologies: *Principles of Database and Knowledge-Base Systems*. Rockville, Md. : Computer Science Press, 1989. – ISBN 071678162X

## Objekt-Datenbanksysteme

- [Ban92] BANCILHON, François: *Building an object-oriented database system: The story of O2*. San Mateo, Calif. : Morgan Kaufmann Publ., 1992 (The Morgan Kaufmann series in data management systems). – ISBN 1-55860-169-4
- [Gep97] GEPPERT, Andreas: *Objektorientierte Datenbanksysteme: Ein Praktikum*. 1. Auflage. Heidelberg : dpunkt Verlag für digitale Technologie, 1997. – ISBN 3-92099-362-4
- [Hug93] HUGHES, John G.: *Object-oriented databases*. 6. Auflage. New York, NY : Prentice Hall, 1993 (Prentice Hall international series in computer science). – ISBN 0-13629-874-5

## Verteilte Datenbanksysteme

- [BG92] BELL, David ; GRIMSON, Jane: *Distributed database systems*. Wokingham : Addison-Wesley, 1992 (International computer science series). – ISBN 0-20154-400-8
- [Dad96] DADAM, Peter: *Verteilte Datenbanken und Client/Server-Systeme: Grundlagen, Konzepte und Realisierungsformen*. Berlin : Springer, 1996. – ISBN 3-54061-399-4
- [Kum06] KUMAR, Vijay: *Mobile Database Systems*. Wiley, 2006. – ISBN 978-0-4714-6792-2
- [OV99] OSZU, M. T. ; VALDURIEZ, Patrick: *Distributed Database Systems*. Upper Saddle River : Prentice Hall, 1999. – ISBN 978-0-1365-9707-0



## Implementierung und Optimierung von Datenbanksystemen

- [HR99] HÄRDER, Theo ; RAHM, Erhard: *Datenbanksysteme: Konzepte und Techniken der Implementierung. Mit 15 Tabellen.* Berlin : Springer, 1999. – ISBN 3-54065-040-7
- [LL95] LANG, Stefan M. ; LOCKEMANN, Peter C.: *Datenbankeinsatz.* Berlin : Springer, 1995. – ISBN 3-54058-558-3
- [LTN07] LIGHTSTONE, Sam ; TEOREY, Toby ; NADEAU, Tom: *Physical Database Design. The database professional's guide to exploiting indexes, views, storage, and more.* CA : Morgan Kaufman, 2007. – ISBN 987-0-1236-9389-1
- [Sch03] SCHNEIDER, Markus: *Implementierungskonzepte für Datenbanksysteme.* Berlin : Springer Verlag, 2003. – ISBN 978-3-5404-1962-4
- [SH99] SAAKE, Gunter ; HEUER, Andreas: *Datenbanken: Implementierungstechniken.* 1. Auflage. Bonn : mitp-Verlag, 1999. – ISBN 3-82660-513-6

## Nebenläufigkeit und Transaktionen

- [BL93] BERNSTEIN, Arthur J. ; LEWIS, Philip M.: *Concurrency in programming and database systems.* Boston : Jones and Bartlett, 1993. – ISBN 086720205X
- [BN09] BERNSTEIN, Philip A. ; NEWCOMER, Eric: *Principles of Transaction Processing for the Systems Professional.* 2. Auflage. San Francisco, CA : Morgan Kaufman Publishers, 2009. – ISBN 978-1-55860-623-4
- [Cho98] CHORAFAS, Dimitris N.: *Transaction Management: Managing complex transactions and sharing distributed databases.* Palgrave Macmillan, 1998. – 328 S. – ISBN 978-0-3122-1018-3
- [Elm92] ELMAGARMID, Ahmed K.: *Database transaction models for advanced applications.* San Mateo, Calif. : Kaufmann, 1992 (The Morgan Kaufmann series in data management systems). – ISBN 1-55860-214-3
- [GR02] GRAY, Jim ; REUTER, Andreas: *Transaction processing: Concepts and techniques.* 9. Auflage. San Francisco, Calif. : Morgan Kaufmann, 2002 (The Morgan Kaufmann series in data management systems). – ISBN 1-55860-190-2
- [Pap86] PAPADIMITRIOU, Christos H.: *The theory of database concurrency control.* Rockville, Md. : Computer Science Press, 1986 (Principles of computer science series). – ISBN 0-88175-027-1
- [Wei88] WEIKUM, Gerhard: *Transaktionen in Datenbanksystemen. Fehlertolerante Steuerung paralleler Abläufe.* Bonn : Addison Wesley Deutschland, 1988. – ISBN 978-3-9251-1891-3

## XML und Semantisches Web

- [Hit08] HITZLER, Pascal: *Semantic Web: Grundlagen*. Berlin, Heidelberg : Springer-Verlag Berlin Heidelberg, 2008 (Springer 11774 Dig. Serial)]
- [KM02] KLATTKE, Meike ; MEYER, Holger: *XML & Datenbanken. Konzepte, Sprachen und Systeme*. Heidelberg : dpunkt Verlag, 2002. – 428 S. – ISBN 978-3-8986-4148-7
- [KST02] KAZAKOS, Wassilios ; SCHMIDT, Andreas ; TOMCZYK, Peter: *Datenbanken und XML. Konzepte, Anwendungen, Systeme*. Berlin : Springer Verlag, 2002 (Reihe Xpert.press). – ISBN 978-3-5404-1956-3
- [Lau05] LAUSEN, Georg: *Datenbanken: Grundlagen und XML-Technologien*. 1. Auflage. München : Elsevier Spektrum Akademischer Verlag, 2005 (Hochschultaschenbuch). – ISBN 3-82741-488-1
- [Rah03] RAHM, Erhard: *Web & Datenbanken: Konzepte, Architekturen, Anwendungen*. 1. Auflage. Heidelberg : dpunkt-Verlag, 2003. – ISBN 978-3-8986-4189-0
- [Sch02] SCHÖNING, Harald: *XML und Datenbanken. Konzepte und Systeme*. München : Hanser Verlag, 2002. – ISBN 978-3-4462-2008-9

## Geoinformationssysteme

- [MECL08] MITCHELL, Tyler ; EMDE, Astrid ; CHRISTL, Arnulf ; LANG, Jorgen: *Web Mapping mit Open Source-GIS-Tools*. Köln : O'Reilly, 2008. – ISBN 978-3-8972-1723-2
- [NM07] NETELER, Markus ; MITASOVA, Helena: *Open Source GIS: A Grass Approach*. 3. Auflage. Berlin : Springer Verlag, 2007. – 406 S. – ISBN 978-0387357676
- [RSV02] RIGAUX, Philippe ; SCHOLL, Michel ; VOISARD, Agnès: *Spatial databases: With applications to GIS*. San Francisco, Calif. : Morgan Kaufmann Publishing, 2002 (The Morgan Kaufmann series in data management systems). – ISBN 1-55860-588-6
- [RT08] RAMM, Frederik ; TOPF, Jochen: *OpenStreetMap: Die freie Weltkarte nutzen und mitgestalten*. 1. Auflage. Berlin : Lehmanns Media, 2008. – ISBN 978-3-8654-1262-1

## Data-Mining, Information Retrieval und Data-Warehouse

- [CWC09] CHAKRABARTI, Soumen ; WITTEN, Ian H. ; COX, Earl: *Data Mining: Know it All*. San Francisco, CA : Morgan Kaufman, 2009. – 496 S. – ISBN 978-0-1237-4629-0
- [Fer03] FERBER, Reginald: *Information Retrieval: Suchmodelle und Data-Mining-Verfahren für Textsammlungen und das Web*. 6. Auflage. Heidelberg : dpunkt-Verlag, 2003. – ISBN 3-89864-213-5

- [KST05] KUMAR, Vipin ; STEINBACH, Michael ; TAN, Pang-Ning: *Introduction to Data Mining*. Addison Wesley, 2005. – ISBN 978
- [Run09] RUNKLER, Thomas A.: *Data Mining. Methoden und Algorithmen intelligenter Datenanalyse*. Wiesbaden : Vieweg, 2009. – 165 S. – ISBN 978-3-8348-0850-5
- [WF05] WITTEN, Ian H. ; FRANK, Eibe: *Data Mining. Practical Machine Learning Tools and Techniques*. 2. Auflage. San Francisco, CA : Morgan Kaufman, 2005 (Morgan Kaufman Series in Data Management). – ISBN 978-0-1208-8407-0

## SQL

- [BBSL08] BROUARD, Frédéric ; BRUCHEZ, Rudi ; SOUTOU, Christian ; LARROUSSE, Nicolas: *SQL*. Paris : Pearson éducation France, 2008 (Collection Synthex). – ISBN 978-2-7440-7318-2
- [BHR07] BEAULIEU, Alan ; HEYMANN-REDER, Dorothea: *Einführung in SQL*. 1. Auflage. Beijing : O'Reilly, 2007. – ISBN 978-3-8972-1443-9
- [Bül91] BÜLTZINGSLOEWEN, Günter: *SQL-Anfragen: Optimierung für parallele Bearbeitung*. Berlin, Heidelberg, New York, London, Paris, Tokyo, Hong Kong, Barcelona, Budapest : Springer, 1991 (FZI-Berichte Informatik). – ISBN 0-38754-252-3
- [Cel08] CELKO, Joe: *Joe Celko's thinking in sets: Auxiliary, temporal, and virtual tables in SQL*. Amsterdam : Morgan Kaufmann/Elsevier, 2008 (The Morgan Kaufmann series in data mangement systems). – ISBN 978-0-1237-4137-0
- [CO93] CANNAN, Stephen J. ; OTTEN, Gerard A. M.: *SQL - the standard handbook: Based on the new SQL standard (ISO 9075:1992(E))*. London : McGraw-Hill, 1993. – ISBN 0-07707-664-8
- [CR07] CUMMING, Andrew ; RUSSEL, Gordon: *SQL Hacks*. 1. Auflage. Beijing : O'Reilly, 2007 (Hacks series). – ISBN 978-0-5965-2799-0
- [Gen07] GENNICK, Jonathan: *SQL: kurz und gut*. 2. Auflage. Köln : O'Reilly, 2007. – ISBN 978-3-89721-522-1
- [Käh90] KÄHLER, Wolf-Michael: *SQL - Bearbeitung relationaler Datenbanken: Eine Anleitung für den Einsatz der Datenbanksprache*. Braunschweig : Vieweg, 1990. – ISBN 3-52804-786-0
- [KKH09] KLINE, Kevin E. ; KLINE, Daniel ; HUNT, Brand: *SQL in a Nutshell*. 3. Auflage. Sebastopol : O'Reilly, 2009. – ISBN 978-0-5965-1884-4
- [KM07] KUHLMANN, Gregor ; MÜLLMERSTADT, Friedrich: *Rororo Computer*. Bd. 61245: *SQL: Der Schlüssel zu relationalen Datenbanken*. 2. Auflage. Reinbek bei Hamburg : Rowohlt Taschenbuch Verlag, 2007. – ISBN 978-3-4996-1245-9
- [MS97] MELTON, Jim ; SIMON, Alan R.: *Understanding the new SQL: A complete guide*. 5. Auflage. San Francisco Calif. : Kaufmann, 1997 (The Morgan Kaufmann series in data management systems). – ISBN 1-55860-245-3

