DIPLOMARBEIT

VERGLEICH VON OPEN-SOURCE-DATENBANKSYSTEMEN: SAP DB, PostgreSQL und MySQL

ausgeführt am Institut für

Informationssysteme
der Technischen Universität Wien

unter der Anleitung von

o.Univ.-Prof. Dipl.-Ing. Dr.techn. Georg Gottlob

durch

Viktor Krammer
Kaiserstraße 28/2/7
1070 Wien

Wien, im Juni 2003

_____

Kurzfassung

Die vorliegende Arbeit beschäftigt sich mit dem Vergleich von drei ausgewählten Open-Source-Datenbanksystemen. Das sind PostgreSQL und MySQL, die in dieser Kategorie zu den bekanntesten und wichtigsten freien relationalen Datenbanksystemen zählen, sowie SAP DB, das erst seit kurzem unter die GPL gestellt wurde, und ein Nachfolger der kommerziellen Datenbank ADABAS D ist. Alle drei untersuchten Datenbanksysteme bieten eine SQL/92-kompatible Schnittstelle an, unterscheiden sich aber in puncto Funktionalität, Wartbarkeit und Geschwindigkeit voneinander.

Das Hauptziel dieser Diplomarbeit ist für den Vergleich geeignete Kriterien auszuwählen, die sich aber nicht nur auf Open-Source-Datenbanken beschränken, sondern auch für einen Vergleich mit kommerziellen Datenbanken herangezogen werden können. Zu den wichtigsten Kriterien zählen die interne Speicherorganisation, die Abfrage- und Transaktionsverarbeitung, die Möglichkeiten der Datensicherung und Replikation, der SQL-Funktionsumfang inklusive der vordefinierten Datentypen, das Vorhandensein von Stored Procedures, Triggers und Views, die Rechteverwaltung, die angebotenen Programmierschnittstellen und die Performance. Die größten Unterschiede liegen in der verwendeten Synchronisationskontrolle, dem SQL-Funktionsumfang und den Möglichkeiten, die Datenbank zu sichern und wiederherzustellen. Letzteres stellt sich als das wesentliche Manko im Vergleich zu kommerziellen Datenbanksystemen heraus. In puncto Verfügbarkeit und Skalierbarkeit herrscht bei allen drei untersuchten Datenbanksystemen noch Nachholbedarf.

Ein weiteres Ziel dieser Arbeit ist es, einen Überblick über die theoretischen Konzepte von Datenbank-Management-Systemen zu geben. Dabei wird jedes Kriterium hinsichtlich seiner Bedeutung erläutert. Darüber hinaus wird auch generell auf das relationale Datenbankmodell eingegangen und eine Einführung in die Datenbanksprache SQL gegeben. Viele praktische Beispiele im Laufe der Diplomarbeit sollen das Verhalten und die Bedienung der untersuchten Datenbanksysteme demonstrieren. Die im Anhang abgedruckte Beispielsammlung zeigt, wie man aus unterschiedlichen Programmiersprachen auf die drei Open-Source-Datenbanken zugreifen kann.

Vorwort

Diese Diplomarbeit wurde im Rahmen einer Auftragsarbeit der Firma Siemens AG Österreich, Abteilung MCS TS2, erstellt. Sie knüpft damit an eine vor etwa zehn Jahren verfasste Arbeit bei Siemens Nixdorf zum Thema „Vergleich von relationalen Datenbanksystemen“ [17] an, die damals vier kommerzielle Systeme untersuchte. Im letzten Jahrzehnt hat sich auf dem Sektor von relationalen Datenbanksystemen einiges getan. Das Aufkommen von konkurrenzfähiger Open-Source-Software wird sicherlich den Datenbankmarkt, der derzeit von drei großen Anbietern (Oracle, IBM und Microsoft) beherrscht wird, nachhaltig beeinflussen. Dieses Quasi-Oligopol führte zu einer Hochpreispolitik, die Open-Source-Datenbanken gerade auch für Unternehmen interessant macht. Diese Arbeit gibt einen ausführlichen Überblick über drei ausgewählte Open-Source-Datenbanksysteme: SAP DB, PostgreSQL und MySQL.

An dieser Stelle möchte ich der Firma Siemens AG und hier im Besonderen Herrn Dipl.-Ing. Roland Strasser für das Zustandekommen dieses Projektes danken. Aber auch meinem Betreuer bei Siemens Herrn Dr. Reinhard Pichler bin ich für die konstruktive Kritik und seine großzügige Hilfestellung in allen Belangen zu aufrichtigem Dank verpflichtet.

Ferner möchte ich Herrn Univ.-Prof. Dr. Georg Gottlob vom Institut für Informationssysteme, Abteilung für Datenbanken und Artificial Intelligence der Technischen Universität Wien, der mir diese Arbeit ermöglicht hat, meinen Dank aussprechen.

Mein Dank gilt weiters meinem Bruder für die wichtigen Hinweise in Bezug auf das wissenschaftliche Arbeiten und das Korrekturlesen der Arbeit. Meiner Freundin danke ich für die wertvollen Layout-Tipps, ihren Einsatz als Lektor und für ihre moralische Unterstützung.

Besonders herzlich danke ich meinen Eltern, die mir überhaupt erst das Informatikstudium an der TU Wien ermöglicht haben.

Wien, im Juni 2003 *V. Krammer*

Inhaltsverzeichnis

KURZFASSUNG 2

VORWORT 3

INHALTSVERZEICHNIS 4

ABBILDUNGS- UND TABELLENVERZEICHNIS 9

EINLEITUNG 10

KAPITEL 1 DAS RELATIONALE DATENBANKMODELL 13

1.1 Relationen 14

1.2 Datenbanken 15

1.3 Datenabhängigkeiten 15

1.3.1 Intrarelationale Abhängigkeiten 16

1.3.2 Interrelationale Abhängigkeiten 17

1.4 Konsistenz 17

1.5 Integritätsbedingungen 18

1.6 Operationen im relationalen Datenbankmodell 19

1.6.1 Relationenalgebra 19

1.6.2 Relationenkalküle 21

KAPITEL 2 SYSTEMARCHITEKTUR 23

2.1 Grundlagen 23

2.2 SAP DB 25

2.2.1 Architektur 26

2.2.2 Installation 27

2.2.3 Starten des Servers 28

2.2.4 Einrichten einer neuen Datenbankinstanz 28

2.3 PostgreSQL 30

2.3.1 Architektur 31

2.3.2 Installation 31

2.3.3 Starten des Servers ... 32
2.3.4 Einrichten einer neuen Datenbank ... 32

2.4 MySQL ... 33
2.4.1 Architektur ... 34
2.4.2 Installation ... 35
2.4.3 Starten des Servers ... 36
2.4.4 Einrichten einer neuen Datenbank ... 36

KAPITEL 3 INTERNE SPEICHERORGANISATION ... 37

3.1 Datentypen ... 37
3.1.1 SAP DB ... 39
3.1.2 PostgreSQL ... 39
3.1.3 MySQL ... 40

3.2 Datensätze und Blöcke ... 41

3.3 Indexstrukturen ... 42
3.3.1 Spezielle Indexstrukturen ... 43
3.3.2 Mögliche Anwendungen von Indexstrukturen ... 47
3.3.3 Vergleich der OSDB ... 48

KAPITEL 4 ABFRAGEVERARBEITUNG ... 51

4.1 Die Abfragesprache SQL ... 51
4.1.1 Datendefinition ... 52
4.1.2 Datenmanipulation ... 54
4.1.3 SELECT-Abfragen ... 55

4.2 Abfrageverarbeitung ... 59

4.3 Optimierungsansätze ... 60
4.3.1 Suchstrategien ... 60
4.3.2 Verbundstrategien ... 62
4.3.3 Sonstige Optimierungen ... 63

KAPITEL 5 TRANSAKTIONSVERARBEITUNG ... 64

5.1 Transaktionskonzept ... 65

5.2 Probleme ohne Synchronisation ... 66
5.2.1 Verlorengegangene Änderungen (Lost Update) ... 66
5.2.2 Zugriff auf nicht freigegebene Daten (Dirty Read/Write) ... 66
5.2.3 Nicht wiederholbares Lesen (Non-repeatable Read) ... 67
5.2.4 Phantom-Phänomen ... 67

5.3 Prinzipien der Transaktionsverarbeitung ... 68
5.3.1 ACID-Prinzip ... 68
5.3.2 Serialisierbarkeit ... 69
5.3.3 Striktheit ... 71
5.3.4 Konsistenzgrade ... 71

5.4 Synchronisationskontrolle in den OSDB ... 72
5.4.1 SAP DB ... 74
5.4.2 PostgreSQL ... 75
5.4.3 MySQL ... 77

KAPITEL 6 RECOVERY UND REPLIKATION ... 79

6.1 Fehlerklassen ... 79

6.2 Recovery ... 80
6.2.1 Recovery von Transaktionsfehlern ... 80
6.2.2 Recovery von Systemfehlern ... 81
6.2.3 Recovery von Medienfehlern ... 82

6.3 Replikation ... 86
6.3.1 SAP DB ... 87
6.3.2 PostgreSQL ... 88
6.3.3 MySQL ... 89

KAPITEL 7 KRITERIENKATALOG ... 91

7.1 Unterstützte Plattformen ... 91

7.2 Transaktionsverarbeitung ... 92

7.3 Backup und Restore ... 92

7.4 Recovery ... 92

7.5 Replikation ... 92

7.6 Optimierung 93

7.7 Performance 93

7.8 Rechteverwaltung 98
7.8.1 SAP DB 99
7.8.2 PostgreSQL 100
7.8.3 MySQL 101

7.9 Datentypen 102

7.10 Funktionen und Operatoren 102
7.10.1 Numerische Funktionen und Operatoren 103
7.10.2 Stringfunktionen 104
7.10.3 Datums- und Zeitfunktionen 106
7.10.4 Datentypkonvertierung 108
7.10.5 Sonstige Funktionen 108

7.11 Aggregatfunktionen 110

7.12 Sequenzen 110
7.12.1 SAP DB 111
7.12.2 PostgreSQL 111
7.12.3 MySQL 112

7.13 Constraints 112

7.14 Sub-SELECTs 114

7.15 Stored Procedures 115
7.15.1 SAP DB 115
7.15.2 PostgreSQL 117

7.16 Trigger 118
7.16.1 SAB DB 119
7.16.2 PostgreSQL 119

7.17 Ausnahmebehandlung (Exceptions) 120

7.18 Cursor-Unterstützung 120

7.19 Rekursive SQL-Anweisungen 121

7.20 Views ... 122

7.21 Rules ... 123

7.22 Modifikation des Datenbankschemas ... 124

7.23 Temporäre Tabellen ... 125

7.24 Objekt-relationaler Ansatz ... 126

7.25 Anfallende Wartungsarbeiten ... 127

7.26 Verfügbare Client-Tools ... 127

7.27 Programmierschnittstellen ... 128

7.28 SQL/92-Konformität ... 129

KAPITEL 8 ZUSAMMENFASSUNG ... 130

8.1 Tabellarischer Vergleich ... 130

8.2 Vergleich mit Oracle ... 138

8.3 Ausblick ... 139

ANHANG A PROGRAMMIERSCHNITTSTELLEN ... 140

A.1 ODBC ... 140

A.2 JDBC ... 143

A.3 API ... 145

A.4 Embedded SQL ... 148

A.5 PHP ... 150

A.6 Perl ... 152

A.7 Python ... 153

ANHANG B ABKÜRZUNGEN ... 155

ANHANG C LITERATURVERZEICHNIS ... 157

Abbildungs- und Tabellenverzeichnis

Abbildung 1.1 Beispiel einer relationalen Datenbank [4] 13
Abbildung 2.1 Architektur von SAP DB [1] 26
Abbildung 2.2 Architektur von PostgreSQL [2] 31
Abbildung 2.3 Architektur von MySQL [3] 34
Abbildung 3.1 Schematische Darstellung der Funktionsweise von Indexstrukturen [6] 42
Abbildung 3.2 B-Baum der Höhe 2 (Nicht-Schlüsselanteil nicht dargestellt) [4] 43
Abbildung 3.3 Dynamische Hashtabelle, Blockgröße=2, Datensätze={a,b,c,d}, 2 Bits von *h* verwendet [6] 45
Abbildung 3.4 R-Baum (zweidimensional) [6] 46
Abbildung 3.5 Regionen des R-Baumes aus Abbildung 3.4 [6] 46
Abbildung 4.1 Ablauf einer Abfrageverarbeitung [4] 59
Abbildung 5.1 Lost-Update-Anomalie 66
Abbildung 5.2 Dirty-Read- und Dirty-Write-Anomalie 67
Abbildung 5.3 Non-repeatable-Read-Anomalie 67
Abbildung 5.4 Phantom-Phänomen 67
Abbildung 5.5 Transaktionsverarbeitung [4] 69
Abbildung 5.6 Kompatiblitätsmatrix von Lese- und Schreibsperren [19] 72
Abbildung 5.7 Growing- und Shrinking-Phase eines strikten Zwei-Phasen-Schedulers [19] 73
Abbildung 5.8 Deadlock auf Grund gegenseitiger Abhängigkeit 73
Abbildung 6.1 Organisation eines Data-Managers [3] 80
Abbildung 6.2 Szenario eines Systemfehlers [3] 81
Abbildung 6.3 High Availability durch Spiegeln auf Dateisystemebene 87
Abbildung 6.4 Architektur von Postgres-R [10] 88

Tabelle 3.1 SQL/92 Datentypen [12] 38
Tabelle 3.2 Zusätzliche Datentypen in PostgreSQL [2] 39
Tabelle 3.3 Zusätzliche Datentypen in MySQL [3] 40
Tabelle 5.1 Isolationsgrade des SQL-Standards [12] 71
Tabelle 7.1 Benchmark-Testkonfiguration 94
Tabelle 7.2 Testergebnis: Datenbankerzeugung 94
Tabelle 7.3 Testergebnis: Single-User-Test 95
Tabelle 7.4 Testergebnis: Mulit-User-Test 96
Tabelle 7.5 Numerische Funktionen und Operatoren 103
Tabelle 7.6 Stringfunktionen 104
Tabelle 7.7 Datums- und Zeitfunktionen 106
Tabelle 7.8 Datentypkonvertierung 108
Tabelle 7.9 Sonstige Funktionen 108
Tabelle 7.10 Aggregatfunktionen 110
Tabelle 7.11 Vergleich von ALTER TABLE 124
Tabelle 8.1 Gegenüberstellung der wichtigsten Merkmale 130

Einleitung

Die Open-Source-Software, die sich früher abseits der kommerziellen Softwareentwicklung vorwiegend auf den privaten und universitären Bereich beschränkte, stößt heutzutage auf ein ständig wachsendes Interesse seitens der Wirtschaft. In fast allen Softwarebereichen existieren mittlerweile mehr oder weniger gut ausgereifte Open-Source-Pendants zu kommerziellen Produkten, für die jedoch keine Lizenzgebühren zu entrichten sind. Auch am Markt für relationale Datenbanksysteme gibt es immer ernst zu nehmendere Konkurrenz aus der ständig größer werdenden Open-Source-Gemeinde. Zu den Klassikern PostgreSQL und MySQL haben sich vor kurzem SAP DB von SAP AG und InterBase 6 von Borland dazugesellt.