- [Pan00] PANNY, Wolfgang: *Einführung in den Sprachkern von SQL-99: Mit zahlreichen Tabellen*. Berlin : Springer, 2000. – ISBN 3-54065-547-6
- [TC89] TRIMBLE, J. H. ; CHAPPELL, David: *A visual introduction to SQL*. 2. Auflage. New York : Wiley, 1989. – ISBN 0-47161-684-2
- [Tür03] TÜRKER, Can: *SQL:1999 & SQL:2003: Objektrelationales SQL, SQLJ & SQL/XML*. 1. Auflage. Heidelberg : dpunkt-Verlag, 2003. – ISBN 3-89864-219-4

## PostgreSQL

- [Boe03] BOENIGK, Cornelia: *PostgreSQL: Grundlagen - Praxis - Anwendungsentwicklung mit PHP*. 1. Auflage. Heidelberg : dpunkt-Verlag, 2003. – ISBN 3-89864-175-9
- [DBB06] DARIE, Cristian ; BALANESCU, Emilian ; BUCICA, Mihai: *Beginning PHP and PostgreSQL, E-Commerce*. 2. Auflage. Berkeley, Calif. : Apress, 2006 (Books for professionals by professionals). – ISBN 978-1-5905-9648-7
- [DD03] DOUGLAS, Korry ; DOUGLAS, Susan: *PostgreSQL: a comprehensive guide to building, programming, and administering PostgreSQL databases*. Indianapolis, Ind. : Sams Publ., 2003. – ISBN 0-73571-257-3
- [EH09] EISENTRAUT, Peter ; HELMLE, Bernd: *PostgreSQL-Administration*. 1. Auflage. Beijing : O'Reilly, 2009. – ISBN 978-3-8972-1777-5
- [Eis03] EISENTRAUT, Peter: *PostgreSQL: Das offizielle Handbuch*. 1. Auflage. Bonn : mitp-Verlag, 2003 (Datenbanken). – ISBN 978-3-8266-1337-1
- [Eis05] EISENTRAUT, Peter: *PostgreSQL: ge-packt*. 1. Auflage. Bonn : mitp-Verlag, 2005 (Datenbanken). – ISBN 978-3-8266-1493-4
- [GS02] GESCHWINDE, Ewald ; SCHÖNIG, Hans-Jürgen: *Datenbank-Anwendungen mit PostgreSQL: Einführung in die Programmierung mit SQL, Java, C/C++, Perl, PHP und Delphi*. München : Markt+Technik Verlag, 2002. – ISBN 3-82726-394-8
- [Mom01] MOMJIAN, Bruce: *PostgreSQL: Introduction and concepts*. Boston, Mass. : Addison-Wesley, 2001. – ISBN 0-20170-331-9
- [MS05] MATTHEW, Neil ; STONES, Richard: *Beginning databases with PostgreSQL: From novice to professional*. 2. Auflage. Berkeley, Calif. : Apress, 2005 (Books for professionals by professionals). – ISBN 978-1-5905-9478-0
- [OH14] OBE, Regina ; HSU, Leo: *PostgreSQL – Up & running*. 2. Auflage. O'Reilly Media, Sebastopol : O'Reilly, 2014. – ISBN 978-1-449-37319-1
- [Posa] POSTGRESQL, Global Development G.: *PostgreSQL 9.0 Reference Manual - Volume 1a: The SQL Language*. Revised. Network Theory Ltd (The PostgreSQL 9.0 Reference manual). – ISBN 978-1-9069-6604-1
- [Posb] POSTGRESQL, Global Development G.: *PostgreSQL 9.0 Reference Manual - Volume 1b: SQL Command Reference*. Revised. Network Theory Ltd (The PostgreSQL 9.0 Reference manual). – ISBN 978-1-9069-6605-8

- [Posc] POSTGRESQL, Global Development G.: *PostgreSQL 9.0 Reference Manual - Volume 2: Programming Guide*. Revised. Network Theory Ltd (The PostgreSQL 9.0 Reference manual). – ISBN 978-1-9069-6606-5
- [Posd] POSTGRESQL, Global Development G.: *PostgreSQL 9.0 Reference Manual - Volume 3: Server Administration Guide*. Revised. Network Theory Ltd (The PostgreSQL 9.0 Reference manual). – ISBN 978-1-9069-6607-2
- [PW10] PFEIFFER, Thomas ; WENK, Andreas: *PostgreSQL: Installation, Grundlagen, Praxis*. 1. Auflage. Bonn : Galileo Press, 2010. – ISBN 978-3-8362-1346-2
- [Sch09] SCHERBAUM, Andreas: *PostgreSQL. Datenbankpraxis für Anwender, Administratoren und Entwickler*. 1. Auflage. München : Open Source Press, 2009. – ISBN 978-3-937514-69-7
- [WD02] WORSLEY, John C. ; DRAKE, Joshua D.: *Practical PostgreSQL*. 1. Auflage. Beijing : O'Reilly, 2002. – ISBN 1-56592-846-6
- [Wei04] WEINSTABL, Paul: *PostgreSQL: Administration und Einsatz*. Böblingen : Computer- und Literatur-Verlag, 2004 (Computer & Literatur). – ISBN 3-93654-622-3
- [Wie09] WIEKEN, John-Harry: *SQL: Einstieg für Anspruchsvolle*. München : Addison Wesley in Pearson Education Deutschland, 2009 (Master Class). – ISBN 978-3-8273-2485-6

## MySQL

- [Atk02] ATKINSON, Leon: *Core MySQL*. Upper Saddle River, NJ : Prentice Hall PTR, 2002 (PH core series). – ISBN 0-13066-190-2
- [DuB08] DUBOIS, Paul: *MySQL*. 4. Auflage. Indianapolis, Ind. : Sams Publ., 2008. – 1224 S. – ISBN 978-0-6723-2938-8
- [Kof06] KOFER, Michael: *MySQL: Einführung, Programmierung, Referenz*. München : Addison-Wesley, 2006 (Linux specials). – ISBN 3-82731-762-2
- [YRK99] YARGER, Randy J. ; REESE, George ; KING, Tim: *MySQL and mSQL*. 1. Auflage. Sebastopol, CA : O'Reilly, 1999. – ISBN 1-56592-434-7

## PHP, HTML und CSS

- [Kra05] KRAUSE, Jörg: *PHP 5 - Webserverprogrammierung unter Windows und Linux*. München, Wien : Hanser Verlag, 2005. – ISBN 978-3-44640-334-5
- [LBH06] LERDORF, Rasmus ; BERGMANN, Sebastian ; HICKING, Garvin: *PHP: kurz und gut*. 3. Auflage. Köln : O'Reilly, 2006. – ISBN 978-3-89721-520-7
- [Mey05] MEYER, Eric A.: *Cascading Style Sheets - Das umfassende Handbuch*. 1. Auflage. Köln : O'Reilly, 2005. – ISBN 3-89721-386-9
- [MG14] MÜNZ, Stefan ; GULL, Clemens: *HTML5 Handbuch*. München : Franzis Verlag, 2014. – ISBN 978-3-64560-345-4

- [Mül14] MÜLLER, Peter: *Einstieg in CSS - Webseiten gestalten mit HTML und CSS*. 1. Auflage. Bonn : Galileo Press, 2014. – ISBN 978–3–83622–776–6
- [NR14] NIEDERST ROBBINS, Jennifer: *HTML5: kurz und gut*. 5. Auflage. Köln : O'Reilly, 2014. – ISBN 978–3–95561–656–4
- [SW04] SEEBOERGER-WEICHSELBAUM, Michael: *PHP - Webseiten dynamisch programmieren*. 2. Auflage. Hamburg : Rowohlt Taschenbuch Verlag, 2004. – ISBN 3–499–61233–X
- [Ter18] TERIETE, Jan: *Grundlagen der PHP7 Programmierung*. Nürnberg : Webmasters Press, 2018. – ISBN 978–1–98087–779–0
- [The06] THEIS, Thomas: *Einstieg in PHP 5 und MySQL 5*. 4. Auflage. Bonn : Galileo Press, 2006. – ISBN 978–3–89842–854–5
- [WH15] WENZ, Christian ; HAUSER, Tobias: *PHP 5.6 und MySQL*. 1. Auflage. Bonn : Rheinwerk Verlag, 2015. – ISBN 978–3–83623–058–2

## 6.6. Übungszettel A-16 – Querschnittsübung

### Aufgabe 16.1.

- (a) Erläutern Sie die Semantik der drei Armstrong-Axiome  
 $Y \subseteq X \Rightarrow X \rightarrow Y$ ,  
 $X \rightarrow Y \wedge Y \rightarrow Z \Rightarrow X \rightarrow Z$ .  
 $X \rightarrow Y \Rightarrow XZ \rightarrow YZ$ ,
- (b) Leiten Sie unter Angabe der verwendeten Axiome die Gültigkeit der Schlussregeln  
(i)  $X \rightarrow Y \wedge WY \rightarrow Z \Rightarrow XW \rightarrow Z$  (Pseudotransitivität)  
(ii)  $X \rightarrow YZ \Rightarrow X \rightarrow Y \wedge X \rightarrow Z$  (Zerlegung)  
ab.
- (c) Gegeben seien die funktionalen Abhängigkeiten  
(1)  $A \rightarrow B$ , (2)  $C \rightarrow B$ , (3)  $D \rightarrow ABC$ , (4)  $AC \rightarrow D$ . Leiten Sie systematisch unter Angabe der verwendeten Gesetze die Gültigkeit der funktionalen Abhängigkeiten  
(i)  $D \rightarrow ABCD$ , (ii)  $AC \rightarrow BD$ , (iii)  $AC \rightarrow ABCD$  her.

**Aufgabe 16.2.** Gegeben sind die zwei Mengen funktionaler Abhängigkeiten  $\mathcal{F} = \{AC \rightarrow B, A \rightarrow C, D \rightarrow A\}$  und  $\mathcal{G} = \{A \rightarrow BC, D \rightarrow AB\}$ . Zeigen Sie mit Hilfe der Armstrong-Axiome, dass  $\mathcal{F} \equiv \mathcal{G}$  ist.

**Aufgabe 16.3.** Für die Relation  $R$  mit  $\mathcal{A}(R) = ABCDEFG$  gelten die funktionalen Abhängigkeiten  $\mathcal{F} = \{B \rightarrow E, C \rightarrow F, BD \rightarrow G\}$ .

- (a) Berechnen Sie  $ABC^+$ .
- (b) Ist  $ABC \rightarrow D$  ableitbar (d.h. aus  $\mathcal{F}^+$ )?  
(schlüssige Erklärung oder Beispielpopulation)

**Aufgabe 16.4.** Untersuchen Sie für die Menge funktionaler Abhängigkeiten  $\mathcal{F} = \{AB \rightarrow C, D \rightarrow E, AE \rightarrow G, GD \rightarrow H, ID \rightarrow J\}$  für die Relation  $R$  mit  $\mathcal{A}(R) = ABCDEFGHIJ$ , ob

- (a)  $ABD \rightarrow GH \in \mathcal{F}^+$   
(b)  $ABD \rightarrow HJ \in \mathcal{F}^+$

gilt.

**Aufgabe 16.5.** Gegeben sind zwei Relationen  $R$  und  $S$  mit Beispielausprägungen:

$R$				$S$		
$A$	$B$	$C$	$D$	$A$	$B$	$E$
0	a	z	1	1	c	k
0	a	t	4	1	a	n
1	a	u	1	0	a	k
0	b	t	1	0	a	n

Anfragen an diese Datenbasis können als Relationenalgebra-, Tupelkalkül-, Domain-Kalkül-Ausdruck bzw. als Datalog-Programm formuliert werden. Zu

(a)  $\pi_{AC}(R \bowtie S)$

(b)  $\{ \langle c, e \rangle \mid \exists a \exists b \exists d \exists f \exists g (R(a, b, c, d) \wedge S(f, g, e) \wedge a < f \wedge b \neq g) \}$

sollen das Anfrageergebnis und die jeweils 3 nicht angegebenen äquivalenten Formulierungen ermittelt werden.

**Aufgabe 16.6.** Gegeben sind die Relationen  $R$ ,  $S$  und  $T$  mit den Relationsschemata  $R(A,B)$ ,  $S(A,C,D)$  und  $T(C,E)$  und folgenden Extensionen:

$R$		$S$			$T$	
$A$	$B$	$A$	$C$	$D$	$C$	$E$
2	c	3	x	1	v	4
2	b	7	z	1	u	5
3	c	2	u	8	z	4
5	d	3	z	9	w	6
7	b				x	5

(a) Geben Sie die Antwortmengen für folgende Anfragen im Tupelkalkül über  $R$ ,  $S$  und  $T$  an:

- (i)  $\{s \mid S(s) \wedge (s[A] = 3 \vee s[D] = 1)\}$
- (ii)  $\{t \mid \exists r, s (R(r) \wedge S(s) \wedge t[A] = r[A] \wedge t[D] = s[D] \wedge r[A] = s[A])\}$
- (iii)  $\{t \mid T(t) \wedge \exists s (S(s) \wedge t[C] = s[C])\}$
- (iv)  $\{t \mid \exists r, s (R(r) \wedge S(s) \wedge t[B] = r[B] \wedge t[D] = s[D] \wedge r[A] \neq s[A] \wedge r[B] = 'c')\}$

(b) Schreiben Sie die Anfragen als Ausdrücke der relationalen Algebra.



**Aufgabe 16.7.** *Ein IT-Konzern bietet hausinterne Fortbildung an. Die Fortbildungskurse decken Themengebiete ab, die für die Entwicklung des Konzerns wichtig sind. Die Fortbildungsorganisation wird datenbankgestützt organisiert. Die Angestellten können Fortbildungsveranstaltungen bestimmter Dauer zu einem festgelegten Termin besuchen. Für die Angestellten sind neben dem Namen die Personalnummer, die Funktion bzw. der Aufgabenbereich und das Gehalt repräsentiert. Die Fortbildungen sind bestimmten Gebieten eindeutig zugeordnet. Unter den Angestellten gibt es welche, die mit der Leitung eines (maximal eines) Fortbildungsgebiets betraut sind. Sie betreuen zu ihm eine Webseite im Intranet des Konzerns. Es wird erwartet, dass die Fortbildungen erfolgreich verlaufen. Mehrfacher Besuch der gleichen Fortbildung ist nicht erlaubt.*

- (a) *Entwerfen Sie zu dieser Beschreibung ein ER-Modell mit Angabe der Komplexitäten. Formulieren Sie ggf. zusätzliche Annahmen durch begleitenden Text.*
- (b) *Geben Sie eine einschränkende Bedingung an, die im Modell nicht repräsentiert werden kann.*
- (c) *Transformieren Sie das Modell in einen normalisierten relationalen Tabellentwurf mit Kennzeichnung der Schlüssel. Wo liegen Fremdschlüsselbeziehungen vor?*

**Aufgabe 16.8.** *Formulieren Sie die folgenden Anfragen in SQL.*

- (a) *Welche Fortbildungen werden im Gebiet ‘Datenbanken’ angeboten?*
- (b) *Welche Angestellten haben einen Fortbildungskurs aus dem Gebiet ‘Datenbanken’ belegt?*
- (c) *Welche Angestellten haben überhaupt keine Fortbildungskurse belegt?*
- (d) *Die Anzahl der Angestellten, die eine ‘Datenbank’-Fortbildung belegt haben.*
- (e) *Das minimale und maximale Gehalt der Angestellten, aufgeschlüsselt nach Aufgabenbereich.*

**Aufgabe 16.9.** *Bearbeiten Sie von den letzten beiden Übungszetteln die Aufgaben 14.1-14.3 und 15.1-15.3. Auch diese dienen als Vorbereitung für die DB-Klausur.*

## Übungsaufgabe 16.1

- (a) Zur Semantik der drei Armstrong-Axiome:
- (i) Reflexivität: Eine Attributmenge bestimmt (trivialerweise) eine Teilmenge von sich selbst.
  - (ii) Bestimmt eine Attributmenge eine zweite und diese eine dritte, so bestimmt die erste auch die dritte.
  - (iii) Erweiterung: Eine Determinante kann um weitere Attribute angereichert werden, diese werden auch von der Vereinigungsmenge mitbestimmt.
- (b) Leiten Sie unter Angabe der verwendeten Axiome die Gültigkeit der Schlussregeln ab:
- (i)  $X \rightarrow Y \wedge WY \rightarrow Z \Rightarrow XW \rightarrow Z$  (Pseudotransitivität)  
 $X \rightarrow Y \xRightarrow{Erw.} XW \rightarrow WY \xRightarrow{Tr.} XW \rightarrow Z$
  - (ii)  $X \rightarrow YZ \Rightarrow X \rightarrow Y \wedge X \rightarrow Z$  (Zerlegung):  
 $X \rightarrow YZ \xRightarrow{Erw.} XX \rightarrow XYZ \xRightarrow{\cup} X \rightarrow XYZ$ .  
 Und:  $XYZ \rightarrow X \wedge XYZ \rightarrow Y \wedge XYZ \rightarrow Z$  (Reflexivität)  $\xRightarrow{Tr.} X \rightarrow Y \wedge X \rightarrow Z$ .
- (c) Gegeben sind die funktionalen Abhängigkeiten  
 (1)  $A \rightarrow B$ , (2)  $C \rightarrow B$ , (3)  $D \rightarrow ABC$ , (4)  $AC \rightarrow D$ .
- (i) Nach (3) gilt  $D \rightarrow ABC$ . Erweiterung  $\Rightarrow DD \rightarrow ABCD \Rightarrow D \rightarrow ABCD$
  - (ii) (2)  $C \rightarrow B \xRightarrow{Erw.} CD \rightarrow BD$   
 (4)  $AC \rightarrow D \xRightarrow{Ref.} AC \rightarrow CD$   
 $\xRightarrow{Tr.} AC \rightarrow BD$
  - (iii) (4)  $AC \rightarrow D \wedge$  (3)  $D \rightarrow ABC \xRightarrow{Tr.} AC \rightarrow ABC$   
 $\xRightarrow{\cup} AC \rightarrow ABCD$

## Übungsaufgabe 16.2

$$\mathcal{F} = \{AC \rightarrow B, A \rightarrow C, D \rightarrow A\}, \mathcal{G} = \{A \rightarrow B, A \rightarrow C, D \rightarrow A, D \rightarrow B\}.$$

Wir betrachten zuerst  $\mathcal{F}$ :  $A \rightarrow C \xRightarrow{Erw.} AA \rightarrow AC \Rightarrow A \rightarrow AC$ . Mit  $AC \rightarrow B \xRightarrow{Trans.} A \rightarrow B$ . Bleibt noch  $D \rightarrow B \in \mathcal{F}$  herzuleiten. Aber  $D \rightarrow A \wedge A \rightarrow B \xRightarrow{Trans.} D \rightarrow B$ . Damit sind mittels FAen aus  $\mathcal{F}$  alle FAen aus  $\mathcal{G}$  abgeleitet. Umgekehrt geht es analog.

## Übungsaufgabe 16.3

- (a) Gegeben ist  $\mathcal{F} = \{B \rightarrow E, C \rightarrow F, BD \rightarrow G\}$ .

$$ABC^+ = ABCEEF, \text{ denn } B \rightarrow E \xRightarrow{Erw.} ABC \rightarrow ACE.$$

$$C \rightarrow F \xRightarrow{Erw.} ABC \rightarrow ABF \xRightarrow{Ver.} ABC \rightarrow ABCEEF.$$

Alternative: Hüllenalgorithmus

Initial für  $ABC^+$  ist  $H_0 = ABC$ .

Mit  $B \rightarrow E$  ist  $H_1 = ABCE$ .

Mit  $C \rightarrow F$  ist  $H_2 = ABCEEF$ .

Es gibt keine weiteres FAen mehr, die  $H_4$  erweitern können, also ist  $ABC^+ = ABCEEF$

- (b) Es gilt  $ABC \not\rightarrow D$ , d.h.  $ABC \rightarrow D \notin \mathcal{F}^+$ .

Dies lässt sich anhand des Hüllenalgorithmus zeigen, da  $ABC^+ = ABCE$  nicht mehr erweiterbar ist.

Ein Gegenbeispiel für  $ABC \rightarrow D$  wäre eine Ausprägung  $r$ :

$A$	$B$	$C$	$E$	$F$	$D$	$G$
0	0	0	0	0	0	0
0	0	0	0	0	1	1

Hier bestimmt  $ABC \rightarrow D$ , aber in  $r$  gelten alle anderen FAen aus  $\mathcal{F}$ .

## Übungsaufgabe 16.4

Gegeben ist  $\mathcal{F} = \{AB \rightarrow C, D \rightarrow E, AE \rightarrow G, GD \rightarrow H, ID \rightarrow J\}$ ,  $\mathcal{A}(R) = ABCDEFGHIJ$ .

- (a)  $ABD \rightarrow GH \in \mathcal{F}^+$ ?

Berechne nach dem Hüllenalgorithmus  $ABD^+$  bzgl.  $\mathcal{F}$ . Initial ist  $H_0 = ABD$ .

Mit  $AB \rightarrow C$  folgt  $H_1 = ABCD$ .

Mit  $D \rightarrow E$  folgt  $H_2 = ABCDE$ .

Nun wird es möglich,  $AE \rightarrow G$  zu verwenden und es ergibt sich  $H_3 = ABCDEG$ .

Das wiederum gestattet es,  $GD \rightarrow H$  zu verwenden und es folgt  $H_4 = ABCDEGH$ .

Jetzt gibt es keine neuen FAen mit linken Seiten in  $H_4$  mehr, also ist  $ABD^+ = ABCDEGH$  bzgl.  $\mathcal{F}$ .  $GH \subseteq ABCDEGH$ , damit ist  $ABD \rightarrow GH$  aus  $\mathcal{F}$  ableitbar.

- (b)  $ABD \rightarrow HJ \in \mathcal{F}^+$ ?