Der Open-Source-Gedanke hat seine Wurzeln in der Entstehungsgeschichte des Betriebssystems UNIX. Das von Richard Stallman gegründete Projekt GNU hatte zum Ziel, ein UNIX-ähnliches Betriebssystem zu schaffen, das ausschließlich aus freien Softwarequellen bestehen sollte. Mit der Entwicklung von Linux und der massiven Verbreitung des Internets war dann die ideale Plattform für Open-Source-Software geschaffen. Das Internet fungiert dabei als Kommunikationsplattform zwischen den Entwicklern und den Benutzern. Open-Source-Software ist dadurch gekennzeichnet, dass sie kostenlos bezogen und eingesetzt werden darf, dass der Quellcode frei zugänglich ist und angepasst werden darf. Er soll sogar erweitert werden, aber immer unter der Voraussetzung, dass die Modifizierungen wieder Open Source sind.

Die Geschichte von PostgreSQL nimmt an der Universität von Kalifornien in Berkeley mit dem Vorläufer Ingres seinen Anfang. Michael Stonebraker arbeitete von 1986 bis 1994 mit seinem Team an dem Nachfolger Postgres, das 1995 von den beiden Studenten Chen und Yu um SQL-Funktionalität erweitert wurde. Das mit SQL fusionierte Postgres bekam dann 1996 den heutigen Namen PostgreSQL.

SAP DB stammt von der kommerziellen Datenbank ADABAS D ab und ist erst seit April 2001 als Open-Source-Datenbank verfügbar. Dieser Schritt soll vor allem der Enterprise Resource Planning (ERP) Software SAP auf die Sprünge helfen, da für SAP nun kein teueres, externes Datenbanksystem angeschafft werden braucht. Weiterentwickelt wird SAP DB selbstverständlich von SAP. Ein Teil der Entwickler von ADABAS D sind nämlich 1997 zu SAP gewechselt. ADABAS D ist hingegen eine Weiterentwicklung des lizensierten Datenbanksystems DDB/4 von Siemens Nixdorf AG, was wiederum auf einem universitären Projekt der TU Berlin basiert.

Der Skandinavier Michael „Monty“ Widenius rief 1995 MySQL ins Leben, als er für die Firma TCX, für die er arbeitete, eine SQL-Schnittstelle zu seiner ISAM-Datenbank schuf. Heute wird MySQL von der eigens dafür gegründeten Firma MySQL AB verbreitet. Obwohl MySQL AB rund um MySQL auch kommerzielle Dienstleistungen und Lizenzen verkauft, ist es bis heute Open Source geblieben.

Diese Diplomarbeit nimmt die beiden bekanntesten Open-Source-Datenbanken (OSDB) PostgreSQL und MySQL, sowie das erst seit kurzem als Open Source freigegebene Datenbanksystem SAP DB unter die Lupe. SAP DB wurde in die Untersuchung mitaufgenommen, da es als ehemals kommerzielles Produkt besonders interessant erscheint. Ziel dieser Arbeit ist es, die drei Systeme untereinander hinsichtlich ihres Funktionsumfangs zu vergleichen, dabei einen Überblick über die theoretischen Konzepte von Datenbank-Management-Systemen (DBMS) zu geben, Erfahrungen mit der Bedienung und dem Verhalten der untersuchten Systeme zu liefern und eine Entscheidungshilfe für die Auswahl der geeignetsten OSDB zu erstellen.

Für die Untersuchung wurden die Anfang 2003 aktuellen Versionen herangezogen, nämlich SAP DB 7.3.23, PostgreSQL 7.2.3 und MySQL 4.0.3 beta MAX. Bei der vorausgehenden Literaturrecherche stellte sich heraus, dass außer populärwissenschaftlicher Literatur kaum relevante, aktuelle wissenschaftliche Publikationen zum Thema vorhanden sind. Deshalb basiert die Arbeit auf einer eingehenden Studie der Benutzerhandbücher, der bekannten Datenbanksysteme betreffenden Standardliteratur sowie einer aus Anwendersicht durchgeführten Softwareanalyse.

Die vorliegende Arbeit gliedert die Untersuchung entsprechend ihrer Zielsetzung in acht Kapitel. Das Kapitel 1 beschäftigt sich mit dem relationalen Datenbankmodell im Allgemeinen und den darin definierten Operationen, auf denen konkrete Abfragesprachen wie SQL beruhen. Es bildet somit die formale Grundlage für die Ausführungen der weiteren Kapitel, deren Verständnis jedoch für den am Endresultat interessierten Leser nicht unbedingt erforderlich ist.
Die eigentliche Untersuchung beginnt mit Kapitel 2, das die Systemarchitektur der OSDB aus Benutzersicht beschreibt und praktische Erfahrungen mit der Installation und Inbetriebnahme der Datenbanksysteme liefert.
Das Kapitel 3 widmet sich der internen Speicherorganisation. Hier werden die unterstützten Datentypen einander gegenübergestellt, und es wird auf die verfügbaren Indexstrukturen und deren Anwendung eingegangen.
Die Abfragesprache SQL, die in allen drei OSDB zum Einsatz kommt, wird in Kapitel 4 vorgestellt. An dieser Stelle wird auch das kapitelübergreifende Begleitbeispiel in SQL defi-

niert. Es soll eine Bibliothek modellieren und ist aus pädagogischen Gründen einfach gehalten. Mögliche Optimierungsansätze in der Ausführung von SQL-Abfragen schließen das Kapitel ab.
Mit dem Themenkomplex Transaktionsverarbeitung befasst sich Kapitel 5. Es zeigt die Problemfälle auf, die ohne Synchronisation auftreten würden, falls Transaktionen im Mehrbenutzerbetrieb quasi parallel abgearbeitet werden. Anschließend werden gewünschte Eigenschaften der Transaktionsverarbeitung definiert und schließlich die Synchronisationskontrollen der OSDB miteinander verglichen.
Das Kapitel 6 widmet sich den Themen Recovery und Replikation und geht insbesondere darauf ein, wie der Datenbankinhalt gesichert und wiederhergestellt werden kann. Im zweiten Teil des Kapitels sehen wir uns Möglichkeiten an, wie man trotz fehlender Replikationsunterstützung Hot-Standby-Lösungen realisieren könnte.
Von zentraler Bedeutung der durchgeführten Untersuchung ist der in Kapitel 7 zusammengestellte Kriterienkatalog für den Vergleich und die Bewertung von (nicht ausschließlich) freien Datenbanksystemen. Dieser soll darüber hinaus einen tieferen Einblick in die Funktionsweise der untersuchten OSDB geben. Zum Umfang dieses Kapitels zählt auch ein ausführlicher Performance-Test, der mit dem frei verfügbaren *Open Source Database Benchmark* im Rahmen der Diplomarbeit durchgeführt wurde.
Abgerundet wird die vorliegende Arbeit mit einer tabellarischen Zusammenfassung und einem Ausblick in Kapitel 8. Dieser beinhaltet einen Vergleich mit dem kommerziellen Datenbanksystem Oracle und zeigt (noch vorhandene) Unzulänglichkeiten der OSDB auf.
Von praktischem Interesse dürfte die im Anhang abgedruckte Beispielsammlung von Programmen unterschiedlicher Programmiersprachen sein, die den Zugriff auf die drei getesteten OSDB demonstrieren. Zusammen mit dem Abkürzungs- und Literaturverzeichnis bildet diese den Abschluss der Arbeit.

Bei der Erstellung der Arbeit wurde nach einem weit gehend einheitlichen Strukturierungsprinzip vorgegangen. Die Kapitel enthalten zu Beginn zumeist einen allgemeinen Teil, in welchem die Konzepte und die Gemeinsamkeiten theoretisch erläutert werden. Im Anschluss daran folgt eine Betrachtung der untersuchten Datenbanksysteme. Diese Gestaltung erscheint uns sinnvoll, da ein Vergleich in dieser Form direkt möglich wird.

Kapitel 1

Das relationale Datenbankmodell

Die Open-Source-Datenbanksysteme SAP DB, PostgreSQL und MySQL basieren auf dem von E. F. Codd entwickelten relationalen Datenbankmodell, der Grundlage aller relationalen Datenbanksysteme. In diesem Kapitel geben wir daher die formale Notation für Relationen, Schemata, Abhängigkeiten und Datenbanken an und beschreiben zwei zentrale Paradigmen, nämlich die Relationenalgebra und die deskriptiven Relationenkalküle.
Zuvor betrachten wir als Einstieg etwas informeller folgende kleine Datenbank, die aus den Tabellen *Buch*, *Ausleihe* und *Leser* besteht:

Buch

BuchNr	Autor	Titel	Verlag
123	Date	Intro DBS	AW
234	Ullmann	DBS Implementation	PH
345	Vossen	Datenmodelle, …	OB

Ausleihe

BuchNr	LeserNr	Entlehndat
123	225	28.01.2003
234	347	19.11.2002

Leser

LeserNr	Name
225	Claudia
347	Viktor

Abbildung 1.1 Beispiel einer relationalen Datenbank [4]

Das Ziel von Datenbanken ist es, Informationen aus unserer Umwelt in einer möglichst redundanzfreien, natürlichen und effizienten Form zum Zweck der Archivierung und automationsgestützten Verarbeitung abzuspeichern. Eine relationale Datenbank besteht also aus der Sicht des Benutzers aus Tabellen, die Informationen strukturiert speichern. Eine Tabelle besteht wiederum aus Zeilen und Spalten. Jede Zeile repräsentiert dabei einen Datensatz (Record) und jede Spalte ein Attribut des Datensatzes. Die Tabellen Buch und Leser können als Entitäten aufgefasst werden, während die Tabelle Ausleihe eine Beziehung (Relationship) zwischen den Entitäten Buch und Leser darstellt. Die Bausteine von Datenbanken sind Entitäten und Beziehungen, mit denen versucht wird, einen Ausschnitt unserer Umwelt zu modellieren. Tabellen sind aus mengentheoretischer Sicht Relationen. Die Metainformation, aus welchen Attributen eine Tabelle besteht, in Abbildung 1.1 die Kopfzeile der Tabellen, bezeichnen wir als Schema. Die Leser- und Buchnummer identifizieren einen Datensatz und sind daher für jeden Eintrag der Tabelle unterschiedlich. Man nennt solche einen ganzen Datensatz identifizierende Attribute Schlüsselattribute. Diese werden für den Zugriff und den Verweis auf einzelne Datensätze verwendet.

Sowohl zwischen den Attributwerten innerhalb einer Tabelle als auch zwischen mehreren Tabellen kann es Abhängigkeiten geben, die in Form von Bedingungen ausgedrückt werden. Diese sorgen dafür, dass das, was in der Datenbank abgespeichert wird, auch „Sinn“ macht. Sie schränken die möglichen Zustände einer Datenbank ein, wobei wir mit Zustand die aktuelle Wertebelegung der Tabellen bezeichnen. Die Tabelle Ausleihe ist ein Beispiel für eine interrelationale Abhängigkeit, eine Abhängigkeit vom zweiten Typ, denn diese ist durch die (Fremdschlüssel-)Attribute *BuchNr* und *LeserNr* mit den Tabellen Buch und Leser verknüpft. Das bedeutet, dass die Daten nur dann „Sinn“ machen, falls nur solche Bücher von solchen Lesern ausgeliehen werden, die auch tatsächlich in der Datenbank existieren.

Im Folgenden wollen wir formalere Definitionen der oben erwähnten Begriffe anführen.

1.1 Relationen

Tabellen können mit dem aus der Mengenlehre bekannten Konzept der Relationen beschrieben werden. Sie sind eine Teilmenge des kartesischen Produktes über den Wertebereich (Domain) aller Attribute der Tabelle.

Definition 1.1[1]

Sei X eine (endliche) Attributmenge, das heißt $X = \{A_1,\ldots,A_m\}$. Jedes Attribut besitzt einen nicht leeren Wertebereich dom(A) mit mindestens zwei Elementen und dom(X) bezeichnet die Vereinigung von $dom(A_1)$ bis $dom(A_m)$.

- Ein Tupel über X ist eine Abbildung $\mu: X \rightarrow dom(X)$ für die gilt $(\forall A \in X)\ \mu(A) \in dom(A)$. Tup(X) bezeichne die Menge aller Tupel über X.
- Eine Relation r über X ist eine (endliche) Menge von Tupeln über X, das heißt $r \subseteq Tup(X)$. Mit Rel(X) bezeichnen wir die Menge aller Relationen über X.

Ein Tupel μ ist also ein konkreter Datensatz. Manchmal bleiben bestimmte Attribute eines Datensatzes frei, das heißt ungesetzt. Man spricht dann von so genannten Nullwerten. Diese kann man durch eine Erweiterung der Tupelabbildung modellieren, indem μ eine Abbildung von X in $dom(X) \cup N$ ist, wobei N für den Nullwertebereich steht.

Definition 1.2

Das X aus Definition 1.1 nennen wir Relationenschema. Wir schreiben auch $R(A_1,\ldots,A_m)$.

1.2 Datenbanken

Formal können wir nun eine Datenbank wie folgt definieren:

Definition 1.3

Eine (relationale) Datenbank d über $\mathbf{R} = \{R_1,\ldots,R_k\}$ (Menge von Relationenschemata) ist eine Menge von (Basis-)Relationen, $d = \{r_1,\ldots,r_k\}$, mit $r_i \in Rel(X_i)$ für $1 \leq i \leq k$.
Dat(**R**) bezeichne die Menge aller Datenbanken über **R**.

1.3 Datenabhängigkeiten

Wir unterscheiden Abhängigkeiten zwischen den Attributen einer Tabelle und Abhängigkeiten zwischen mehreren Tabellen. Die hier definierten Abhängigkeiten drücken semantische Bedingungen aus, die zu einem festen Zeitpunkt überprüft werden können. Die Einhaltung dieser Bedingungen beschränkt die Menge Rel(X) aller möglichen Relationen über X. Beispielsweise soll es keine zwei Tupel mit gleichem Schlüssel aber verschiedenen Attributwerten geben. Genauso sollen Verknüpfungen mit anderen Tabellen, stets auf existierende Datensätze verweisen. Die zweite Eigenschaft wird in der Literatur als „referenzielle Integrität“ bezeichnet. Üblicherweise kontrolliert ein DBMS mindestens die Verwendung von eindeutigen Schlüsselwerten und die Einhaltung der referenziellen Integrität.