$HJ \not\subseteq ABCDEGH$ , damit ist  $ABD \not\rightarrow HJ$  (nicht aus  $\mathcal{F}$  ableitbar),  $ABD \rightarrow H$  aber schon.

## Übungsaufgabe 16.5

- (a) Anfrageergebnis:  $\{(0, z), (0, t), (1, u)\}$

- (i) In TRC:

$$\{t \mid \exists r \exists s (R(r) \wedge S(s) \wedge t[A] = r[A] \wedge t[C] = r[C] \wedge r[A] = s[A] \wedge r[B] = s[B])\}$$

- (ii) In DRC:

$$\{ \langle a, c \rangle \mid \exists b \exists d \exists e (R(a, b, c, d) \wedge S(a, b, e)) \}$$

- (iii) In Datalog:

$$\text{anfragea}(A, C) \text{ :- } r(A, B, C, \_), s(A, B, \_)$$

- (b) Anfrageergebnis:  $\{(z, k), (t, k), (t, n)\}$

- (i) In RALG:

$$\pi_{CE}(\sigma_{R.A < S.A \wedge R.B \neq S.B}(R \times S))$$

- (ii) In TRC:

$$\{t \mid \exists r \exists s (R(r) \wedge S(s) \wedge t[C] = r[C] \wedge t[E] = s[E] \wedge r[A] < s[A] \wedge r[B] \neq s[B])\}$$

- (iii) In Datalog:

$$\text{anfrageb}(C, E) \text{ :- } r(A1, B1, C, \_), s(A2, B2, E), A1 < A2, \text{NOT } (B1 = B2)$$

## Übungsaufgabe 16.6

- (a) (i)  $\{s | S(s) \wedge (s[A] = 3 \vee s[D] = 1)\}$ :  
 $\{(3, x, 1), (7, z, 1), (3, z, 9)\}$   
 (ii)  $\{t | \exists r, s (R(r) \wedge S(s) \wedge t[A] = r[A] \wedge t[D] = s[D] \wedge r[A] = s[A])\}$ :  
 $\{(2, 8), (3, 1), (3, 9), ((7, 1))\}$   
 (iii)  $\{t | T(t) \wedge \exists s (S(s) \wedge t[C] = s[C])\}$ :  
 $\{(u, 5), (z, 4), (x, 5)\}$   
 (iv)  $\{t | \exists r, s (R(r) \wedge S(s) \wedge t[B] = r[B] \wedge t[D] = s[D] \wedge r[A] \neq s[A] \wedge r[B] = 'c')\}$ :  
 $\{(c, 1), (c, 9), (c, 8)\}$
- (b) Schreiben Sie die Anfragen als Ausdrücke der relationalen Algebra:

- (i)  $\sigma_{A=3 \vee D=1}(S)$   
 (ii)  $\pi_{AD}(R \bowtie S)$   
 (iii)  $\pi_{CE}(S \bowtie T)$   
 (iv)  $\pi_{R.B, S.D}(\sigma_{R.A \neq S.A \wedge R.B = 'c'}(R \times S))$

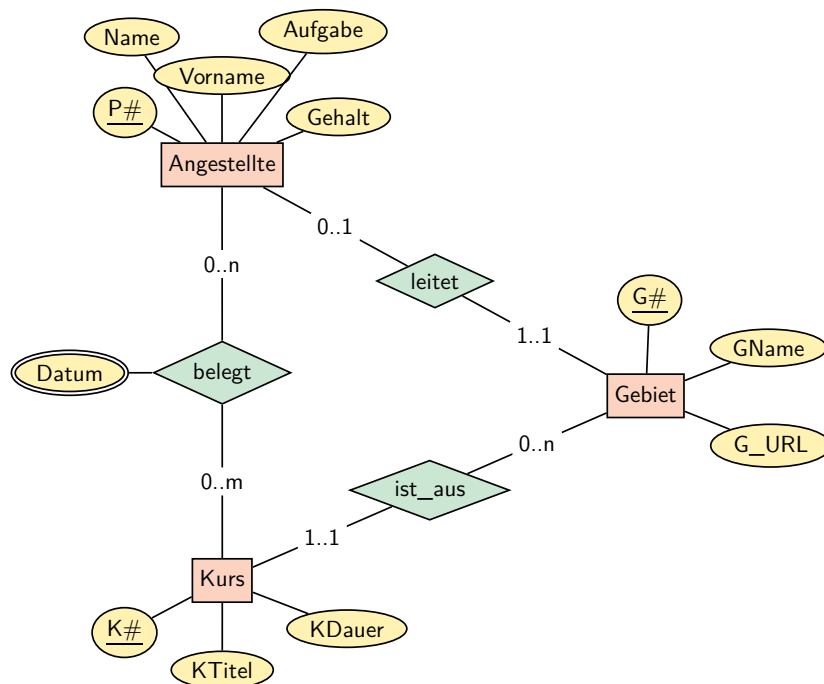
$R \times S$				
$R.A$	$S.A$	$B$	$C$	$D$
2	3	c	x	1
2	7	c	z	1
2	2	c	u	8
2	3	c	z	9
2	3	b	x	1
2	7	b	z	1
2	2	b	u	8
2	3	b	z	9
3	3	c	x	1
3	7	c	z	1
3	2	c	u	8
3	3	c	z	9
5	3	d	x	1
5	3	d	z	1
5	2	d	u	8
5	3	d	z	9
7	3	b	x	1
7	7	b	z	1
7	2	b	u	8
7	3	b	z	9

$R \bowtie S$			
$A$	$B$	$C$	$D$
2	c	u	8
2	b	u	8
3	c	x	1
3	c	z	9
7	b	z	1

$S \bowtie T$			
$A$	$C$	$D$	$E$
3	x	1	5
7	z	1	4
2	u	8	5
3	z	9	4

## Übungsaufgabe 16.7

- (a) Entity-Relationship-Modell:



- (b) Der Ausschluss mehrfacher Belegung desselben Kurses von einem Angestellten kann über Komplexitäten nicht ausgedrückt werden.
- (c) Angestellte (P#, Vorname, Name, Aufgabe, Gehalt)  
 Gebiet (G#, GName, G\_URL, GLeiter) Foreign Key GLeiter Angestellte(P#)  
 Kurs (K#, KTitel, KDauer, Geb) Foreign Key Geb Gebiet(G#)  
 belegt (P#, K#, Datum)

## Übungsaufgabe 16.8

- (a) 

```
SELECT K.K#, K.KTitel, K.KDauer
FROM   Kurs K, Gebiet G
WHERE  K.Geb = G.G# AND G.GName = 'Datenbanken'
```
- (b) 

```
SELECT P#, Name
FROM   Angestellte A, Kurs K, Gebiet G, belegt B
WHERE  K.Geb = G.G# AND
       K.Titel = 'Datenbanken' AND
       B.K# = K.K# AND
       A.P# = B.P#
```
- (c) 

```
SELECT A.P#, ...
FROM   Angestellte A
WHERE  A.P# NOT IN (
    SELECT B.P#
    FROM   belegt B
)
```
- (d) 

```
SELECT COUNT ( DISTINCT B.P# )
```

```
FROM   belegt,  
      ( SELECT K.K#, K.Titel, K.Dauer  
        FROM   Kurs K  
        WHERE  exists  
              ( SELECT *  
                FROM   Gebiet G  
                WHERE  K.Geb = G.G# AND G.Titel = 'Datenbanken' )) AS D  
WHERE  D.P# = B.P#
```

```
(e) SELECT   MIN(A.Gehalt), MAX(A.Gehalt)  
FROM         Angestellte A  
GROUP BY    A.Aufgabe
```

## Übungsaufgabe 16.9

Siehe Lösungen zu A-14 und A-15.

## C. psql – das PostgreSQL-Terminal

Aufruf (SL-Distribution, Benutzer lewein):

```
psql
```

Der Aufruf verbindet den Datenbankbenutzer lewein vom Terminal des Linux-Benutzers lewein aus mit dem unter localhost laufenden Datenbankserver-Dämon postgres (ohne Passworтеingabe). Die für den Datenbankbenutzer lewein eingerichtete Standarddatenbank heißt ebenfalls lewein, sie wird am Prompt angezeigt:



```
xterm lewein@ramanujan: ~  
lewein@ramanujan ~  
$ psql  
psql (8.4.20)  
Geben Sie »help« für Hilfe ein.  
lewein=>
```

### Verbindung zu einem entfernten Datenbankserver

Soll eine Verbindung zu einem entfernten Datenbankserver aufgebaut werden, ist die Option `-h` von Interesse.<sup>108</sup> Für die Lehrerweiterbildung werden als Datenbankserver Minicomputer vom Typ Raspberry Pi mit Linux als Betriebssystem verwendet, die in ein kleines Netz integriert werden. Der Zugang zu diesem Netz erfolgt entweder über eine drahtgebundene LAN-Verbindung oder über den WLAN-Router (ESSID: LWB\_DBS, Passwort: databases), der gleichzeitig via DHCP (Netzsegment 192.168.2.0/24) dynamisch IP-Adressen vergibt. Die individuellen Datenbankserver sind nach Einbuchen in das WLAN unter der IP 192.168.2.X erreichbar. Eine Liste aller verfügbaren Raspberry Pis kann vom UDBP-Server unter der Adresse <http://192.168.2.13/udbp> abgerufen werden.

Der Zugriff auf den fernen Server gelingt mit dem Kommando

```
lewein$ psql -h 192.168.2.X
```

Zur Authentifizierung muss (als Benutzer lewein) das Datenbankserver-Passwort `niewel` eingegeben werden, das bei der Einrichtung durch den Postgresql-Administrator vergeben wurde.

<sup>108</sup>Einen PostgreSQL-Server so einzurichten, dass er entfernten Clienten zur Verfügung steht, ist nicht ganz trivial. Die zentrale Konfigurationsdatei ist `pg_hba.conf`, die unter SL-Linux unterhalb von `/var/lib/pgsql/data` zu finden ist. Weiteres zur Einrichtung eines entfernten Datenbankservers findet man im Kapitel *Administration* von [Eis05].

## Ein- und Ausgabeumlenkung

Um nicht alle Eingaben am `psql`-Terminal über die Tastatur machen zu müssen, kann man eine Textdatei erstellen, die man am `psql`-Prompt mit der Direktive `\i <Dateiname>` einliest. Die Wirkung ist die gleiche, als hätte man den Inhalt der Textdatei via Tastatur eingegeben.

Ähnlich verhält es sich mit der Ausgabeumlenkung. Der `psql`-Client gibt die Antwort des Datenbank-Servers im Terminalfenster aus. Will man die Antworten weiterverarbeiten, bietet sich die `\o`-Direktive an. Gibt man im `psql`-Terminalfenster `\o <Dateiname>` ein, werden alle nachfolgenden Anfrageergebnisse in die angegebene Datei umgelenkt. Ein leeres `\o` löscht die Umlenkung wieder.

## Aufrufoptionen von `psql`

`psql` is the PostgreSQL interactive terminal.