---

[1] Vgl. [4].

1.3.1 Intrarelationale Abhängigkeiten

Definition 1.4

Eine intrarelationale Abhängigkeit auf alle möglichen Relationen über X ist eine Abbildung σ: Rel(X) $\rightarrow$ {0,1}, wobei 1 für „Abhängigkeit erfüllt“ bzw. 0 für „Abhängigkeit nicht erfüllt“ steht. Wir fassen mehrere Abhängigkeiten zu einer Menge $\sum_X=\{\sigma_1,\ldots,\sigma_n\}$ zusammen und definieren dafür folgende Abbildung $\sum_X$: Rel(X) $\rightarrow$ {0,1} mit $\sum_X(r) := \min(\sigma_1(r),\ldots,\sigma_n(r))$.

Als Beispiel für eine intrarelationale Abhängigkeit betrachten wir als nächstes (Primär-) Schlüssel:

Definition 1.5

Sei X eine Attributmenge, K $\subseteq$ X.

- K heißt **Schlüssel** für r $\in$ Rel(X), falls gilt
  (a) ($\forall$ $\mu,\nu \in$ r) μ[K] = ν[K] $\Rightarrow$ μ = ν
  (b) für keine echte Teilmenge K' $\subset$ K gilt (a)
- Eine **Schlüsselabhängigkeit** K $\rightarrow$ X bezeichnet folgende semantische Bedingung
  Sei r $\in$ Rel(X):
  (K $\rightarrow$ X)(r) := 1, falls K Schlüssel für r
  0, sonst

μ[K] ist die Einschränkung der Relation μ auf die Attribute K $\subseteq$ X. Außerdem heißt ein Schlüssel K $\subseteq$ X trivial, falls K = X gilt. Mit einer Schlüsselabhängigkeit K $\rightarrow$ X kann getestet werden, ob eine Relation r mit den Attributen X die Schlüssel-Bedingung erfüllt, das heißt, ob K tatsächlich ein Schlüssel von r ist. Die Definition des Relationenschemas kann durch das Hinzufügen von Abhängigkeitsbeziehungen $\sum_X$ erweitert werden: R = (X, $\sum_X$)
Schlüssel sind laut obiger Definition eine nicht notwendigerweise eindeutige, minimale Menge von Attributen einer Relation, die herangezogen werden, um jedes Tupel eindeutig zu indentifizieren. **Fremdschlüssel** hingegen sind Referenzen auf andere Relationen. Die in Abbildung 1.1 unterstrichenen Attribute sind die Schlüssel der abgebildeten Relationen. BuchNr und LeserNr in Ausleihe sind Fremdschlüssel, die ein bestimmtes Tupel in den Relationen Buch respektive Leser referenzieren.

Um Relationen, die Abhängigkeiten verletzen, von gültigen zu unterscheiden, definieren wir:

Definition 1.6

Die Menge Sat(X, $\sum_X$) := {r $\in$ Rel(X) | $\sum_X$(r) = 1} gibt die Menge aller Relationen über X an, welche die Abhängigkeiten $\sum_X$ erfüllen.

1.3.2 Interrelationale Abhängigkeiten

Analog zu den zuvor definierten intrarelationalen Abhängigkeiten können wir wieder eine Abbildung auf {0,1} definieren, die angibt, ob eine interrelationale Abhängigkeit gilt oder nicht. Erweitern wir dies wieder zu Mengen von Abhängigkeiten, ergibt sich
$\sum_{\mathbf{R}}$: Dat(**R**) $\rightarrow$ {0,1} mit $\sum_{\mathbf{R}}$ (d) := min(σ_1(d),…,σ_n(d)).
Ebenso kann die Menge der gültigen Datenbanken Sat($\sum_{\mathbf{R}}$) := {d $\in$ Dat(**R**) | $\sum_{\mathbf{R}}$(d) = 1} definiert werden.
Die Abhängigkeitsabbildung $\sum_{\mathbf{R}}$ ist durch eine prädikatenlogische Formel darstellbar. Der genaue Aufbau und die Bedingungen, die diese erfüllen muss, sind in [4] beschrieben.
Als wichtiges Beispiel für interrelationale Abhängigkeiten betrachten wir Inklusionsabhängigkeiten, die uns erlauben, die Einhaltung der referenziellen Integrität auszudrücken.

Definition 1.7

Es sei **R** eine Menge von Relationenschemata, R_i, $R_j \in$ **R**, $R_i \neq R_j$, $R_i = (X_i, \sum_i)$, $R_j = (X_j, \sum_j)$. Ferner sei V eine Folge von *n* verschiedenen Attributen aus X_i, W eine Folge von *n* verschiedenen Attributen aus X_j. Eine Inklusionsabhängigkeit IND $R_i[V] \subseteq R_j[W]$ (Inclusion Dependency) bezeichnet folgende semantische Bedingung: Sei d $\in$ Dat(**R**):

$$(R_i[V] \subseteq R_j[W])(d) := \begin{cases} 1, \text{ falls } \{\mu[V] \mid \mu \in r_i\} \subseteq \{\mu[W] \mid \mu \in r_j\} \\ 0, \text{ sonst} \end{cases}$$

Beispiel 1.1

Die Beziehung zwischen Leser und Ausleihe aus Abbildung 1.1 kann als Inklusionsbeziehung formalisiert werden. Jeder Leser, der sich ein Buch ausleiht, muss auch in der Relation Leser(LeserNr, Name) existieren. Formal also: IND Ausleihe[LeserNr] $\subseteq$ Leser[LeserNr]

1.4 Konsistenz

Definition 1.8

- Eine Datenbank d $\in$ Dat(**R**) heißt punktweise konsistent, falls $r_i \in$ Sat(R_i) für alle $r_i \in$ d gilt. Sat(**R**) bezeichne die Menge aller punktweise konsistenten Datenbanken über **R**.
- Eine Datenbank d $\in$ Dat(**R**) heißt konsistent, falls d $\in$ Sat(R, $\sum_{\mathbf{R}}$) := Sat(**R**) $\cap$ Sat($\sum_{\mathbf{R}}$) gilt.

Eine Datenbank ist also konsistent, wenn sie sowohl die intra- als auch interrelationalen Abhängigkeiten erfüllt. Intuitiv sind die Konsistenz und die im nächsten Abschnitt erläuterten Integritätsbedingungen dafür verantwortlich, dass der aktuelle Zustand der Datenbank Sinn macht, das heißt in der modellierten Wirklichkeit auch wirklich vorkommen kann.

1.5 Integritätsbedingungen

Neben den zuvor erläuterten Datenabhängigkeiten, die semantische Bedingungen darstellen, möchte man oft noch weiter reichende Beschränkungen und Plausibilitätskontrollen in das Datenbanksystem integrieren. Früher waren die Applikationsprogramme dafür zuständig, Integritätsbedingungen zu kontrollieren, während heute diese Aufgabe vom Datenbanksystem immer mehr übernommen wird.
Prinzipiell unterscheiden wir zwischen statischen, transitionalen und dynamischen Integritätsbedingungen. Statische Bedingungen können zu jedem festen Zeitpunkt überprüft werden und hängen nur vom momentanen Datenbankzustand ab. Transitionale Bedingungen nehmen den letzten Zustand der Datenbank in die Betrachtung mit auf. Unter Zustandsänderungen verstehen wir hier das Hinzufügen, Löschen oder Ändern von Tupeln. Ein Beispiel für eine transitionale Bedingung ist, dass das Alter einer Person nur steigen darf. Dynamische Integritätsbedingungen sind am flexibelsten, da sie den gesamten Zustandsänderungsverlauf berücksichtigen können. Damit lässt sich zum Beispiel eine Bedingung definieren, die den maximalen Kursverlust einer Aktie pro Tag auf zehn Prozent begrenzt.
Weiters unterscheidet man, ob sich die Bedingung auf ein einzelnes Attribut, ein Tupel, eine Relation oder auf die gesamte Datenbank bezieht. Mit dem Werkzeug der Integritätsbedingungen lässt sich das dynamische Verhalten der Datenbank derart steuern, dass keine unrealistischen Zustandsübergänge stattfinden.

Beispiel 1.2

Typische statische Integritätsbedingungen (Integrity Constraints) können sein:

- **Domain- oder Attribut-Bedingungen**: Ober-, Untergrenzen für numerische Werte, Festlegung bestimmter Werte im Sinne einer Aufzählung (zum Beispiel Ja, oder Nein), NOT NULL-Bedingungen (ein Attribut muss einen Wert besitzen), …
- **Tupel-Bedingungen**: Betreffen einzelne Tupel innerhalb einer gegebenen Relation. Zum Beispiel soll in einer Relation, die Zugangsdaten speichert, sichergestellt werden, dass das Passwort nicht mit dem Geburtsdatum oder gar dem Benutzernamen übereinstimmt.
- **Relationen-Bedingungen**: Betreffen die Menge aller Tupel einer Relation und können zum Beispiel sein: Schlüssel-Bedingungen (BuchNr soll der Primärschlüssel der Relation Buch sein), Aggregat-Bedingungen (die Summe der Gehälter einer Abteilung darf eine bestimmte Obergrenze nicht überschreiten), rekursive Bedingungen wie „jeder Ort ist von jedem anderen aus erreichbar“ in einer geografischen Datenbank, …
- **Referenzielle Bedingungen**: Spezifizieren semantische Verbindungen zwischen Relationen, meistens in Form von Fremdschlüsseln, siehe Abschnitt 1.3.2.

Integritätsbedingungen schränken den Zustandsraum von Datenbanken semantisch ein und ermöglichen eine syntaktische Überprüfung, ob die Datenbank sinnvolle Werte speichert oder nicht. In der Praxis wird man Zustandsübergänge, die Integritätsbedingungen verletzen, nach Möglichkeit nicht zulassen und mit einer Fehlermeldung durch das DBMS abweisen.

1.6 Operationen im relationalen Datenbankmodell

Als nächstes wollen wir untersuchen, welche Operationen wir auf den zuvor definierten Relationen anwenden können. Wir wollen zum Beispiel in einer Datenbank ein bestimmtes Buch suchen, oder jene Leser mahnen, die Bücher nicht termingerecht zurückgegeben haben.
Als formales Werkzeug für derartige Abfragen können wir auf die von Codd eingeführte Relationenalgebra oder auf den der Prädikatenlogik ähnlichen Relationenkalkül zurückgreifen. Beide Ansätze erweisen sich als gleich ausdrucksstark, das heißt zu jedem Ausdruck aus der Relationenalgebra lässt sich ein äquivalenter Ausdruck aus dem Relationenkalkül angeben und umgekehrt.

1.6.1 Relationenalgebra

Im Gegensatz zu den Relationenkalkülen ist dies der prozedurale Ansatz Abfragen zu beschreiben. Mit Hilfe der in der folgenden Definition eingeführten Operationen entstehen aus (Basis-)Relationen neue abgeleitete Relationen.

Definition 1.9

Sei $R,S = (X, \sum_X)$ ein Relationenschema, $r,s \in Rel(X)$ und $Y \subseteq X$:

- **Projektion** (π)
  $\pi_Y(r) := \{\mu[Y] \mid \mu \in r\}$ ist die Projektion von r auf Y, das heißt die Einschränkung einer Relation auf die Attribute Y.

- **Selektion** (σ)
  $\sigma_{A\Theta a}(r) := \{\mu \in r \mid \mu(A)\ \Theta\ a\}$ bzw. $\sigma_{A\Theta B}(r) := \{\mu \in r \mid \mu(A)\ \Theta\ \mu(B)\}$ ist die Selektion von r bezüglich einer Vergleichsbedingung ($A\Theta a$ bzw. $A\Theta B$), die entscheidet, welche Tupel in das Ergebnis aufgenommen werden. A, B sind Attribute von R, a ist eine Konstante und Θ ist ein Vergleichsoperator aus $\{<,\leq,>,\geq,=,\neq\}$. Zu beachten ist, dass A, B und a vergleichbar sein müssen, das heißt aus derselben Domain stammen und unter Umständen geordnet sind. Atomare Selektionen der Form $A\Theta B$ dürfen auch zu komplexeren boolschen Ausdrücken erweitert werden.

- **Kartesisches Produkt** ($\times$)
  $r_1 \times r_2 := \{\mu \in Tup(X_1X_2) \mid \mu[X_1] \in r_1 \wedge \mu[X_2] \in r_2\}$, wobei X_1, X_2 die (disjunkten) Attributmengen von den Relationen r_1 und r_2 sind. Sind die Attributmengen von r_1 und r_2 nicht disjunkt, müssen gleichnamige Spalten zuvor umbenannt werden. $r_1 \times r_2$ erzeugt also eine neue Relation, deren Tupel aus der Zusammensetzung der Tupel aus r_1 und r_2 entstehen und dessen Attributmenge die Vereinigung von X_1 und X_2 ist.

- **Umbenennung** ($\rho_{B \leftarrow A}$)
  $\rho_{B \leftarrow A}(r)$ benennt in der Relation r das Attribut A in B um.

- **Verbund** ($\bowtie$)
  $\bowtie_{i=1,\ldots,n} r_i := \{\mu \in Tup(\cup_{i=1,\ldots,n} X_i) \mid (\forall i, 1 \leq i \leq n)\ \mu[X_i] \in r_i\}$ ist der natürliche Verbund (Natural Join) von den Relationen $r_1,\ldots,r_n$ mit den Attributmengen $X_1,\ldots,X_n$. Der natürliche Verbund verknüpft mehrere Relationen, die sich durch ein oder mehrere gemeinsame Attribute referenzieren.
  Beispiel für n = 2: Ausleihe $\bowtie$ Buch erzeugt folgende neue Relation

BuchNr	Autor	Titel	Verlag	LeserNr	Entlehndat
123	Date	Intro DBS	AW	225	28.01.2003
234	Ullmann	DBS Implementation	PH	347	19.11.2002

- **Vereinigung** ($\cup$)
  $r \cup s := \{\mu \in Tup(X) \mid \mu \in r \vee \mu \in s\}$

- **Durchschnitt** ($\cap$)
  $r \cap s := \{\mu \in Tup(X) \mid \mu \in r \wedge \mu \in s\}$

- **Differenz** ($-$)
  $r - s := \{\mu \in r \mid \mu \notin s\}$

In der Literatur findet man noch eine Reihe weiterer Operationen (wie die Division, Equi-, Theta- und Semijoin), die jedoch mit den oben angeführten Operationen ausgedrückt werden können.

Beispiel 1.3

Suche nach dem Buch mit der Nummer 123: $\sigma_{BuchNr=123}(Buch)$

Beispiel 1.4

Zeige die Namen jener Leser an, die Bücher länger als 4 Wochen ausgeborgt haben:
$\pi_{Name}((\sigma_{aktuelles\ Datum\ >\ (Entlehndat\ +\ 4\ Wochen)}(Ausleihe)) \bowtie Leser)$

1.6.2 Relationenkalküle

Relationenkalküle basieren auf der Prädikatenlogik erster Stufe und stellen den deklarativen Ansatz für das Formulieren von Abfragen dar. Wir unterscheiden je nach Bedeutung der Variablen zwei Varianten der Relationenkalküle, den Relationen-Tupel-Kalkül und den Relationen-Domain-Kalkül. Die Abfragesprache SQL, die in den meisten relationalen Datenbanksystemen zum Einsatz kommt, fällt in die erste Kategorie.

1.6.2.1 Relationen-Tupel-Kalkül (RTK)

Ausdrücke des RTK haben die Form

$$E = \{t \mid \delta(t)\},$$

wobei δ eine Formel des RTK und t die einzige freie Tupelvariable in δ ist. Damit ein Tupel in der Ergebnisrelation E aufgenommen wird, muss sie die Formel δ erfüllen. δ beschreibt also deklarativ, wann dies der Fall sein soll.
Formeln des RTK sind der Prädikatenlogik ähnlich, wobei Variablen ganze Tupel einer Relation repräsentieren. Es gelten noch gewisse praktische Einschränkungen, die in [4] zusammengefasst sind.

Schauen wir uns ein paar Beispiele an, wie Ausdrücke der Relationenalgebra (RA) in RTK-Ausdrücke umgeformt werden können.

Beispiel 1.5

- $r \cup s$ in RA entspricht $\{ t \mid r(t) \vee s(t) \}$
- die Projektion $\pi_{AB}(r)$ der Relation $r \in Rel(ABCD)$ wird zu
  $\{ t \mid (\exists u) (r(u) \wedge t[A]=u[A] \wedge t[B]=u[B]) \}$
- die Selektion $\sigma_{BuchNr=123}(Buch)$ wird zu$\{ t \mid Buch(t) \wedge t[BuchNr] = 123 \}$
- Ausleihe $\bowtie$ Buch entspricht
  $\{ t \mid (\exists u \exists v) (Ausleihe(u) \wedge Buch(v) \wedge u[BuchNr] = v[BuchNr] \wedge$
  $t[BuchNr,Autor,Titel,Verlag] = v[BuchNr,Autor,Titel,Verlag] \wedge$
  $t[LeserNr,Entlehndat] = u[LeserNr,Entlehndat]) \}$

1.6.2.2 Relationen-Domain-Kalkül (RDK)

Der wesentliche Unterschied zu RTK besteht in der Verwendung von Variablen für Elemente von Wertebereichen (Domain-Variablen) an Stelle von Tupelvariablen. Ausdrücke des RDK haben die Form

$$E = \{x_1 \ldots x_n \mid \delta(x_1,\ldots,x_n)\},$$

wobei $x_1,\ldots,x_n$ für Attribute stehen und die einzigen freien Variablen in der RDK-Formel δ sind.

Wir vergleichen im folgenden Beispiel die RTK-Ausdrücke aus Beispiel 1.5 mit den korrespondierenden RDK-Ausdrücken, um den Unterschied zu veranschaulichen.

Beispiel 1.6

- RA $r \cup s$
  RTK $\{ t \mid r(t) \vee s(t) \}$
  RDK $\{ x_1 \ldots x_n \mid r(x_1,\ldots,x_n) \vee s(x_1,\ldots,x_n) \}$
- RA $\pi_{AB}(r)$
  RTK $\{ t \mid (\exists u)\, (r(u) \wedge t[A]=u[A] \wedge t[B]=u[B]) \}$
  RDK $\{ x\ y \mid (\exists v)(\exists w)\, (r(x,y,v,w)) \}$
- RA $\sigma_{BuchNr=123}(Buch)$
  RTK $\{ t \mid Buch(t) \wedge t[BuchNr]=123 \}$
  RDK $\{ BuchNr\ Autor\ Titel\ Verlag \mid Buch(BuchNr,Autor,Titel,Verlag) \wedge BuchNr=123 \}$
- RA $Ausleihe \bowtie Buch$
  RTK $\{ t \mid (\exists u \exists v)\ (\ Ausleihe(u) \wedge Buch(v) \wedge u[BuchNr]=v[BuchNr] \wedge$
  $t[BuchNr,Autor,Titel,Verlag]=v[BuchNr,Autor,Titel,Verlag] \wedge$
  $t[LeserNr,Entlehndat]=u[LeserNr,Entlehndat]\ ) \}$
  RDK $\{ BuchNr\ Autor\ Titel\ Verlag\ LeserNr\ Entlehndat \mid$
  $Ausleihe(BuchNr,LeserNr,Entlehndat) \wedge Buch(BuchNr,Autor,Titel,Verlag) \}$

Kapitel 2
Systemarchitektur

Wir wollen in diesem und in den folgenden Kapiteln die drei OSDB genauer vorstellen. Wir betrachten dazu die Systemarchitektur, die interne Speicherverwaltung und Aspekte der Abfrage- und Transaktionsverarbeitung. In diesem Kapitel geben wir einen Überblick über die Komponenten und mitgelieferten Tools der einzelnen Datenbanksysteme. Die Betrachtung erfolgt dabei aus der Sichtweise des DB-Administrators, der für die Installation, den Betrieb und die Wartung zuständig ist, sowie aus Sicht der Benutzer, die auf das System zugreifen. Wir zeigen, wie die Installation abläuft, wie das DBMS gestartet wird und wie man eine neue Datenbank einrichtet.
Die folgenden Ausführungen basieren allesamt auf den UNIX-Versionen der untersuchten OSDB. SAP DB und MySQL sind auch für Microsoft Windows erhältlich, wobei SAP DB unter Windows ein grafisches Frontend anbietet, das nicht in UNIX portiert worden ist. Ansonsten sind die Unterschiede marginal.

2.1 Grundlagen

Nach [4] besteht ein Datenbanksystem einerseits aus einer oder mehreren Datenbanken[2] und einem Softwarepaket, Datenbank-Management-System (DBMS) genannt, das die Datenbanken verwaltet und den Zugriff von außen regelt. Das DBMS stellt also die Schnittstelle zwischen den Daten und den dazugehörigen Applikationen her. Eine Datenbank umfasst nicht nur die Daten selbst, sondern auch alle notwendigen Metainformationen (wie Schemata, Benutzerrechte, Datenbankparameter, etc.) und zwecks Recovery eine Form von Logdatei, in der Änderungen an den Daten mitprotokolliert werden (siehe Kapitel 6).

[2] Eine Datenbank ist gemäß Definition 1.3 eine Sammlung von mehreren Tabellen.

Ein DBMS kann logisch in folgende Komponenten gegliedert werden, die sich auch in den OSDB wiederfinden. In den nachfolgenden Kapiteln werden wir uns mit den hier aufgezählten Konzepten und Begriffen näher auseinandersetzen.

- **Abfrageverarbeitung**
  Das DBMS bietet zumeist SQL als Datenbanksprache an, womit sämtliche Datendefinitionen, Datenmanipulationen und Abfragen abgewickelt werden. Bestandteile der Abfrageverarbeitung sind: Parser, Compiler, Autorisierungskontrolle, Integritätsprüfung, Optimierer und der Update- sowie Query-Prozessor

- **Transaktionsverarbeitung**
  Die Transaktionsverarbeitung ermöglicht einen kontrollierten gleichzeitigen Zugriff auf die Daten. Dazu werden logisch zusammengehörende Operationen in eine Transaktion verpackt, die als Einheit angesehen wird und als solche entweder vollständig oder gar nicht ausgeführt wird. Die Aufgabe des DBMS ist es, Transaktionen, die auf dieselben Datenobjekte zugreifen, zu synchronisieren. Dies übernimmt der Scheduler in Zusammenarbeit mit dem Recovery-Manager, der dafür sorgt, dass sich die Datenbank jederzeit in einem konsistenten Zustand befindet. Der Scheduler entscheidet, in welcher Reihenfolge die Befehle der einzelnen Transaktionen abgearbeitet werden sollen.

- **Speicherverwaltung**
  Aufgabe der Speicherverwaltung ist es, die Daten der einzelnen Tabellen in einer möglichst effizienten Datenstruktur abzuspeichern und die Daten auf Platte persistent zu halten. Da die Datenbank im Allgemeinen nicht vollständig in den Hauptspeicher geladen werden kann, werden immer nur kleine Teile davon in einem Cache gehalten.

- **Schnittstellen**
  Um mit dem DBMS in Kontakt zu treten und seine Dienste von außen zu nutzen, gibt es zumeist die Möglichkeit über einen TCP/IP-Port auf den Datenbankserver zuzugreifen. Da der Zugriff zumeist über ein proprietäres Protokoll abgewickelt wird, benötigt man Treiber für standardisierte Schnittstellen wie ODBC oder JDBC, um von anderen Programmen, die nicht zum DBMS gehören, transparent zugreifen zu können. Oft werden auch eigene APIs zur Verfügung gestellt, um das DBMS von höheren Programmiersprachen wie C anzusprechen.

- **Externe Tools**
  Neben den vorher erwähnten Komponenten, die den unmittelbaren Datenbankserver ausmachen, bietet ein DBMS noch eine Reihe von externen Tools an, um auf die Datenbanken zuzugreifen und sie zu verwalten. Dazu zählen beispielsweise ein interaktives SQL-Tool, mit dem SQL-Befehle verarbeitet werden können, aber auch Tools, um ein Backup und Restore durchzuführen.

2.2 SAP DB

SAP DB 7.3.23 ist ein relationales DBMS, das für Online Transaction Processing (OLTP) konstruiert wurde, sprich Transaktionsverarbeitung unterstützt. Der Source Code wird unter der GNU GPL bzw. LGPL [11] vertrieben, ist jedoch kaum kommentiert und ein Mix aus den Programmiersprachen C, C++ und Pascal. Einen Überblick über die wichtigsten Features, die im Laufe dieser Arbeit noch genauer erläutert werden, kann folgender Liste entnommen werden:

- SQL/92-Entry-Level-Unterstützung mit vielen Erweiterungen
- IBM DB2 Version 4 und Oracle7 SQL-Unterstützung
- Unterschiedliche Isolationsgrade von READ COMMITTED bis SERIALIZABLE
- Referenzielle Integrität und Constraints
- BLOB-Datentyp
- Sub-SELECTs
- Views, die teilweise auch Updates zulassen
- Trigger und Stored Procedures
- Sub-Transaktionen
- Sequenzen
- Rechteverwaltung mittels Rollen
- Temporäre Tabellen
- Sperren auf Zeilenebene
- Online Backup
- Inkrementelles Backup
- Grafisches Frontend, jedoch nur unter Windows verfügbar
- Administration und Abfragen über ein eigenes Webinterface
- Keine explizite Reorganisation notwendig
- Verschiedene Programmierschnittstellen (ODBC, JDBC, Perl, Pytho, PHP)
- Precompiler für C/C++

2.2.1 Architektur

SAP DB kann eine oder mehrere Datenbanken verwalten, die Datenbankinstanzen genannt werden. Jede Instanz besteht aus mehreren *Threads*, *Caches* und *Devspaces*. Für zukünftige Entwicklungen werden verschiedene Datenbankinstanztypen unterschieden. Der Standardtyp ist OLTP und ermöglicht Transaktionsverarbeitung und parallelen Benutzerbetrieb. Der Typ LiveCache ist für objektorientierte Datenbanken, die im Hauptspeicher gehalten werden, gedacht, ist aber auf Grund fehlender Dokumentation für den Open-Source-Bereich noch nicht einsatzfähig. Beispiele für Threads sind der *User Task*, der *Data Writer* und der *Log Writer*. Der User Task verwaltet je eine Verbindung (Session) von einem Client. Der Data Writer und der Log Writer schreiben Einträge in die Devspaces. Ein Devspace kann eine reguläre Datei bestimmter Größe oder ein RAW-Device unter UNIX sein. Jede Datenbankinstanz hat genau einen *System* Devspace und je ein oder mehrere *Data* und *Log* Devspaces. In den Data Devspaces werden die eigentliche Datenbank und die dazugehörige Metainformation, Catalog genannt, gespeichert. In den Log Devspaces werden alle Änderungen an den Daten mitprotokolliert. Sollte der Speicherplatz in den Data oder Log Devspaces knapp werden, kann neuer Speicherplatz in Form von neuen Devspaces hinzugefügt werden. Die maximale Anzahl von Devspaces muss aber bei der Einrichtung einer neuen Datenbankinstanz angegeben werden.