Usage:

`psql [OPTION]... [DBNAME [USERNAME]]`

General options:

<code>-c, --command=COMMAND</code>	run only single command (SQL or internal) and exit
<code>-d, --dbname=DBNAME</code>	database name to connect to (default: "lewein")
<code>-f, --file=FILENAME</code>	execute commands from file, then exit
<code>-l, --list</code>	list available databases, then exit
<code>-v, --set=, --variable=NAME=VALUE</code>	set <code>psql</code> variable <code>NAME</code> to <code>VALUE</code>
<code>-X, --no-psqlrc</code>	do not read startup file ( <code>~/.psqlrc</code> )
<code>-1 ("one"), --single-transaction</code>	execute command file as a single transaction
<code>--help</code>	show this help, then exit
<code>--version</code>	output version information, then exit

Input and output options:

<code>-a, --echo-all</code>	echo all input from script
<code>-e, --echo-queries</code>	echo commands sent to server
<code>-E, --echo-hidden</code>	display queries that internal commands generate
<code>-L, --log-file=FILENAME</code>	send session log to file
<code>-n, --no-readline</code>	disable enhanced command line editing (readline)
<code>-o, --output=FILENAME</code>	send query results to file (or <code> pipe</code> )
<code>-q, --quiet</code>	run quietly (no messages, only query output)
<code>-s, --single-step</code>	single-step mode (confirm each query)
<code>-S, --single-line</code>	single-line mode (end of line terminates SQL command)

Output format options:

<code>-A, --no-align</code>	unaligned table output mode
<code>-F, --field-separator=STRING</code>	set field separator (default: " ")
<code>-H, --html</code>	HTML table output mode
<code>-P, --pset=VAR[=ARG]</code>	set printing option <code>VAR</code> to <code>ARG</code> (see <code>\pset</code> command)
<code>-R, --record-separator=STRING</code>	set record separator (default: newline)
<code>-t, --tuples-only</code>	print rows only
<code>-T, --table-attr=TEXT</code>	set HTML table tag attributes (e.g., width, border)
<code>-x, --expanded</code>	turn on expanded table output

Connection options:



```
-h, --host=HOSTNAME      database server host or socket directory (default: "local socket")
-p, --port=PORT          database server port (default: "5432")
-U, --username=USERNAME  database user name (default: "lewein")
-w, --no-password        never prompt for password
-W, --password           force password prompt (should happen automatically)
```

For more information, type "?" (for internal commands) or "\help" (for SQL commands) from within psql, or consult the psql section in the PostgreSQL documentation.

Report bugs to <pgsql-bugs@postgresql.org>.

## Kommandos der interaktiven psql-Konsole

### General

```
\copyright              show PostgreSQL usage and distribution terms
\g [FILE] or ;          execute query (and send results to file or |pipe)
\h [NAME]               help on syntax of SQL commands, * for all commands
\q                     quit psql
```

### Query Buffer

```
\e [FILE]              edit the query buffer (or file) with external editor
\ef [FUNCNAME]         edit function definition with external editor
\p                     show the contents of the query buffer
\r                     reset (clear) the query buffer
\s [FILE]              display history or save it to file
\w FILE                write query buffer to file
```

### Input/Output

```
\copy ...              perform SQL COPY with data stream to the client host
\echo [STRING]         write string to standard output
\i FILE                execute commands from file
\o [FILE]              send all query results to file or |pipe
\qecho [STRING]        write string to query output stream (see \o)
```

### Informational

```
(options: S = show system objects, + = additional detail)
\d[S+]                 list tables, views, and sequences
\d[S+] NAME            describe table, view, sequence, or index
\da[+] [PATTERN]       list aggregates
\db[+] [PATTERN]        list tablespaces
\dc[S] [PATTERN]        list conversions
\dC [PATTERN]           list casts
\dd[S] [PATTERN]        show comments on objects
\dD[S] [PATTERN]        list domains
\des[+] [PATTERN]       list foreign servers
\deu[+] [PATTERN]       list user mappings
\dew[+] [PATTERN]       list foreign-data wrappers
\df[antw][S+] [PATRN]   list [only agg/normal/trigger/window] functions
\dF[+] [PATTERN]        list text search configurations
\dFd[+] [PATTERN]       list text search dictionaries
\dFp[+] [PATTERN]       list text search parsers
\dFt[+] [PATTERN]       list text search templates
\dg [PATTERN]           list roles (groups)
\di[S+] [PATTERN]       list indexes
\dL                     list large objects, same as \lo_list
\dn[+] [PATTERN]        list schemas
\do[S] [PATTERN]        list operators
\dp [PATTERN]           list table, view, and sequence access privileges
```

<code>\ds[S+] [PATTERN]</code>	list sequences
<code>\dt[S+] [PATTERN]</code>	list tables
<code>\dT[S+] [PATTERN]</code>	list data types
<code>\du [PATTERN]</code>	list roles (users)
<code>\dv[S+] [PATTERN]</code>	list views
<code>\l[+]</code>	list all databases
<code>\z [PATTERN]</code>	same as <code>\dp</code>

#### Formatting

<code>\a</code>	toggle between unaligned and aligned output mode
<code>\C [STRING]</code>	set table title, or unset if none
<code>\f [STRING]</code>	show or set field separator for unaligned query output
<code>\H</code>	toggle HTML output mode (currently off)
<code>\pset NAME [VALUE]</code>	set table output option (NAME := {format border expanded fieldsep footer null numericlocale recordsep tuples_only title tableattr pager})
<code>\t [on off]</code>	show only rows (currently off)
<code>\T [STRING]</code>	set HTML <code>&lt;table&gt;</code> tag attributes, or unset if none
<code>\x [on off]</code>	toggle expanded output (currently off)

#### Connection

<code>\c[connect] [DBNAME - USER - HOST - PORT -]</code>	connect to new database (currently "lewein")
<code>\encoding [ENCODING]</code>	show or set client encoding
<code>\password [USERNAME]</code>	securely change the password for a user

#### Operating System

<code>\cd [DIR]</code>	change the current working directory
<code>\timing [on off]</code>	toggle timing of commands (currently off)
<code>\! [COMMAND]</code>	execute command in shell or start interactive shell

#### Variables

<code>\prompt [TEXT] NAME</code>	prompt user to set internal variable
<code>\set [NAME [VALUE]]</code>	set internal variable, or list all if no parameters
<code>\unset NAME</code>	unset (delete) internal variable

#### Large Objects

<code>\lo_export LOBOID FILE</code>	
<code>\lo_import FILE [COMMENT]</code>	
<code>\lo_list</code>	
<code>\lo_unlink LOBOID</code>	large object operations

## D. Miniwelten

### D.1. Uni-Miniwelt

PROFESSOREN				
PersNr	Name	Rang	Gehalt	Raum
2125	Sokrates	C4	5400	226
2126	Russel	C4	5700	232
2127	Kopernikus	C3	3800	310
2133	Popper	C3	4100	52
2134	Augustinus	C3	4000	309
2136	Curie	C4	6200	36
2137	Kant	C4	5800	7

STUDENTEN		
MatrNr	Name	Semester
24002	Xenokrates	18
25403	Jonas	12
26120	Fichte	10
26830	Aristoxenos	8
27550	Schopenhauer	6
28106	Carnap	3
29120	Theophrastos	2
29555	Feuerbach	2

VORLESUNGEN			
VorlNr	Titel	SWS	gelesenVon
5001	Grundzüge	4	2137
5041	Ethik	4	2125
5043	Erkenntnistheorie	3	2126
5049	Mäeutik	2	2125
4052	Logik	4	2125
5052	Wissenschaftstheorie	3	2126
5216	Bioethik	2	2126
5259	Der Wiener Kreis	2	2133
5022	Glaube und Wissen	2	2134
4630	Die 3 Kritiken	4	2137

VORAUSSETZEN	
Vorgaenger	Nachfolger
5001	5041
5001	5043
5001	5049
5041	5216
5043	5052
5041	5052
5052	5259

HOEREN	
MatrNr	VorlNr
26120	5001
27550	5001
27550	4052
28106	5041
28106	5052
28106	5216
28106	5259
29120	5001
29120	5041
29120	5049
29555	5022
25403	5022
29555	5001

ASSISTENTEN			
PersNr	Name	Fachgebiet	Boss
3002	Platon	Ideenlehre	2125
3003	Aristoteles	Syllogistik	2125
3004	Wittgenstein	Sprachtheorie	2126
3005	Rhetikus	Planetenbewegung	2127
3006	Newton	Keplersche Gesetze	2127
3007	Spinoza	Gott und Natur	2134

PRUEFEN			
MatrNr	VorlNr	PersNr	Note
28106	5001	2126	1
25403	5041	2125	2
27550	4630	2137	2

## D.2. Bibliotheks-Miniwelt

ADRESSE					
AdNr	PLZ	Ort	Bezirk	Strasse	HNr
100	10623	Berlin	Charlottenburg	Fasanenstr.	88
101	14195	Berlin	Dahlem	Arnimallee	3
102	12489	Berlin	Adlershof	Rudower Chaussee	26
103	10117	Berlin	Mitte	Unter den Linden	8
104	10785	Berlin	Tiergarten	Potsdamer Str	33
105	10961	Berlin	Kreuzberg	Blücherplatz	1
106	10178	Berlin	Mitte	Breite Str	30-36
500	13349	Berlin	Mitte	Müllerstr.	120
501	12165	Berlin	Steglitz	Grunewaldstr.	27
502	10319	Berlin	Lichtenberg	Am Tierpark	35A
503	12247	Berlin	Steglitz	Burowstr.	2
504	14513	Teltow	Null	Feldweg	21

LEIHBUECHEREI		
LNr	LName	AdNr
5	TU Zentralbibliothek	100
7	FU Fachbereichsbibliothek Mathematik und Informatik	101
8	HU UB Standort Erwin-Schrödinger-Zentrum	102
9	Staatsbibliothek Unter den Linden	103
10	Staatsbibliothek Potsdamer Straße	104
11	Zentral- und Landesbibliothek AGB	105
12	Zentral- und Landesbibliothek Berliner Stadtbibliothek	106

VERLAG	
VName	Ort
Springer	Berlin
Pearson Education	Upper Saddle River
Pearson Studium	München
Pearson	Boston
Oldenbourg	München
Suhrkamp	Berlin
Hanser	München

ENTLEIHER				
ENr	EName	EVorname	GebDat	AdNr
3	Meyer	Kurt	1966-04-03	500
9	Müller	Hans	1978-11-06	501
14	Kunze	Robert	1984-02-13	502
12	Krause	Maria	1975-12-06	503
11	Schneider	Anna	1956-08-23	504

BUECHER					
ISBN	Titel	VName	Jahr	Seiten	Preis
9783540763932	Taschenbuch der Algorithmen	Springer	2008	448	19,95
9783827372680	Algorithmen und Datenstrukturen	Pearson Studium	2008	575	39,95
0132144980	Database systems: Models and Languages	Pearson	2011	1155	65,95
9783446409446	Taschenbuch Datenbanken	Hanser	2007	582	29,95
9780131354289	Database systems: The complete book	Pearson Education	2009	1203	56,95
9783826616648	Datenbanken: Konzepte und Sprachen	mitp	2008	783	39,95
3486598341	Datenbanksysteme: Eine Einführung	Oldenbourg	2011	792	39,95
3827371368	Grundlagen von Datenbanksystemen	Pearson Studium	2003	1103	59,95