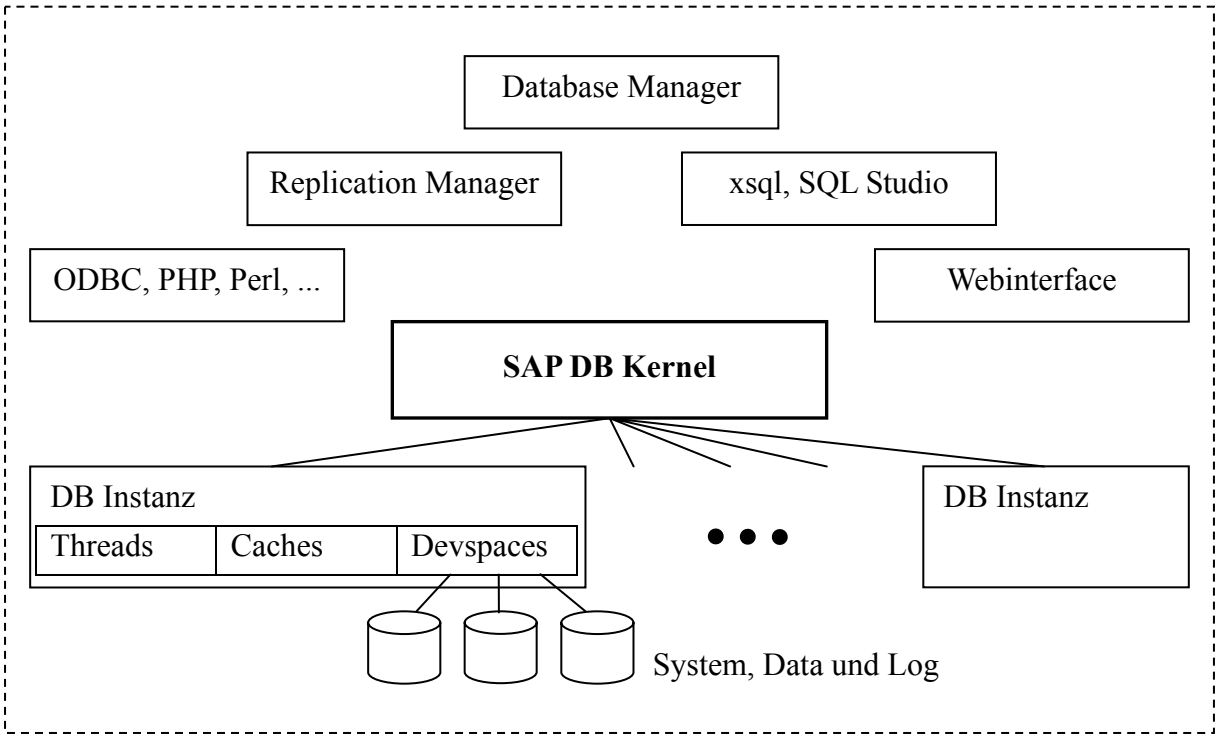

Abbildung 2.1 Architektur von SAP DB [1]

Der *Database Manager* ist ein Tool, das es sowohl in einer Kommandozeilenvariante als auch als grafisches GUI für Windows gibt. Damit können neue Datenbankinstanzen angelegt und bestehende verwaltet werden. Eine wichtige Aufgabe des Database Managers ist das Anlegen von Backups. Für interaktives SQL stehen *xsql* in der Kommandozeile und unter Windows *SQL Studio* zur Verfügung. Es ist auch möglich, ein Webinterface für SQL-Zugriffe und Database-Manager-Aufgaben einzurichten. Dazu muss allerdings ein alternativer Webserver (zum Beispiel Apache) vorhanden sein, da der mitgelieferte Webserver zu instabil läuft. Der *Replication Manager* ermöglicht das Extrahieren und Importieren von Tabellen und Datenbanken in verschiedenen Formaten.

2.2.2 Installation

Der Source Code, vorkompilierte Binärdistributionen und RPM-Pakete für Linux können von der SAP DB-Homepage http://www.sapdb.org bezogen werden. Wir zeigen im Folgenden wie eine Binärdistribution unter Linux installiert wird.

1. Herunterladen der Distribution: sapdb-server-linux-32bit-i386-7_3_0_23.tgz
2. Entpacken des Tarballs:

   ```
   $ tar –xpvzf sapdb-server-linux-32bit-i386-7_3_0_23.tgz
   ```

3. Starten des Installationsprogramms SDBINST:

   ```
   $ ./SDBINST
   ```

4. Wahl der Zielverzeichnisnamen und des Benutzer- und Gruppennamens, unter dem der Server laufen soll.

Als Benutzer auf Betriebssystemebene könnte man zum Beispiel den Benutzer *sapdb* in der Gruppe *sapdb* mit dem Passwort *sapdb* anlegen (useradd sapdb -g sapdb ; passwd sapdb).
Die Verzeichnisstruktur sieht bei einer Standardinstallation wie folgt aus:

```
$ ls –R /usr/sapdb #(verkürzte Ausgabe)
/usr/sapdb:
.  ..  depend  indep_data  indep_prog
/usr/sapdb/depend:
. ..  bin  env  etc  incl  lib  misc  pgm  runtime  sap  wrk
/usr/sapdb/indep_data:
.  ..  config  wrk
```

```
/usr/sapdb/indep_prog:
.  ..  bin  etc  pgm  runtime  terminfo  wrk
```

Das Verzeichnis *depend* ist versionsabhängig, während *indep_data* und *indep_prog* versionsunabhängig sind. Der Datenbankkernel und die mitgelieferten Tools werden über indep_prog/bin/ aufgerufen. Das Verzeichnis indep_prog stellt also eine versionsunabhängige Schnittstelle zu den im Verzeichnis depend enthaltenen Programmen dar. In indep_data werden Konfigurationsdateien der angelegten Datenbankinstanzen und standardmäßig die Devspaces gespeichert.

2.2.3 Starten des Servers

Der Server wird unter dem zuvor angelegten Benutzer sapdb mit

```
$ /usr/sapdb/indep_prog/bin/x_server start
```

gestartet und kann mit

```
$ /usr/sapdb/indep_prog/bin/x_server stop
```

wieder beendet werden. Dabei wird eine Logdatei *vserver.prot* im Verzeichnis indep_data/wrk angelegt.

2.2.4 Einrichten einer neuen Datenbankinstanz

Bevor man eine neue Datenbankinstanz einrichtet, muss man sich einige Dinge überlegen:

- Name der Datenbankinstanz
- Name und Passwort des Database-Manager-Operators (DBM)
- Name und Passwort des DB-System-Administrators (SYSDBA)
- Datenbankparameter (wie MAXUSERTASKS, MAXDATAPAGES, …)
- Ort und Größe der Data und Log Devspaces
- Backup-Strategie (Was? Wo? Wann? Wie?)

Der DBM-Benutzer ist ausschließlich für die Verwaltung der DB-Instanz mittels des Database Managers zuständig. Er kann also keine SQL-Sitzungen aufbauen. Dafür ist der SYSDBA-Benutzer zuständig, der neue DB-Benutzer anlegen darf und der Eigentümer von Systemtabellen ist. Bei der Einrichtung einer neuen Datenbankinstanz wird automatisch ein Benutzer DOMAIN eingerichtet, der das gleiche Passwort wie SYSDBA hat, und Eigentümer der Systemkatalogtabellen ist.

Wir wollen nun als Beispiel eine Datenbankinstanz mit dem Namen BIBLIO anlegen, die später das Bibliotheksbeispiel aus Kapitel 1 praktisch modellieren soll. Der DBM-Benutzer soll DBM (Passwort: dbm) und SYSDBA einfach nur DBA (Passwort: dba) heißen. Es soll zunächst nur ein Data und ein Log Devspace angelegt werden, mit den Größen 20 bzw. 8 MB. Wir starten also den Database Manager in der Kommandozeile und führen eine Reihe von Befehlen aus, die näher in [1] erläutert sind:

```
$ dbmcli –s db_create BIBLIO DBM,dbm
$ dbmcli –d BIBLIO –u DBM,dbm # Database Manager starten
dbmcli> param_startsession # Datenbankparameter initialisieren
dbmcli> param_init
dbmcli> param_put MAXUSERTASKS 5
dbmcli> param_checkall
dbmcli> param_commitsession # Devspaces anlegen
dbmcli> param_adddevspace 1 SYS SYS_001 F
dbmcli> param_adddevspace 1 LOG LOG_001 F 8000
dbmcli> param_adddevspace 1 DATA DAT_001 F 20000
dbmcli> db_cold
dbmcli> util_connect
dbmcli> util_execute INIT CONFIG
dbmcli> util_activate DBA,dba # SYSDBA anlegen
dbmcli> db_warm
dbmcli> load_systab -ud sapdb # Systemtabellen laden
dbmcli> sql_connect dba,dba # SQL-Befehl ausführen
dbmcli> sql_execute CREATE USER test PASSWORD test dba
        NOT_EXCLUSIVE
```

Mit dem letzten Befehl wurde ein Benutzer TEST mit dem Passwort *test* angelegt. Er gehört zur DBA-Benutzerklasse und hat somit die Rechte neue Ressourcen (Tabellen, etc.) und Benutzer anzulegen. Um mit der Datenbank arbeiten zu können, muss jetzt noch ein vollständiges Backup erstellt und ein Autolog-Backup[3] eingerichtet werden:

[3] Vgl. Abschnitt 6.2.3.1.

```
dbmcli> medium_put data ./datasave FILE DATA
dbmcli> medium_put auto ./autosave FILE AUTO
dbmcli> backup_start data # Daten-Backup durchführen
dbmcli> autlog_on auto # Automatisches Log-Backup aktivieren
```

Mittels xsql können wir nun eine SQL-Sitzung starten:

```
$ xsql -d BIBLIO -u TEST,test
```

Bemerkung: -d gibt den Datenbanknamen, -u den Benutzernamen und das Passwort an.

2.3 PostgreSQL

PostgreSQL 7.3.1 ist ein objekt-relationales Datenbanksystem, das in der Programmiersprache C implementiert ist und unter der BSD-Lizenz veröffentlicht wird. Objekt-relational bedeutet, dass es einige Erweiterungen wie Vererbung, benutzerdefinierte Datentypen, Funktionen und Operatoren anbietet. Dies ist ein Ansatz in Richtung objektorientierte Datenbanken. Im Folgenden sind die wichtigsten Features zusammengefasst:

- SQL/92- und teilweise SQL/99-Unterstützung mit vielen Erweiterungen
- Referenzielle Integrität und Constraints
- Multiversion Concurrency Control
- Isolationsgrade READ COMMITTED und SERIALIZABLE
- Große Anzahl an vordefinierten Datentypen und Funktionen
- Verschiedene Indexstrukturen wie B-Tree, Hash und R-Tree
- Sub-SELECTs
- Rules, Trigger und Stored Procedures
- Vererbung
- Benutzerdefinierte Typen, Operatoren und Funktionen
- Sequenzen
- Rechteverwaltung
- Temporäre Tabellen
- Online Backup
- Unterschiedliche Authentifizierungsverfahren (Host Based, Password, IDENT und Kerberos V4/V5)
- Verschiedene Programmierschnittstellen (ODBC, JDBC, Perl, Python, PHP, Tcl)
- Precompiler für C/C++

2.3.1 Architektur

Auch PostgreSQL kann mehrere Datenbanken verwalten, die als Gesamtheit Datenbank *Cluster* bezeichnet werden. Nach der Installation existieren zwei Datenbanken mit den Namen *template0* und *template1*. Diese werden als Vorlage für neue Datenbanken verwendet, wobei template0 eine Sicherheitskopie von template1 darstellt, die nicht verändert werden sollte. Das Anlegen einer neuen Datenbank stellt also in Wirklichkeit ein Kopieren einer Template-Datenbank dar.
Der Serverprozess von PostgreSQL heißt *postmaster*. Er wartet auf Verbindungen auf einen bestimmten TCP/IP-Port (5432) und erzeugt für jede Clientverbindung einen neuen Prozess mit dem Namen *postgres*. Das interaktive SQL-Tool *psql* erlaubt, SQL-Verbindungen aufzubauen und Wartungsarbeiten durchzuführen. Für Backup-Zwecke stehen die Tools *pg_dump*, *pg_dumpall* und *pg_restore* zur Verfügung. Sie ermöglichen ein Exportieren (Importieren) von Datenbanken in SQL und anderen Formaten.

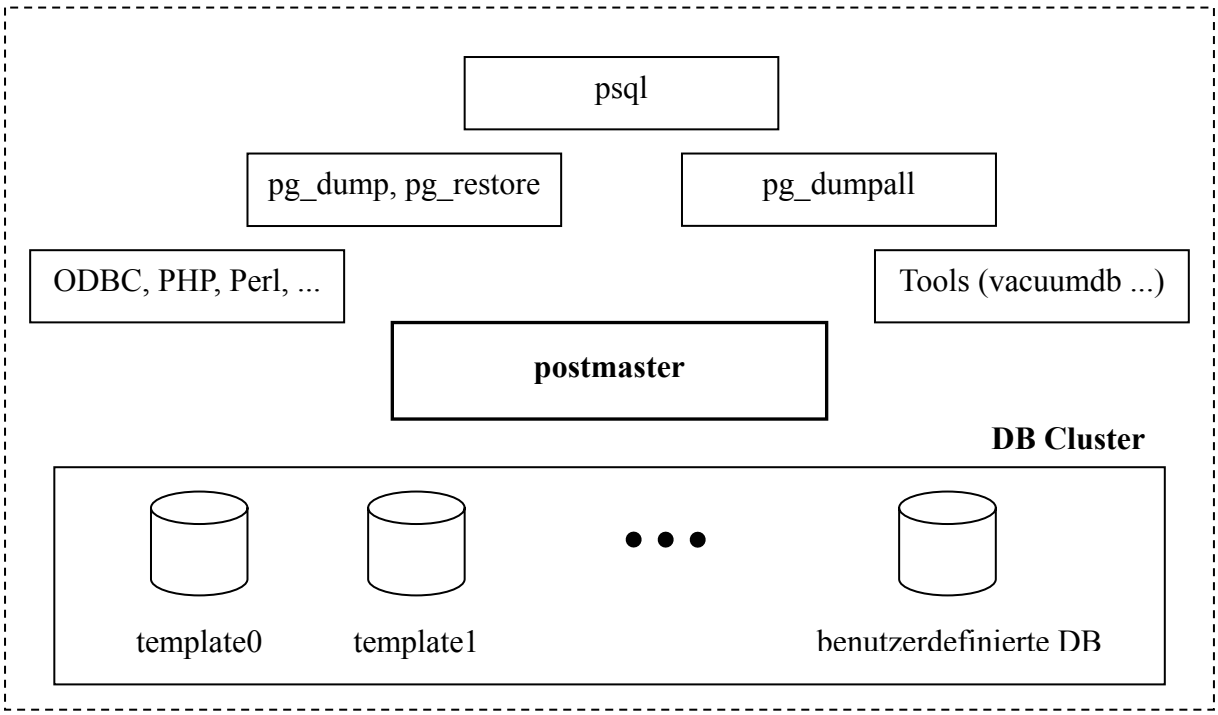

Abbildung 2.2 Architektur von PostgreSQL [2]

2.3.2 Installation

Der Source Code kann von der offiziellen PostgreSQL-Homepage http://www.postgresql.org heruntergeladen werden. Das Kompilieren des Source Codes ist dank GNU Autoconf sehr einfach. Vor der Installation muss ein Benutzer unter Linux eingerichtet werden, der normalerweise den Namen *postgres* hat. Ohne Angabe erfolgt die Installation in das Verzeichnis /usr/local/pgsql.

```
$ ./configure; gmake; gmake install # Source Code kompilieren
$ adduser postgres # Benutzer anlegen
$ mkdir /usr/local/pgsql/data # Datenverzeichnis anlegen
$ chown postgres /usr/local/pgsql/data # Eigentümer ändern
$ su – postgres # Benutzerkennung wechseln
```

Nach erfolgreicher Installation muss noch das Datenverzeichnis mit den Template-Datenbanken initialisiert werden:

```
$ /usr/local/pgsql/bin/initdb -D /usr/local/pgsql/data
$ ls # ein Blick in das Installationsverzeichnis
.  ..  bin  data  doc  include  lib  man  share
```

2.3.3 Starten des Servers

Aus Sicherheitsgründen empfehlen wir, den Server nicht als root, sondern unter dem Account *postgres* zu starten. Dies geschieht entweder über das Skript *pg_ctl* oder direkt mittels:

```
$ postmaster –iD /usr/local/pgsql/data > logfile 2>&1 &
```

Die Option *i* erlaubt externe TCP/IP-Verbindungen. Weiters wird das Datenverzeichnis angegeben und die Ausgabe samt stderr in die Datei *logfile* umgeleitet. Beim Starten wird die Konfigurationsdatei data/postgresql.conf gelesen.

2.3.4 Einrichten einer neuen Datenbank

Eine neue Datenbank kann entweder mit dem SQL-Befehl CREATE DATABASE oder mit dem Kommandozeilenbefehl *createdb* erzeugt werden. Ein Aufruf könnte wie folgt aussehen:

```
$ createdb –T template1 biblio
```

Dieser Befehl instanziert die Datenbank template1 und übernimmt alle Objekte und Daten, die darin gespeichert waren. Nun verbinden wir uns mit psql mit der neu angelegten Datenbank, um dort mit dem SQL-Befehl CREATE USER den Benutzer *test* anzulegen:

```
$ psql biblio postgres
biblio=# CREATE USER test CREATEUSER;
biblio=# \q # verlässt psql
```

Mit diesem Befehl wird ein Benutzer test erzeugt, der kein Passwort gesetzt hat und neue Benutzer anlegen darf.

2.4 MySQL

MySQL ist ebenso wie SAP DB unter der GNU GPL [11] lizensiert. Falls jemandem die sich daraus ergebenden Beschränkungen nicht zusagen, ist es auch möglich, eine kommerzielle Lizenz der Firma MySQL AB zu erwerben. MySQL ist die wohl bekannteste und verbreitetste Open-Source-Datenbank. Diese Stellung hat sie auf Grund ihrer Einfachheit und Stabilität inne, denn sie unterstützt von Haus aus keine Transaktionen, aber sehr wohl einen Sperrmechanismus, der Konsistenz unter Inkaufnahme von weniger Parallelität garantiert. MySQL wird in verschiedenen Versionen angeboten. Derzeit wird die Version 3.23 für den Produktionsbetrieb empfohlen. Im Rahmen der Diplomarbeit wurde aber bereits die neuere Version 4.0.3 beta MAX untersucht. Diese Version beinhaltet neben der MyISAM-Speicherorganisation auch noch zwei externe Datenbanksysteme, InnoDB und BerkeleyDB, die in MySQL integriert worden sind und mit deren Hilfe Transaktionen unterstützt werden. MySQL basiert auf den Programmiersprachen C und C++. Die wesentlichen Features sind:

- Unterstützung einer Untermenge von SQL/92 und SQL/99 mit eigenen Erweiterungen
- Verschiedene Tabellentypen (MyISAM, Hash, InnoDB, BerkeleyDB)
- Verschiedene Datentypen, unter anderem BLOB
- Query Cache
- Volltext-Indexierung und -Suche
- Rechteverwaltung
- Komprimierte Read-only-Tabellen
- Temporäre Tabellen
- Rudimentäre Replikationsunterstützung für Hot Standby
- Auch als Embedded Variante einsetzbar
- Verschiedene Programmierschnittstellen (ODBC, JDBC, Perl, Python, PHP, Tcl)

MySQL ist zwar einfach sowie für bestimmte Anwendungen effizient und schnell, auf der anderen Seite fehlen MySQL im Gegensatz zu SAP DB und PostgreSQL wichtige Features, worauf ebenfalls hingewiesen werden soll:

- Keine Constraints und referenzielle Integritätsüberprüfung (außer bei InnoDB)
- Keine Trigger und Stored Procedures
- Keine Views
- Keine Subquery-Ausdrücke
- Kein BOOLEAN-Datentyp
- Außer UNION keine Mengenoperationen auf Tabellen

2.4.1 Architektur

Auch MySQL basiert auf einer Clientserver-Architektur. Der Serverprozess von MySQL ist *mysqld*, der über UNIX- und TCP/IP-Sockets (Port: 3306) auf Clientverbindungen wartet. MySQL ist multi-threaded, das heißt für jede neue Verbindung wird unter UNIX ein neuer Prozess erzeugt. Es werden ODBC und auch alle gängigen Skriptsprachen unterstützt. Nach der Installation sind zwei Datenbanken vorhanden. Eine Testdatenbank, die leer ist, und eine Rechte-Datenbank (mysql), welche die Benutzer und die Zugriffsberechtigungen verwaltet. Diese speichert, welche Benutzer von welchen Rechnern aus auf welche Datenbanken zugreifen dürfen und welche Rechte sie auf den einzelnen Tabellen besitzen.

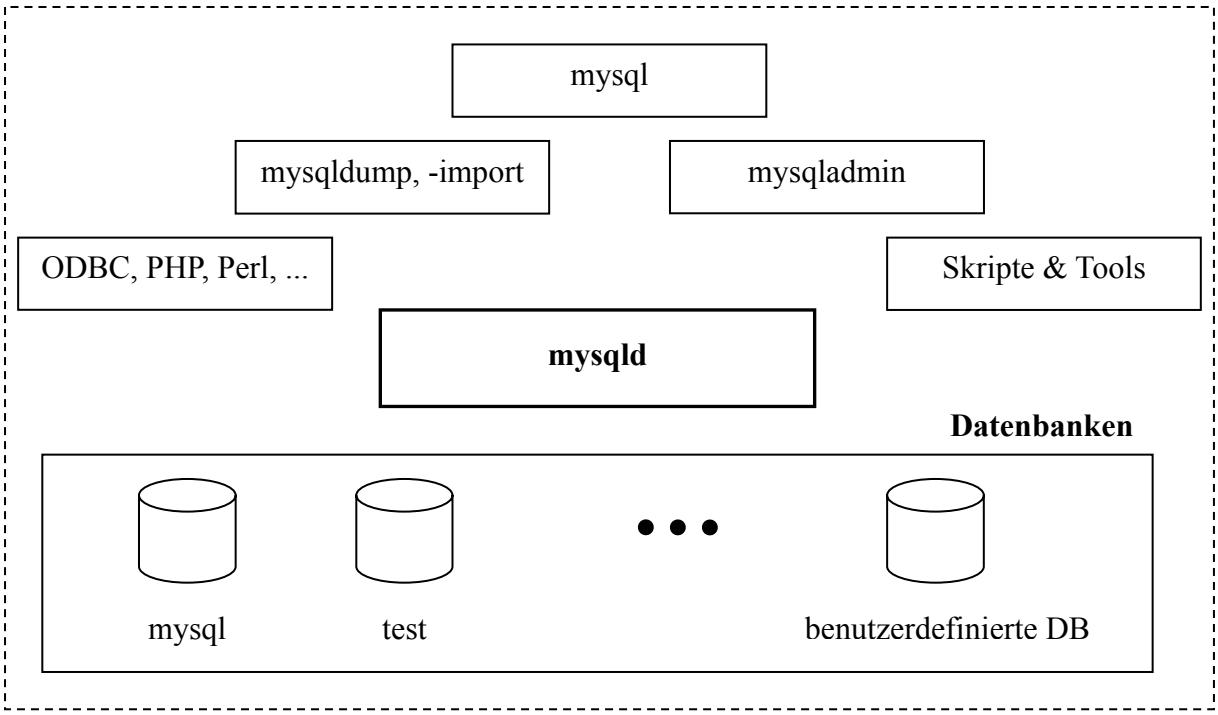

Abbildung 2.3 Architektur von MySQL [3]

MySQL verzichtet bewusst auf viele Features, die sonst schon fast Industriestandard geworden sind. Aber zumindest Transaktionen können durch die Integration der Embedded Datenbanken InnoDB und BerkeleyDB (BDB) angeboten werden. In dieser Hinsicht ist MySQL sogar sehr flexibel. Es kann nämlich für jede Tabelle entschieden werden, ob der Standardtyp MyISAM oder doch InnoDB oder BDB verwendet werden soll. Das abgespeckte MyISAM hat den Vorteil, dass es wesentlich schneller ist und dass die Tabellen weniger Speicherplatz benötigen. Jede MyISAM-Tabelle besteht aus drei Dateien. Die *.frm-Datei wird bei jedem Tabellentyp angelegt und enthält Schemainformationen, *.myd enthält die Daten und *.myi Indexinformationen, um schneller auf die Daten zuzugreifen.

Neben dem interaktiven SQL-Tool *mysql* gibt es noch eine Reihe von Perlskripten und Tools, mit denen man die Datenbanken verwalten kann. Um ein SQL-Level-Backup zu erzeugen, kann man zum Beispiel *mysqldump* verwenden. Mit *mysqladmin* kann man aus der Kommandozeile unter anderem neue Datenbanken anlegen, den Server herunterfahren oder einen Statusbericht ansehen.

2.4.2 Installation

Den Source Code, Binärdistributionen und RPM-Pakete bekommt man in vielen unterschiedlichen Varianten (einmal mit BerkeleyDB, einmal ohne) von der MySQL-Homepage http://www.mysql.com. Die Installation des Source Codes erweist sich als genauso einfach wie die von PostgreSQL. Im Folgenden wird angenommen, dass der Tarball mysql-max-4.0.3-beta-pc-linux-gnu-i686.tar.gz bereits heruntergeladen ist.

```
# Gruppe und Benutzer anlegen
$ groupadd mysql; useradd -g mysql mysql
$ ./configure –prefix=/usr/local/mysql; make; make install
$ scripts/mysql_install_db # DB mysql und test erzeugen
# Eigentümer der Dateien setzen
$ chown -R root  /usr/local/mysql
$ chown -R mysql /usr/local/mysql/var
$ chgrp -R mysql /usr/local/mysql
$ cp support-files/my-medium.cnf /etc/my.cnf
```

Die Installation erfolgt in das Verzeichnis /usr/local/mysql. Es wird wieder ein neuer Benutzer angelegt, unter dem später der Serverprozess laufen soll und dem die Datenbankdateien in mysql/var gehören. Zum Schluss wird noch die Standardkonfigurationsdatei nach /etc/my.cnf kopiert.

Sehen wir uns wieder die Verzeichnisstruktur nach der Installation an:

```
$ ls /usr/local/mysql
. .. bin include info lib libexec share sql-bench var
```

Die Datenbanken befinden sich im Verzeichnis *var*, der Serverprozess mysqld in *libexec* und die Skripte und Tools in *bin*.

2.4.3 Starten des Servers

Der Server kann mit dem Skript *mysqld_safe* gestartet werden:

```
$ /usr/local/mysql/bin/mysqld_safe --user=mysql &
```

2.4.4 Einrichten einer neuen Datenbank

Mit dem Aufruf von

```
$ mysqladmin create biblio
```

kann unsere Bibliotheksdatenbank erzeugt werden. Es wird im Datenverzeichnis mysql/var ein neues Verzeichnis mit dem Namen *biblio* angelegt. Den Benutzer *test* legen wir am Besten in einer SQL-Sitzung an. Der SQL-Befehl GRANT bewerkstelligt dies implizit:

```
$ mysql biblio
```

```
mysql> GRANT ALL PRIVILEGES ON biblio.* TO test@"*";
```

Bemerkung: test@"*" bedeutet, dass der Benutzer test von allen Rechnern aus vollständigen Zugriff auf alle Tabellen der Datenbank biblio bekommt.

Kapitel 3
Interne Speicherorganisation

Eine wichtige Aufgabe eines DBMS ist die persistente Speicherung der Daten auf einem Sekundär-Speichermedium. Man wünscht, dass das DBMS möglichst viele, unterschiedliche Daten speichern und diese möglichst effizient verarbeiten kann. Erst wenn beides zutrifft, wird das DBMS universell einsatzfähig.

In diesem Kapitel betrachten wir, welche Datentypen von den OSDB unterstützt werden und welche Indexstrukturen für die effiziente Nutzung der Daten zum Einsatz kommen. Die einfachste Suche in den Daten ist die sequenzielle Suche, die jedoch bei großen Datenmengen nicht praktikabel ist. Ein Index kann hingegen Abfragen wesentlich beschleunigen, da im Idealfall nur noch zwei Zugriffe notwendig sind, einer auf den Index und einer auf den Datensatz.

3.1 Datentypen

Jedes Attribut einer Tabelle hat einen Datentyp, der die Domain, die physikalische Repräsentation und die auf ihn anwendbaren Funktionen und Operatoren bestimmt. Die Datentypen der Attribute sind Teil des Tabellenschemas. Falls nicht ausdrücklich verboten, kann jedes Attribut eines konkreten Datensatzes auch unspezifiziert bleiben, das heißt es wird ihm „kein“ Wert zugewiesen. Dieser besondere Zustand wird in SQL mit dem Schlüsselwort NULL symbolisiert.

Die untersuchten OSDB geben an, sich an den SQL/92-Standard zu halten, der daher für die weitere Betrachtung als Referenz dient und folgende Datentypen definiert [12]:

Tabelle 3.1 SQL/92 Datentypen [12]

Datentyp	**Beschreibung**	**Beispiel**
CHARACTER(*n*) CHAR(*n*)	Zeichenkette fester Länge mit genau *n* Zeichen (*n*>0). Nicht verwendete Zeichen werden durch Leerzeichen aufgefüllt.	CHAR(10): 'Text '
CHARACTER VARYING(*n*) oder VARCHAR(*n*)	Zeichenkette variabler Länge mit bis zu *n* Zeichen (*n*>0)	VARCHAR(10): 'Text'
BIT(*n*)	Bitkette fester Länge mit genau *n* Bits (*n*>0)	BIT(4): B'1100'
BIT VARYING(*n*)	Bitkette variabler Länge mit bis zu *n* Bits (*n*>0)	BIT VARYING(16): B'11000001'
NUMERIC(*p*,*q*) DECIMAL(*p*,*q*)	Fixkommazahl mit Vorzeichen, die insgesamt *p* Ziffern lang ist und *q* dezimale Nachkommastellen speichert ($0 \leq q \leq p$, *p*>0)	NUMERIC(5,2): -123.45
INTEGER INT	Vorzeichenbehaftete ganze (Dezimal-)Zahl	INT: 12345
SMALLINT	Vorzeichenbehaftete ganze (Dezimal-)Zahl, jedoch mit kleinerem Wertebereich als INTEGER	SMALLINT: 28
FLOAT(*p*)	Gleitkommazahl mit *p* dezimalen Ziffern in der Mantisse	FLOAT(24): 1.123E+20
REAL bzw. DOUBLE PRECISION	REAL und DOUBLE PRECISION sind Abkürzungen für FLOAT(*p*), wobei *p* durch eine fixe implementierungsabhängige Zahl ersetzt wird.	REAL: 1.123E+20
DATE	Datumsangabe	DATE: '2003-01-30'
TIME [WITH TIME ZONE]	Zeitangabe, eventuell mit Zeitzone	TIME: '13:37:00'
TIMESTAMP [WITH TIME ZONE]	Kombinierte Zeit- und Datumsangabe, eventuell mit Zeitzone	TIMESTAMP: '2003-01-30 13:37:10.35'
INTERVAL	Zeitraum	INTERVAL YEAR: 3 years

Bitketten und Intervalle werden nur von PostgreSQL untersützt. Ansonsten finden sich diese Grundtypen in allen drei OSDB wieder.