AUTOR			ROLLE			
ANr	AName	AVorname	ANr	ISBN	Typ	RangNr
107	Voecking	Berthold	107	9783540763932	'H'	1
108	Alt	Helmut	108	9783540763932	'H'	2
109	Dietzfelbinger	Martin	109	9783540763932	'H'	3
110	Reischuk	Ruediger	110	9783540763932	'H'	4
111	Scheideler	Christian	111	9783540763932	'H'	5
112	Vollmer	Heribert	112	9783540763932	'H'	6
113	Wagner	Dorothea	113	9783540763932	'H'	7
120	Kudraß	Thomas	120	9783446409446	'H'	1
133	Saake	Gunter	133	9783826616648	'A'	1
134	Sattler	Kai-Uwe	134	9783826616648	'A'	2
135	Heuer	Andreas	135	9783826616648	'A'	3
145	Elmasri	Ramez	145	3827371368	'A'	1
146	Navathe	Shamkant B.	146	3827371368	'A'	2
153	Kemper	Alfons	153	3486598341	'A'	1
154	Eickler	Andre	154	3486598341	'A'	2
156	Garcia-Molina	Hector	156	9780131354289	'A'	1
157	Ullman	Jeffrey D.	157	9780131354289	'A'	2
158	Widom	Jennifer	158	9780131354289	'A'	3
172	Pomberger	Gustav	172	9783827372680	'A'	1
173	Dobler	Heinz	173	9783827372680	'A'	2

BUCHEXEMPLAR				BUCHEXEMPLAR			
ISBN	ExNr	LNr	RegalNr	ISBN	ExNr	LNr	RegalNr
9783540763932	1	7	23A	9783446409446	5	12	A5c
9783540763932	2	7	23A	9780131354289	1	5	18
9783540763932	3	5	17	9780131354289	2	5	18
9783540763932	4	5	17	9780131354289	3	7	24C
9783540763932	5	8	2	9780131354289	4	7	24C
9783827372680	1	7	23D	9780131354289	5	8	2
9783827372680	2	7	23D	9780131354289	6	10	16
9783827372680	3	5	18	9783826616648	1	5	18
9783827372680	4	5	18	9783826616648	2	7	24C
9783827372680	5	9	A5b	9783826616648	3	8	2
9783827372680	6	12	A5b	9783826616648	4	9	2
0132144980	1	5	16	9783826616648	5	12	A5b
0132144980	2	5	16	3486598341	1	5	18
0132144980	3	5	16	3486598341	2	5	18
0132144980	4	7	24B	3486598341	3	7	24C
0132144980	5	7	24B	3486598341	4	7	24C
0132144980	6	8	2	3486598341	5	7	24C
0132144980	7	8	2	3486598341	6	8	2
0132144980	8	10	15	3486598341	7	8	2
0132144980	9	11	33	3486598341	8	12	A5c
9783446409446	1	5	17	3486598341	9	12	A5c
9783446409446	2	7	23C	3827371368	1	7	24B
9783446409446	3	8	2	3827371368	2	7	24B
9783446409446	4	10	15	3827371368	3	7	24B

AUSLEIHE				
ENr	ISBN	ExNr	LNr	Datum
3	9783540763932	2	7	2011-07-23
3	9783827372680	3	5	2011-08-22
3	0132144980	4	7	2011-07-30
3	9783446409446	2	7	2011-08-14
3	9780131354289	5	8	2011-09-13
3	9783826616648	2	7	2011-08-14
3	3486598341	2	5	2011-08-22
3	9780131354289	4	7	2011-07-30
11	9783540763932	3	5	2011-08-03
12	9783827372680	1	7	2011-09-07
9	9780131354289	6	10	2011-07-27
9	0132144980	7	8	2011-09-05
9	9783446409446	3	8	2011-09-05

## F. Unterrichtsbezogenes Datenbankpraktikum

Die im unterrichtsbezogenen Datenbankpraktikum (UDBP) zu erbringende Leistung stellt den Bewertungsteil des zweiten Semesters der Vorlesung zu Datenbanksystemen dar. Er besteht in der schriftlichen Dokumentation der Praktikumsergebnisse incl. für die Datenbank verwendbarer Dateien. Das Praktikum wird in kleinen Gruppen durchgeführt (2-4 Personen), idealerweise zu dritt.

Aufgabenstellung:	27.03.2023
Praktikumsbeginn:	17.04.2023
<b>Abgabe der Ausarbeitung:</b>	<b>19.06.2023</b>
<b>Präsentationen:</b>	ab 26.06.2023

Eine Abgabe nach dem angegebenen Abgabetermin führt zur Bewertung *nicht bestanden*! Es muss zudem eine Zuordnung einzelner Teile der schriftlichen Ausarbeitung zu den Gruppenmitgliedern möglich sein. Bei fehlender Zuordnung wird davon ausgegangen, dass alle Mitglieder zu gleichen Teilen für alle Abschnitte der Ausarbeitung verantwortlich sind.



Damit die Orientierung in fremdem Material später für alle Gruppenteilnehmer schnell und einfach möglich ist, ist es wichtig, dass Sie sich beim Zusammenstellen der Dokumente an eine einheitliche Ordnerstruktur für Ihre Ausarbeitung halten (siehe Seite F–6). Sinnvoll ist die Erstellung eines Git-Archivs.

Wünschenswert wäre, die Ausarbeitungen aller Gruppen zum Zweck der allgemeinen Nutzung zur Verfügung zu stellen.

### F.1. Einordnung

*DBS-Miniwelten* modellieren den Kern eines größeren realen Anwendungssystems, das wegen seiner Komplexität aus informatischer Sicht nicht im Unterricht anzuwenden ist<sup>109</sup>. Die Modellierung der Miniwelt kann als ein Beispiel und Leitlinie bei der Einführung und Erarbeitung von fachlichen Konzepten des Themenkreises „Datenbanksysteme“ im Unterricht dienen.

Die Erfahrung lehrt, dass Informatiklehrer viele schöne Projektideen für den Unterricht aus reinem Zeitmangel liegen lassen. Die Praktikumsaufgabe setzt hier an, indem sie die zeitfressenden Vorbereitungsaufgaben für die Durchführung eines Projekts erledigt. Die Umsetzung in den Unterricht wird durch die Herstellung einer guten und vollständigen Dokumentation erleichtert. Die Arbeitsergebnisse dieses Praktikums können und sollen als Grundlage für zukünftige Unterrichtsprojekte dienen. Eine konkrete Umsetzung in Unterrichtsplanung (schülergerecht aufbereitetes Material, Arbeitsblätter, Tests, ...) gehört nicht mehr zur Aufgabenstellung.

<sup>109</sup> Was nicht heißt, dass einige Kollegen nicht auf den Gedanken kommen, das „große reale System“ aus Anwendersicht im Unterricht zu behandeln. Das ist möglich, kostet viel Zeit, kann erstaunliche Einsichten erzeugen, der Nutzen für die Realisierung tiefergehender informatischer Lernziele ist jedoch begrenzt.

## Modellierung

Die Kunst bei der Erfindung einer geeigneten Miniwelt liegt unter anderem darin,

- Welten zu finden, die dem Erfahrungsschatz der Schülerinnen und Schüler nahestehen oder zumindest mit geringem Aufwand erfasst werden können,
- eine knappe und präzise textuelle Beschreibung der Miniwelt zu entwerfen,
- verständlich zu machen, welcher Detaillierungsgrad (nicht) erreicht werden soll,
- zu berücksichtigen, welche Teile der Miniwelt nur lose oder gar nicht spezifiziert werden,
- eine Modellierbarkeit im ER-Modell mit 5-7 Entitäten zu gewährleisten,
- eine kleine, aber 'artenreiche' Population der Miniwelt zu generieren
- und eine spezielle Sicht auf die Daten zu ermöglichen, die modular mit Go oder PHP realisierbar ist.

### F.2. Teilaufgaben

- (a) Präzise textuelle Beschreibung des zu modellierenden Weltausschnitts bzw. der Anforderungen an die Datenbank ohne Vorgriff auf Entwurfsentscheidungen des Designers
- (b) Erstellung eines ER-Modells mit vollständiger Attributierung und Angabe von Schlüsseln und Komplexitäten
- (c) Angabe von statischen und ggf. dynamischen Integritätsbedingungen, die nicht im ERM erfasst werden
- (d) Transformation des ER-Modells in eine Menge von Relationen (incl. Schlüssel, Fremdschlüssel, abstrakte Wertebereiche für Attribute)
- (e) Angabe der verwendeten Transformationsregeln
- (f) Angabe von funktionalen Abhängigkeiten in den Relationen
- (g) Konkrete Umsetzung des Relationenentwurfs in einen Datenentwurf für die Implementierung in SQL (postgreSQL),
  - (i) Umsetzung des Relationenentwurfs in DDL und DML, die als Dateien in die Datenbank eingelesen werden können
  - (ii) Begründung der Wahl besonderer Attributdatentypen (z. B. NUMERIC, BOOL oder konkrete VARCHAR-Längen)
  - (iii) Umsetzung der Integritätsbedingungen, wo es möglich ist und nicht zu komplex wird
- (h) Entwurf einer Beispielpopulation der Tabellen: realitätsnah, variationsreich, aber klein
- (i) Konstruktion einer Aufgabensequenz von insgesamt ca. 10 einfachen bis zu schwierigen Anfragen für eine Umsetzung in
  - (i) Relationenalgebra,
  - (ii) TRC und DRC (optional),
  - (iii) Datalog (optional)
  - (iv) und SQL.

Eine Reihe von Aufgabenstellungen werden nur in SQL sinnvoll sein (Aggregatfunktionen, Gruppierung etc.), so dass für die anderen Anfragesprachen eine kleinere Auswahl getroffen werden kann.
- (j) Erstellung von Musterlösungen zu diesen Anfragen in DCL der genannten Sprachen
- (k) ggf. auch Alternativlösungen
- (l) Dokumentation der Anfragergebnisse



- (m) Auswahl und Implementierung einer Sicht auf die erstellten Daten in Form einer interaktiven Anwendung (in Go oder webbasiert in PHP), dazu die
  - (i) Begründung und Motivation der Sicht
  - (ii) Beschreibung der interaktiven Anwendungsmöglichkeiten
  - (iii) Beschreibung optionaler Möglichkeiten der Erweiterung
  - (iv) Erstellung einer datenbankgestützten, interaktiven Anwendung in Go oder PHP auf der Basis der erstellten Daten

Es geht *nicht* darum, lediglich die SQL-Anfragen in Go auszuführen. Es geht darum, eine kleine, praktikable Anwendung für den Endbenutzer zu erstellen, der die Datenbank nutzen soll.

### Anforderung an die Miniwelten

Die Konstruktion eigener Miniwelten nach den bereits genannten Kriterien ist ausdrücklich erwünscht, sollte aber bitte vor Beginn der Arbeiten mit der Projektleitung abgesprochen werden. Bei der verbalen Formulierung kann man sich hinsichtlich des Umfangs und der Detailgenauigkeit an den vielen Beispielen der Vorlesung leiten lassen. Es sei noch einmal darauf hingewiesen, dass die Miniwelt der Erfahrungswelt der Schülerinnen und Schüler möglichst nahe stehen sollte.