3.1.1 SAP DB

SAP DB unterstützt neben den Grundtypen noch BOOLEAN und den Typ LONG für Binary Large Objects (BLOBs). BLOBs sind dazu gut, um gößere Binärdaten oder Texte in der Datenbank zu speichern, die nicht in ein VARCHAR-Feld passen würden. In SAP DB ist VARCHAR mit 8000 ASCII-Zeichen beschränkt. NUMERIC und FLOAT haben eine Genauigkeit von maximal 38 Stellen. Der Datentyp TIMESTAMP, nicht aber TIME, ermöglicht Mikrosekunden abzuspeichern. Eine Zeitzonenangabe wird nicht unterstützt.

3.1.2 PostgreSQL

PostgreSQL bietet den mit Abstand größten Umfang an Datentypen und ermöglicht auch das Erstellen von benutzerdefinierten Datentypen inklusive den dazugehörigen Operatoren. An Stelle von FLOAT(*p*) werden nur die Datentypen REAL und DOUBLE PRECISION mit fixem *p* angeboten. Dafür unterstützt NUMERIC eine Genauigkeit von bis zu 1000 Stellen. Das Rechnen mit diesen Zahlen ist aber verglichen mit den Gleitkommazahlen sehr langsam, da nicht auf die Hardware-Floatingpoint-Einheit zurückgegriffen werden kann.
In PostgreSQL kann man die Anzahl der Nachkommastellen bei den Sekunden (maximale Genauigkeit ist wieder eine Mikrosekunde) für Attribute vom Typ TIME und TIMESTAMP festlegen. Zeittypen mit Zeitzonenangabe sind auch möglich.
In der folgenden Tabelle sind die Erweiterungen von PostgreSQL zusammengefasst:

Tabelle 3.2 Zusätzliche Datentypen in PostgreSQL [2]

Datentyp	**Beschreibung**	**Beispiel**
BIGINT	Integerzahl mit 8 Byte	3000000000000
MONEY	Fixkommazahlen mit zwei Nachkommastellen. Die Verwendung dieses Typs wird nicht mehr empfohlen. Stattdessen soll NUMERIC verwendet werden.	’$1,000.00’
TEXT	Zeichenkette variabler Länge, für BLOBs geeignet	
BYTEA	Binärkette variabler Länge, für BLOBs geeignet	
BOOLEAN	Boolscher Wahrheitstyp	TRUE
POINT	Zweidimensionaler Punkt, Format (x,y)	’(5,3)’
LINE	Linie, die durch die zwei angegebenen Punkte geht (noch nicht vollständig implementiert)	’((0,0),(1,1))’
LSEG	Liniensegment, spezifiziert durch Anfangs- und Endpunkt	’((1,2),(10,2))’
BOX	Rechteck	’((5,5),(0,0))’
PATH	Offener oder geschlossener Weg	’((0,0),(1,0), (1,1),(2,2))’

POLYGON	Geschlossene Polygonzüge	’((0,0),(1,0), (1,1))’
CIRCLE	Kreis mit Mittelpunkt und Radius	’<(0,0),1>’
INET bzw. CIDR	IPv4-Internetadresse mit Netzwerkmaske, bei CIDR müssen die Bits des Subnetzes 0 sein	212.186.106.148 192.168.0.0/25

PostgreSQL bietet auch die Möglichkeit, mehrdimensionale Arrays aus bereits existierenden Datentypen zu definieren. Obwohl man die Dimension in der Definition angeben kann, verwendet PostgreSQL in der aktuellen Version nur Arrays variabler Länge. Hier ist ein Beispiel für ein Array, das mehrere Untertitel zu einem Buchtitel speichern könnte:

```
Titel VARCHAR(200)[]
```

Der Titel könnte dann in Titel[1] und die Untertiteln in Titel[2], Titel[3] usw. stehen.

3.1.3 MySQL

Die obere Grenze für CHAR und VARCHAR ist 255. Dafür gibt es in MySQL eine Reihe von BLOB-Typen, die sich in der maximal zulässigen Länge unterscheiden. Alle Zahlentypen können auch vom Typ UNSIGNED sein, das heißt vorzeichenlos. Damit ergibt sich in manchen Fällen ein vergrößerter positiver Wertebereich. FLOAT wird intern entweder als 4 oder 8 Byte lange Gleitkommazahl repräsentiert. NUMERIC wird als String abgespeichert und kann maximal 255 Stellen, davon maximal 30 Nachkommastellen besitzen.
Der Typ TIMESTAMP hat in MySQL eine besondere Eigenschaft. Wenn man einen neuen Datensatz anlegt oder einen bestehenden verändert und den Wert des TIMESTAMP-Attributes gar nicht oder (später) auf NULL setzt, wird die aktuelle Systemzeit in das Attribut geschrieben. Der Datentyp DATETIME ist mit TIMESTAMP verwandt, hat jedoch die erwähnte Eigenschaft nicht. Es gibt weder Datentypen mit Zeitzonenangabe noch mit einer Genauigkeit von weniger als einer Sekunde.

Die Erweiterungen von MySQL sind in folgender Tabelle dargestellt:

Tabelle 3.3 Zusätzliche Datentypen in MySQL [3]

Datentyp	**Beschreibung**	**Beispiel**
TINYINT	Zahl mit 1 Byte	10
SMALLINT [4]	Zahl mit 2 Byte	1000
MEDIUMINT	Zahl mit 3 Byte	100000
INT [4]	Zahl mit 4 Byte	10000000

[4] INT und SMALLINT gehören zum SQL/92-Standard und werden hier des Vergleiches wegen angegeben.

BIGINT	Zahl mit 8 Byte	10000000000
YEAR	Jahreszahl	2003
TINYTEXT TINYBLOB	BLOB mit maximal 255 Bytes	
TEXT oder BLOB	BLOB mit bis zu 64 KB (minus einem Byte)	
MEDIUMTEXT MEDIUMBLOB	BLOB mit bis zu 16 MB (minus einem Byte)	
LONGBLOB	BLOB mit bis zu 4GB (minus einem Byte)	
ENUM ('Wert1','Wert2',...)	Aufzählungstyp mit bis zu 65535 verschiedenen Werten	'Wert3'
SET ('Wert1','Wert2',...)	Mengentyp mit bis zu 64 Elementen	'Wert1,Wert3'

3.2 Datensätze und Blöcke

Datensätze werden aus Performance-Gründen häufig in Blöcken fixer Größe gespeichert. Wie wir gesehen haben, gibt es Datentypen variabler Länge. Daraus folgt, dass auch die Größe der Datensätze variabel sein kann. Das bringt natürlich einige Probleme mit sich. Zum Beispiel kann ein Datensatz im Laufe der Zeit kleiner oder größer werden, wofür Platz vorgesehen werden muss. Wenn der Platz in einem Block nicht mehr ausreicht, muss ein neuer Block erstellt werden, der womöglich am Ende der Datei angehängt wird. Das Löschen von Datensätzen erzeugt oft freie Bereiche, die erst später oder gar nicht mehr aufgefüllt werden. Dieses Phänomen nennt man Datenfragmentierung in Datenbanken.

In PostgreSQL wird für jedes Update ein Delta-Eintrag erstellt und gelöschte Datensätze werden nicht sofort entfernt. Um nicht mehr benötigte Einträge in der Datendatei zu entfernen, muss regelmäßig die so genannte *Vacuum-Routine* aufgerufen werden. In MySQL werden zwar keine Delta-Einträge erstellt, aber es kann zu Datenfragmentierung kommen, falls Datensätze variabler Länge verwendet werden. Hier kann mit dem Befehl OPTIMIZE defragmentiert und freier Speicherplatz auf Grund von gelöschten Datensätzen zurückgewonnen werden. SAP DB vermeidet durch automatische Reorganisation beim Einfügen und Löschen das Problem der Fragmentierung.

Wir unterscheiden, ob Datensätze bzw. Blöcke nach dem Einfügen immer eine statische Position in den Datendateien einnehmen, also sequenziell geschrieben werden, oder während der Laufzeit auch verschoben werden können, also dynamisch sind. MySQL und PostgreSQL

verwenden eine statische, SAP DB eine bedingt durch den B*-Baum[5] dynamische Datenorganisation. Das im Folgenden vorgestellte Konzept der Indexstrukturen ermöglicht trotz sequenzieller Speicherorganisation effiziente Zugriffe auf die gespeicherten Daten.

3.3 Indexstrukturen

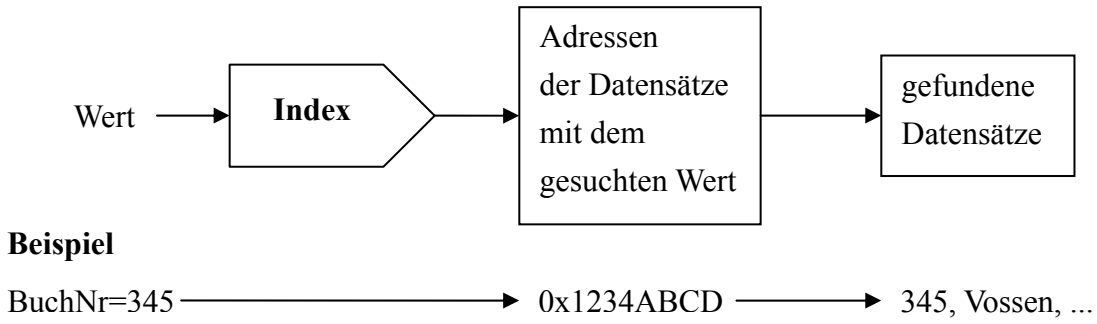

Abbildung 3.1 Schematische Darstellung der Funktionsweise von Indexstrukturen [6]

Ein Index ist eine Datenstruktur, die das Suchen in einer Tabelle nach bestimmten Attributen beschleunigen soll. Der einfachste Index besteht aus einer Tabelle mit zwei Attributen. Das erste Attribut enthält konkrete Werte aus einer Tabellenspalte, die indexiert werden soll, zum Beispiel die Buchnummern (BuchNr) aus der Tabelle Buch. Das zweite Attribut des Index gibt die Adresse an, mit der auf den Datensatz zugegriffen werden kann. Oft werden in DBMS mehrere Datensätze in einem Block fixer Größe untergebracht. Dann wird als Adresse die Blocknummer gespeichert. Wesentlich ist dabei, dass der Index nach dem Attribut, das Suchschlüssel genannt wird, sortiert ist. Die Suche nach einem Buch mit einer bestimmten Buchnummer kann dann mittels binärer Suche in ld *n* Schritten durchgeführt werden, wobei *n* die Anzahl der Einträge im Index bezeichnet. Ein weiterer Vorteil ist, dass der Index wesentlich weniger Speicherplatz als die indexierte Tabelle benötigt, da nicht der gesamte Datensatz, sondern lediglich der Suchschlüssel und die Adresse gespeichert werden.

Einen Index auf den Primärschlüssel einer Tabelle nennen wir Primärindex, sonst Sekundärindex. Besteht der Suchschlüssel aus mehreren Attributen, werden diese in der Regel zu einem Attribut zusammengesetzt und basierend auf diesem zusammengesetzten Attribut wird ein eindimensionaler Index kreiert. Wird beispielsweise ein Index auf die Attribute Autor und Titel (in dieser Reihenfolge!) erstellt, kann der Index für eine Suche nach dem Titel alleine

[5] Vgl. Abschnitt 3.3.1.1 und Data Management Using B* Trees in [1].

nicht herangezogen werden, da der Titel im Suchschlüssel als Postfix vorkommt, und die Liste nicht nach diesem Teil sortiert ist. Die Suche nur nach dem Autor ist aber möglich.
Ein mehrdimensionaler Index hingegen bezeichnet eine Datenstruktur, die zum Beispiel x- und y-Koordinaten speichert, und Abfragen der Form „Suche mir alle Punkte, die nicht weiter als ε von einem bestimmten Punkt entfernt sind“ vorsieht. Ein mehrdimensionaler Index unterstützt auch das Suchen in einem beliebigen Unterraum.

Ein Suchschlüssel eines Sekundärindex ist nicht notwendigerweise eindeutig. In diesem Fall speichert der Index oft eine ganze Liste der Blöcke, in denen ein bestimmtes Suchwort vorkommt.

3.3.1 Spezielle Indexstrukturen

Die einfache Indexstruktur aus der Einleitung, nämlich die sortierte Tabelle mit den zwei Spalten Suchschlüssel und Adresse, hat den Nachteil, dass eine etwaige Reorganisation beim Einfügen und Löschen von Einträgen aus der indexierten Tabelle aufwändig ist, da zum Beispiel beim Einfügen in den Index Platz geschaffen werden muss, indem alle nachfolgenden Einträge nach hinten verschoben werden. Das ist im Grunde genau das gleiche Problem, das man hätte, wenn man die Daten selbst sortiert abspeichern würde. Natürlich ist das Sortieren eines solchen Index einfacher, da er in der Regel kleiner ist als die gesamte Tabelle, aber es geht noch besser und zwar mit Bäumen oder Hashtabellen als Index.