Da es hier oft zu Missverständnissen kommt: Die Beschreibung der Miniwelt ist kein Vorgriff auf das ER- oder Relationenmodell, sondern aus der Perspektive des Auftraggebers "die Beschreibung seiner Situation (z. B. der Garten-AG für den Schulgarten) für die eine Datenbank erstellt werden soll – also *bevor* ein ER-Modell entworfen wird. Es werden dem Designer alle Informationen gegeben, die in der Datenbank festgehalten werden sollen. Sie können sich dabei auch überlegen, welche Nachfragen ein Designer an die Garten-AG stellen würde, um die Datenbank korrekt zu entwerfen, und diese Information in die Miniwelt einbauen. Hier ist aber noch nicht von "SSchlüsselattributen", ternären Beziehungen, dem Begriff "Entitäten" o. ä. die Rede, wenn es diese im Schulgarten nicht gibt. Ebenso wird z. B. vermutlich nicht für jedes Ding eine ID-Nummer existieren, diese mglw. einzuführen ist Entscheidung des Designers. Beispiele für Miniweltbeschreibungen sind den zahlreichen Übungen der Vorlesung zu Datenbanksystemen zu entnehmen.

### Anmerkungen zur Bearbeitung

Es geht nicht um ein möglichst detailliertes, realitätsnahes Modell, vielmehr soll von zu vielen Details und sehr seltenen Sonderfällen abgesehen werden, um ein möglichst übersichtliches und nachvollziehbares (aber natürlich wiederum nicht *zu* kleines) Modell zu erhalten. Einiger zeitlicher Aufwand geht in die Konstruktion der Aufgabensequenz (Anfragen, Lösungen, Ergebnisse). Angestrebt ist eine graduell ansteigende Schwierigkeit der Aufgaben mit breiter Vielfalt in den Lösungsmethoden. Schön wäre, wenn die meisten Anfragen jeweils einen neuen Aspekt gegenüber den vorherigen Anfragen demonstrierten. Besonders soll die Ausarbeitung möglichst viele Konzepte von SQL verdeutlichen. Hierfür sind Angaben von Alternativlösungen besonders sinnvoll.

Auch der programmiertechnische Teil (GO oder PHP) des UDBP soll explizit nicht jedes mögliche in der Realität vorkommende Detail abbilden bzw. fordern. Es können allerdings

zusätzliche Erweiterungsmöglichkeiten angeregt werden. Dieser Teil liefert für den Unterricht die Möglichkeit, weitere binnendifferenzierte Aufgaben zu stellen. Wichtig ist bei der Konzeption eine möglichst modulare Struktur der Anwendungsschicht zu erreichen, so dass Schülerinnen und Schüler in jedem Fall zu einem fertigen Produkt kommen können. Wichtig im Web-Zusammenhang ist eine saubere Trennung von Design und Struktur (wenn Sie darin Erfahrungen haben, sollten CSS-Dateien verwendet werden) der datenbankunabhängige Zugriff mittels PDO sowie eine übersichtliche und gut dokumentierte Implementierung.

### F.3. Schriftliche Ausarbeitung

Die Ausarbeitung des Berichts sollte folgende Punkte umfassen:<sup>110</sup>

#### Modellierung

- (a) Knappe, präzise textuelle Beschreibung der Miniwelt in einfachen Aussagesätzen. Dokument: <db>-Welt.txt, eine halbe bis eine Seite Text.
- (b) Das Entity-Relationship-Modell in klassischen Notation unter Angabe aller Attribute, der Schlüssel und der Beziehungskomplexitäten. Achten Sie darauf, dass unter den Attributen mindestens ein numerisches ist, so dass Aggregierungsanfragen möglich werden.  
Dokument: Computererstellte Grafik oder übersichtliche Handzeichnung in Form einer Datei <db>-ERModell.<xxx> (mit xxx = png, pdf, eps, o. ä.).
- (c) Verbale Angabe von semantischen Bezügen bzw. Integritätsbedingungen, die sich im Modell nicht wiederfinden. (<db>-SemBed.txt).
- (d) Ein Relationenmodell dazu in dritter Normalform mit (informeller) Angabe von Schlüsseln, Fremdschlüsseln und umsetzbaren Integritätsbedingungen in einer Datei <db>-RelMod.txt.
- (e) Verbale Kurzbeschreibung des Transformationswegs vom ER-Modell der Datenbank zum Relationenmodell unter Angabe der benutzten Transformationsregeln
- (f) Eine Angabe der im Modell vorkommenden funktionalen Abhängigkeiten in einer Datei <db>-FAen.txt. Bei fraglichen Abhängigkeiten geben Sie einen kurzen Hinweis z. B. Pflanzen-ID->Pflanzenname, aber Pflanzenname->Pflanzen-ID? (<db>-Trafo.txt).
- (g) Eine kleine, gut überlegte Sammlung von weltnahen Beispieldaten zu allen Relationen in der Datenbank <db>. Die verwendeten Namen sollen plausibel, realitätsnah und paradigmatisch für eine größere Datenmenge sein. Und sie sollte mit einigen Sonderfällen auch komplexere Anfragen ermöglichen. *Format*: <db>-Daten.txt und optional auch <db>-Daten-<rel>.csv in Tabellenform. Für das csv-Format wird jede Relation jeweils in einer eigenen Datei abgelegt
- (h) Zur Konstruktion der Anfragen: Anordnung nach steigendem Schwierigkeitsgrad. Jede Anfrage soll möglichst eine neue Komplexitätsstufe, Methode oder Implementierungstechnik demonstrieren. Insbesondere sollen auch Anfragen mit Negation, Unteranfragen, Aggregatfunktionen etc. vertreten sein.  
Mit anderen Worten: der Konstruktionsprozess der Anfragen orientiert sich weniger an der Semantik des Miniweltmodells, sondern vielmehr an der Demonstration der jeweiligen Abfragemittel.  
Der Entwicklungs- und Ausleseprozess der Anfragen wird vermutlich eng mit den Implementierungen gekoppelt sein

<sup>110</sup> Alle Textdateien dabei bitte in UTF-8-Kodierung.

*Dokumentationsformat:* <db>-Anfragen.txt als eine zusammenfassende Textdatei mit allen Anfragen in fortlaufender Nummerierung.

## Relationenalgebra mit DES

Im Einzelnen sind gesucht:

- (a) Das Erstellen und Befüllen der Relationen für DES in einer DDL/DML-Datei in simplem SQL (s. F.5).
- (b) Die Implementierung der Anfragen in einer kommentierten ra-Datei <db>-query.ra. Nicht alle Anfragen werden im SQL-fernen Kontext sinnvoll sein (solche mit Aggregatfunktionen z. B.), sie können weggelassen werden. Die Aufgabenstellung steht jeweils als Kommentar über der auswertbaren Anfrage.
- (c) Das Protokoll aller Anfrageergebnisse in einer Datei <db>-query.ans. Das geht z. B. mit einer einfachen Funktion, die alle Anfragen sequentiell auswertet und dabei die Ausgabe mit &> in eine Datei umlenkt.

## Implementierung im Datenbanksystem (PostgreSQL)

Im Einzelnen sind gesucht:

- (a) *Der DDL-Teil:* eine dokumentierte Datei Create-<db>.sql zur Anlage der Tabellenstruktur einer Datenbank (oder eines Schemas) <db> in PostgreSQL.
- (b) Eine dokumentierte Datei Drop-<db>.sql zur Beseitigung der Datenbank <db>.
- (c) *Die Daten:* zwei Dateien Insert-<db>-data.sql und Delete-<db>-data.sql, die die Probedaten mit generischen SQL-Befehlen (INSERT, DELETE) in <db> einfügen bzw. aus ihr löschen.
- (d) *Die Anfragen:* Die Implementierung aller Anfragen (numeriert) als <db>-query.sql. Die Aufgabe steht jeweils als Kommentar über der ausführbaren Anfrage. Oft sind Alternativ-Implementierungen möglich, deren Varianten in dieser Datei ebenfalls angegeben werden können, z. B. mit a- und b-Varianten. Ein begleitender didaktisch-methodischer Kommentar zu den Anfragen und den Lösungsvarianten ist in der zusammenfassenden Ausarbeitung erwünscht, z. B. wenn ein neuer Befehl verwendet wird.
- (e) Die Protokollierung der Anfrageergebnisse in den Dateien <db>-query<nn>.ans, z. B. mittels `psql -o` oder mittels des Schalters `o <datei>`.

## Didaktisch-Methodisches

Würdigen Sie unter den Aspekten Schwierigkeitsgrad, Umfang, Zeitbedarf und Bedeutsamkeit im Kontext des Informatikunterrichts ihre bis hierher geleistete Arbeit. Welche Teile sind unverzichtbar, welche ggf. fakultativ für ein Schülerprojekt? Haben Sie Ideen zur methodischen Umsetzung? Was wären in einer Weiterführung Ihres Projektes sinnvolle Fragestellungen? Die Antworten gehen in eine Datei <db>-did.txt

## Ausarbeitung

Erwünscht ist eine zusammenfassende Darstellung Ihrer Praktikumsausarbeitung als PDF-Dokument. Die textliche Ausarbeitung steht in einer Druckfassung `<db>.pdf` und ggf. weiteren Formaten wie `<db>.txt` (formatiertes ASCII, UTF8),  $\text{\LaTeX}$ -Format `<db>.tex`, `<db>.odt` oder `<db>.docx`. In jedem Fall wird eine kurze Beschreibung des Projektes zur Orientierung eines unbeteiligten Lesers geliefert (z. B. "Dies ist ein Datenbankprojekt der Lehrerweiterbildung Informatik, das ..."). Es handelt sich um eine kurze Einleitung im Format `README.txt`.

Alle Texte, Aufgaben und Lösungen sollen in einer *logischen Struktur* bereitgestellt werden (s. u.), die es leicht macht, für Unterrichtszwecke geeignete Teile zu übernehmen. Stellen Sie alle Praktikumsergebnisse in einer Archiv-Datei `<db>.tgz` mit folgender Binnenstruktur zusammen:

```
<db>
|-- doc                                (A) Bündelung aller Ausarbeitungen in einer Datei
|   |-- README.txt                    ASCII-Text mit Kurzbeschreibung des DB-Projektes
|   |-- <db>.tex                      optional :-)
|   |-- <db>.odt                      optional
|   |-- <db>.doc[x]                  optional
|   '-- <db>.pdf                     in jedem Fall
|
|-- modell                             (B) Modelldokumente
|   |-- <db>-Welt.txt                Miniwelt
|   |-- <db>-ERModell.png             (oder .pdf, .eps, .jpg, .svg)
|   |-- <db>-SemBed.txt               zusätzliche sematische bzw. Integrationsbedingungen
|   |-- <db>-FAen.txt                vorhandene Funktionale Abhängigkeiten
|   |-- <db>-RelMod.txt              Relationenmodell
|   |-- <db>-Trafo.txt               angewandte Transformationsregeln
|   |-- <db>-Daten.txt               übersichtliche Tabelle der Beispielpopulation (kein SQL)
|   '-- <db>-Daten-<rel>.csv         Daten der jeweiligen Relation (ggf.)
|
|-- relational                         (C) RALG/TRC/DRC/DL (in DES-kompatiblen Format)
|   |-- <db>-DDL.ddl                 Erstellen und befüllen der Relationen in simplem SQL
|   |-- <db>-query.pdf               Alle RALG-Anfragen in originärer Syntax (Pi, Sigma usw.)
|   |-- <db>-query.ra                Alle RALG-Anfragen nummeriert mit Kommentar
|   |-- <db>-query.trc               Alle Tupelkalkül-Anfragen mit Kommentar
|   |-- <db>-query.drc               Alle Domänenkalkül-Anfragen mit Kommentar
|   |-- <db>-query.dl                Alle Datalog-Anfragen mit Kommentar
|   |-- <db>-query-ra.ans            Alle RALG-Anfrageergebnisse
|   |-- <db>-query-trc.ans           Alle Tupelkalkül-Anfrageergebnisse
|   |-- <db>-query-drc.ans           Alle Domänenkalkül-Anfrageergebnisse
|   '-- <db>-query-dl.ans           Alle Datalog-Anfrageergebnisse
|
|-- sql                               (D) SQL (PostgreSQL)
|   |-- Create-<db>.sql              Erstellen der Relationen
|   |-- Drop-<db>.sql                Löschen der Relationen
|   |-- Insert-<db>-data.sql         Befüllen der Relationen mit der Beispielpopulation
|   |-- Delete-<db>-data.sql         Löschen aller Inhalte der Relationen
|   |-- <db>-query.sql              Alle Anfragen nummeriert mit Kommentar
|   |-- <db>-query.ans              Anfrageergebnisse
|
|-- did                               (E) Didaktisch-Methodisches
|   |-- <db>-did.txt                 Didaktisch-methodische Hinweise
|   '-- ...                          Ggf. Weiteres Material (Arbeitsblätter, ...)
|
'-- view                             (F) Implementierung eines Views in Go/PHP (Auswahl)
    |-- <db>-view.txt                Motivation, Anwendung und mgl. Erweiterungen des Views
    |-- <db>-htdocs                  Webserver-Verzeichnisinhalt (nur PHP)
    |   |-- index.html/php           Startdatei im Webserververzeichnis
    |   |-- <db>.css                  CSS-Datei (ggf.)
    |   '-- ...
    '-- <db>-go                      Go-Dateien (nur Go)
        |-- Pakete                    Verwendete externe Go-Pakete (ggf.)
        |   |-- <pak1>
        |   '-- ...
        '-- <db>.go                    Hauptprogramm-Datei
```

## **Präsentation**

Bei der Präsentation wird eine kurze Einführung in die modellierte Miniwelt erwartet, wobei Entitäten und Beziehungen mit ihren Komplexitäten anhand eines ER-Diagramms vorzustellen sind. Die Relevanz des Weltausschnittes für SuS muss deutlich werden. Anschließend ist eine kleine Auswahl an Anfragen an das Relationenalgebra- bzw. SQL-System vorzuführen sowie der programmiertechnische Teil vorzustellen. Dabei soll insbesondere die Realitätsnähe des gewählten Views erläutert werden. Zentrale Teile des Codes sollen vorgestellt werden. Die Präsentation soll eine Dauer von 30 Minuten nicht überschreiten.

## F.4. Bewertung des UDBP

Kandidat A: \_\_\_\_\_

Kandidat C: \_\_\_\_\_

Kandidat B: \_\_\_\_\_

Kandidat D: \_\_\_\_\_

### Ermittlung des Gesamtergebnisses

Jeder Kandidat muss erkennbar an **mindestens zwei Bewertungskriterien der schriftlichen Ausarbeitung** (Qualität der Miniwelt, Modellierung, Entwurf und Population, Güte der Anfragen (SQL/RA) und Anwendungsprogramm) mitgewirkt haben. Für diese Kriterien erfolgt eine individuelle Bewertung. Zusammen mit der Bewertung der allgemeinen Aspekte ergeben sich daraus die individuellen Punktskizzen und daraus die Anteile für die schriftliche Ausarbeitung. Bei fehlender Zuordnung wird davon ausgegangen, dass alle Mitglieder zu gleichen Teilen für alle Abschnitte der Ausarbeitung verantwortlich sind. Die Zuordnung des Gesamtanteils  $a$  zur Note erfolgt nach dem in der gymnasialen Oberstufe üblichen Schlüssel. Bei  $a < 45\%$  gilt das Datenbankpraktikum als nicht bestanden.

### Bedeutung und Bewertung der Merkmalsausprägungen

Symbol	textliche Beschreibung	Punkte
++	sehr ausgeprägt	5
+	ausgeprägt	4
+/o	im Allgemeinen ausgeprägt	3

Symbol	textliche Beschreibung	Punkte
o/–	teilweise ausgeprägt	2
–	kaum ausgeprägt	1
– –	nicht vorhanden	0

### Individuelle Gesamtergebnisse

	A	B	C	D
erreichter Anteil $a$ für die schriftliche Ausarbeitung				
erreichte Note				

## Bewertungsraster

<b>schriftliche Ausarbeitung</b>		++	+	+/-	o/-	-	--
<b>Qualität Miniwelt und ERM</b> (Relevanz, Beschreibung, angemessene Komplexität, Umsetzung in das ER-Modell)	A						
	B						
	C						
	D						
<b>Modellierung</b> (Transformation ins RA-Modell, Begründung von Entwurfsentscheidungen, Funktionale Abhängigkeiten)	A						
	B						
	C						
	D						
<b>Entwurf und Population</b> (DDL-Qualität, Berücksichtigung von Integritätsbedingungen, Güte der Population)	A						
	B						
	C						
	D						
<b>Güte der Anfragen (SQL/RA)</b> (Progression, Vielfalt und Motivation, Alternativen, Korrektheit und Vollständigkeit, Ausnutzen von SQL-Spezifika)	A						
	B						
	C						
	D						
<b>Anwendungsprogramm</b> (Lauffähigkeit, Fehlertoleranz, Übersichtlichkeit und Wiederverwendbarkeit des Codes, Benutzerfreundlichkeit, Modularisierung für Unterrichtszwecke)	A						
	B						
	C						
	D						
<b>Allgemeine Aspekte</b> (Vollständigkeit und Übersichtlichkeit der Dokumentation, Einhalten der vorgegebenen Binnenstruktur, Erleichterung der Fremdnutzung des Projektes)	A						
	B						
	C						
	D						



## F.5. Praktische Hinweise zum UDBP

Während der Arbeit im unterrichtsbezogenen Praktikum fallen viele Aufgaben an, die praktischer Natur sind und in der laufenden Vorlesung nicht oder nur peripher angesprochen wurden. In diesem Kapitel werden einige solcher Probleme kurz angesprochen, um den Start in die selbständige Arbeit zu erleichtern.

### Hinweise zur Verwendung von DES

**Datentypen** Will man mit DES die Daten des postgresQL-Datenbankservers verwenden, dürfen nur folgende Datentypen verwendet werden:

- VARCHAR (n)
- INTEGER
- VARCHAR
- REAL

Insbesondere sind Zeit- und Datums-Datentypen sowie Boolean-Datentypen **nicht** erlaubt. Um dieses Manko zu umgehen, könnte man für Boolean-Typen einen Integer-Typ verwenden, den man auf die Werte 0 und 1 beschränkt und Zeit- und Datumsangaben in Zeichenketten-Typen abspeichern. Eine syntaktische Prüfung auf solcherlei Datentypen zu realisieren wäre allerdings ein Overkill.

**Integration in Geany** Um ähnlich wie für postgresQL auch ra-Dateien für DES in Geany durch einen Druck auf F5 in DES verarbeiten zu lassen und die Ausgabe gleich im Terminalfenster zu bekommen, muss eine Datei `des.sh` irgendwo im Pfad (z. B. in `/home/lewein/bin`) untergebracht werden, die folgenden Inhalt hat:<sup>111</sup>

```
1  #!/bin/bash
2
3  # HERE-Dokument - erzeugt die passende des.ini
4  cat > des.ini <<DES
5  /RA
6  /p $1
7  /q
8  DES
9
10 # startet des, das die des.ini einliest und die Ausgabe
11 # auf dem Terminal darstellt.
12 des
13 # automatisch erzeugte des.ini löschen
14 rm des.ini
```

Zusätzlich muss über das Menu in Geany das Kommando zu Erstellen konfiguriert werden. Unter Ausführen trägt man bei geöffneter ra-Datei ein

```
des.sh %f
```

Nun wird bei Betätigen der Taste F5 eine temporäre ini-Datei erzeugt, die DES beim Start einliest und die aktuelle ra-Datei verarbeitet. Die Ausgabe von DES erscheint dann im sich öffnenden Terminalfenster. Soll die Ausgabe – etwa zur weiteren Verwendung in der UDBP-Doku – noch zusätzlich in einer LOG-Datei protokolliert werden, muss die ra-Datei mit

<sup>111</sup>Die Datei findet man auch auf der Freigabe **udbp** im Ordner *Geany*.

```
/log <Dateiname>
```

beginnen und mit

```
/nolog
```

enden.

**ODBC-Anbindung** Für den Zugriff von DES auf bestehende Datenbanken – z. B. auf welche, die der PostgreSQL-Server beheimatet – muss ggf. noch die ODBC-Unterstützung (*open database connection*) installiert werden. Dafür sind die Pakete

- postgresql-odbc und
- libiodbc

auszuwählen und zu installieren. Anschließend editiert man (als root(!)) im Ordner /etc die Datei odbcinst.ini – falls sie noch nicht existieren – die Zeilen

```
1  [PostgreSQL]
2  Description = ODBC for PostgreSQL
3  Driver      = /usr/lib/psqlodbcw.so
4  Setup       = /usr/lib/libodbcpsqlS.so
5  Driver64    = /usr/lib64/psqlodbcw.so
6  Setup64     = /usr/lib64/libodbcpsqlS.so
7  FileUsage   = 1
8
9  [MySQL]
10 Description = ODBC for MySQL
11 Driver      = /usr/lib/libmyodbc5.so
12 Setup       = /usr/lib/libodbcmyS.so
13 Driver64    = /usr/lib64/libmyodbc5.so
14 Setup64     = /usr/lib64/libodbcmyS.so
15 FileUsage   = 1
```

ein. Danach erstellt man als root die Datei /etc/odbc.ini und fügt dort die Zeilen

```
1  [LWB]
2  Description = Tabellen der Lehrerweiterbildung
3  Driver      = PostgreSQL
4  Database    = lewein
5  Servername  = 127.0.0.1
6  User        = lewein
7
8  [PI]
9  Description = Datenbank von lewein auf fernem Pi
10 Driver      = PostgreSQL
11 Database    = pi
12 Servername  = 192.168.2.13
13 User        = lewein
14 Password    = niewel
```

ein. Nun können in DES mittels des Kommandos

```
/open_db 'LWB'
```

z. B. alle Tabellen der Datenbank lewein auf dem lokalen Rechner verfügbar gemacht werden.