3.3.1.1 B-Baum

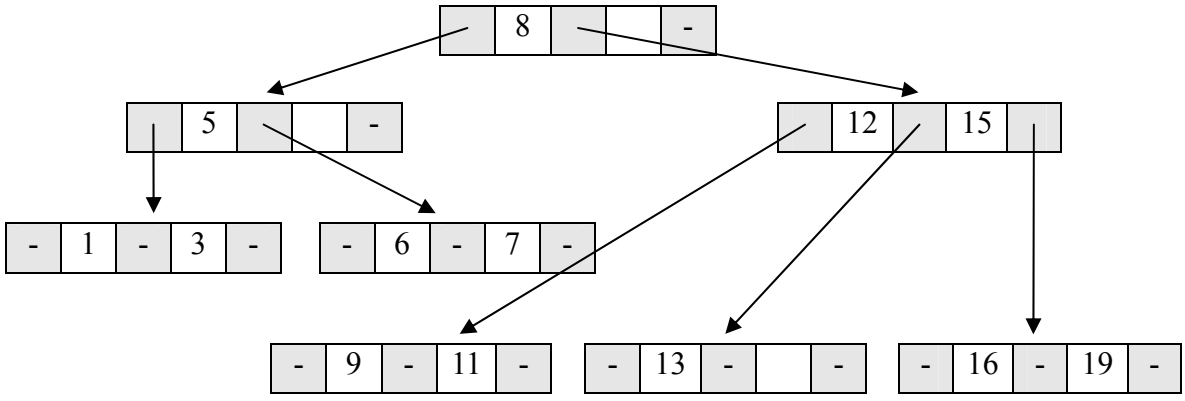

Abbildung 3.2 B-Baum der Höhe 2 (Nicht-Schlüsselanteil nicht dargestellt) [4]

Der B-Baum (kurz für Balancierter-Baum bzw. Bayer-Baum nach seinem Autor) ist ein gerichteter Baum, in welchem jeder Pfad von der Wurzel zu einem Blatt die gleiche Länge hat. Einen solchen Baum nennt man ausbalanciert. Jeder Knoten des Baumes besteht aus

einem Block konstanter Länge, der n Datensätze[6] aufnimmt und n+1 Zeiger auf seine Söhne besitzt. Hier liegt der entscheidende Vorteil, denn die Knoten müssen physikalisch nicht zusammenhängen, können also beispielsweise sequenziell angelegt werden.

Ein B-Baum erfüllt aber noch weitere Bedingungen [4]:

- Jeder Knoten (Block) enthält mindestens k sowie höchstens $2k(=n)$ Datensätze[7] für eine fest vorgegebene natürliche Zahl k. In Abbildung 3.2 ist k = 1.
- Jeder Datensatz besteht aus einem Schlüssel- (beispielsweise dem Suchschlüssel) und einem Nicht-Schlüsselanteil (beispielsweise die Adresse auf den dazugehörigen Datensatz in der ursprünglichen Tabelle).
- Jeder Block ist nach aufsteigenden Schlüsselwerten sortiert, und jeder Vaterknoten stellt einen Index für seine Söhne dar. Dies bedeutet, dass ein Zeiger an Position i im Vaterknoten auf einen Unterbaum verweist, dessen Wertebereich durch die benachbarten Suchschlüssel i–1 und i+1 (sofern vorhanden) beschränkt ist.
- Die Wurzel hat keinen Sohn oder mindestens zwei Söhne.
- Jeder innere Knoten, welcher also weder Wurzel noch Blatt ist, hat die für ihn maximal mögliche Anzahl von Söhnen, das heißt er hat n+1 Söhne, falls er n Schlüsselwerte enthält.

Durchsucht wird der Baum wie folgt: Ausgehend von der Wurzel wird zunächst der Knoten nach dem Schlüssel durchsucht. Enthält der Knoten den Schlüssel nicht, wird entweder bei jenem Sohn rekursiv weitergesucht, dessen benachbarte Schlüsselwerte den gesuchten Schüssel eingrenzen oder ganz links respektive ganz rechts weitergesucht, falls der gesuchte Schlüssel kleiner respektive größer ist als alle anderen Schlüsselwerte des Knoten.

Die Operationen Löschen und Einfügen dürfen die oben angegebenen Bedingungen nicht verletzen. Es müssen unter Umständen Knoten zusammengelegt und neue eingefügt werden. Diese Reorganisation kann sich bis zur Wurzel fortsetzen. Für die genaue Vorgehensweise wird auf die diesbezügliche Literatur verwiesen [4,6].

Von einem B*-Baum sprechen wir, falls die Daten (= Nicht-Schlüsselanteil) nur in den Blättern gespeichert werden. In diesem Fall muss immer bis zu den Blättern gesucht werden.

---

[6] $n=2k$ mit $k>0$

[7] Die untere Schranke gilt nicht für die Wurzel.

3.3.1.2 Hashindex

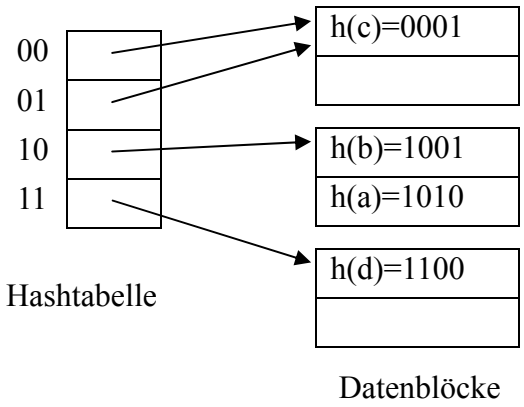

Abbildung 3.3 Dynamische Hashtabelle, Blockgröße=2, Datensätze={a,b,c,d}, 2 Bits von *h* verwendet [6]

Die Idee von Hashtabellen ist es, eine (nicht eindeutige) Abbildung, Hashfunktion *h* genannt, von Schlüsseln auf Blockadressen zu verwenden, und die Datensätze in dem von der Funktion *h* errechneten Block abzuspeichern. Das Suchen nach einem bestimmten Datensatz funktioniert wie folgt: Zuerst wird der Block berechnet, in dem sich der Datensatz befinden müsste. Anschließend wird in diesem Block sequenziell gesucht. Wichtig ist, dass die Hashfunktion die Daten möglichst gleichmäßig auf die Blöcke verteilt, damit beim Suchen im Durchschnitt nur auf einen Block zugegriffen werden muss. Falls ein Block überläuft, müssen so genannte Überlaufsblöcke angehängt werden. Dies kann ein Anzeichen dafür sein, dass die Größe der Hashtabelle zu klein ist. Bei statischem Hashing ist die Größe der Hashtabelle konstant, bei dynamischem Hashing wird sie abhängig vom Auslastungsgrad dynamisch angepasst.

Dynamische Hashtabellen verwenden nur einen Teil des errechneten Wertes der Hashfunktion *h* für die Bestimmung des Blocks. Am Anfang nur das erste Bit (= 2 Einträge), dann die ersten beiden (= 4 Einträge) usw. Die Größe der Hashtabelle nimmt dann immer mit einer 2-er Potenz zu. Zu einer solchen Verdoppelung kommt es bei jedem Überlauf eines Blocks, solange bis der neue Datensatz in einen Block passt. Überlaufende Blöcke müssen aufgeteilt und die Datensätze in den richtigen Block verschoben werden, abhängig von dem identifizierenden Teil des Ergebnisses der Hashfunktion.
Bei linearen Hashtabellen wird die Verdoppelung der Blöcke verlangsamt, indem ein durchschnittlicher Füllungsgrad der Blöcke angestrebt wird und – falls nötig – Überlaufblöcke angelegt werden, bevor eine Verdoppelung stattfindet.

3.3.1.3 R-Baum

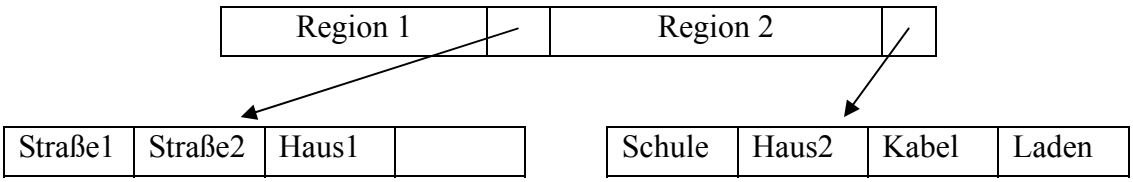

Abbildung 3.4 R-Baum (zweidimensional) [6]

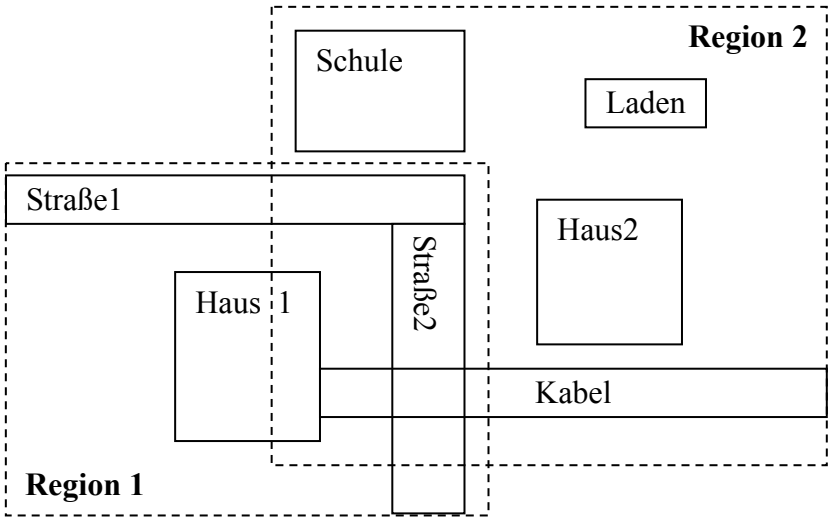

Abbildung 3.5 Regionen des R-Baumes aus Abbildung 3.4 [6]

R-Bäume[8] sind eine Generalisierung von B-Bäumen für mehrdimensionale Abfragen. Wenn wir uns einen Knoten eines B-Baumes ansehen, teilt dieser den Schlüsselbereich eindimensional in disjunkte Teilbereiche auf. Hat ein Knoten beispielsweise die Schlüssel 5, 10 und 20, dann befinden sich im ersten Sohn alle Schlüssel kleiner als 5, im zweiten Sohn alle zwischen 10 und 20 und im dritten alle größer als 20.

R-Bäume speichern nicht einzelne Punkte, sondern mehrdimensionale Objekte ab. Im zweidimensionalen Fall könnten das zum Beispiel die Koordinaten von Gebäuden in einer Stadt sein. Eine Abfrage könnte dann lauten: Welche Gebäude befinden sich in meiner Nähe?
Ein R-Baum verwendet dazu mehrdimensionale Schlüssel wie zum Beispiel Rechtecke, die geografisch als Regionen interpretiert werden können. Die Söhne teilen die Region entweder weiter auf, oder, falls es sich um Blätter handelt, beinhalten sie die Objekte, die von der Region eingeschlossen werden.

[8] Das R steht für Region.

Die Regionen müssen nicht disjunkt sein, sollten aber eine möglichst kleine Überlappung aufweisen. Beim Einfügen und Löschen werden Regionen vergrößert oder verkleinert und gegebenenfalls aufgeteilt bzw. zusammengelegt. Das Suchen erfolgt ähnlich wie in einem B-Baum. Es müssen aber womöglich mehrere Söhne durchsucht werden, falls die Wahl auf Grund von Überlappungen nicht eindeutig ist.

3.3.2 Mögliche Anwendungen von Indexstrukturen

Indexstrukturen können bestimmte Abfragen beschleunigen. Falls kein Index verwendet werden kann, ist das DBMS gezwungen, eine sequenzielle Suche zu starten, das heißt alle Datensätze der Reihe nach zu lesen. Das kann sehr zeitaufwändig sein.

Typische Anwendungsfälle von B-Bäumen sind:

- **Exakte Suche (Punktabfrage)**
  Beispiel: Finde den Datensatz mit der BuchNr = 345.
- **Bereichsabfrage**
  Beispiel: Zeige alle Bücher von A bis K.
- **Sortierte Ausgabe von Tabellen**
  Beispiel: Sortiere die Tabelle Leser nach dem Attribut Name.
- **Minimum- und Maximum-Berechnung**
  Entspricht der Ausgabe des ersten und letzten Elementes einer sortierten Tabelle.
- **Verbund von Tabellen**
  Bei einem Verbund müssen übereinstimmende Datensätze gefunden werden. Mit Hilfe eines Index auf die verknüpften Attribute einer Tabelle, kann die zu verknüpfende Tabelle sequenziell durchgegangen werden und für den Verbund im Index nachgesehen werden.

Hashindexe können hingegen nur für die exakte Suche und insbesondere für Verbundoperationen eingesetzt werden. R-Bäume können für mehrdimensionale Bereichsabfragen verwendet werden. Beispielsweise ermöglichen sie die Suche nach allen Objekten, die sich in einem angegebenen Unterraum befinden. In Abschnitt 4.3 werden die möglichen Anwendungen von Indexstrukturen im Kontext von SQL beleuchtet.

3.3.3 Vergleich der OSDB

3.3.3.1 SAP DB

SAP DB verwendet als Speicherorganisation ausschließlich B*-Bäume mit einer Blockgröße von 8 KB. Da in B*-Bäumen die Daten in den Blättern gespeichert werden, brauchen die Zwischenknoten nicht den gesamten Schlüssel, sondern nur einen für die Unterscheidung der Söhne relevanten Teil davon abzuspeichern. Dies macht sich SAP DB zu nutze, um Speicherplatz zu sparen. In einem Blattknoten befinden sich je nach Datensatzlänge unterschiedlich viele Datensätze, die unsortiert der Reihe nach hinzugefügt werden. Um die Sortierung innerhalb eines Blattes zu beschleunigen, wird im Blattknoten eine sortierte Liste, auf die Datensätze geführt. Ein Verändern oder Löschen von Datensätzen verschiebt gegebenenfalls andere Datensätze im Block, um Fragmentierung zu vermeiden. Diese Vorzüge bezahlt man mit der in B-Bäumen laufenden Reorganisation, die bei Einfüge-, Änderungs- oder Löschoperationen notwendig werden kann. Dafür ist aber keine explizite Reorganisation (wie zum Beispiel das Entfernen von Löschlücken) notwendig, die den laufenden Betrieb sperren könnte.
Attribute vom Typ LONG werden in eigenen B*-Bäumen gespeichert, um sie von den anderen Daten zu trennen. Sekundärindexe verwenden ebenfalls B*-Bäume, speichern aber in den Blättern nur Adressen auf die Daten.

3.3.3.2 PostgreSQL

PostgreSQL speichert die Tabellendaten in einer sequenziellen blockorientierten Datei ab, wo jede Änderung eines Datensatzes einen Delta-Eintrag erzeugt und auch das Löschen von Datensätzen nicht sofort durchgeführt wird. Das hat den Vorteil, dass im Laufe der Zeit verschiedene Versionen der Datensätze vorhanden sind, die in der Transaktionsverarbeitung eine Anwendung finden. Dazu später mehr. Auf der anderen Seite wird es vor allem bei Tabellen, die häufig geändert werden, notwendig, periodisch eine Reorganisation durchzuführen, die unnötige Einträge löscht und Speicherplatz zurückgewinnt. Diese Operation wird in PostgreSQL als Vacuum-Routine bezeichnet, die nur bei Angabe der Option FULL die betroffene Tabelle sperrt. Der Befehl CLUSTER ermöglicht es, die Tabellendatei physikalisch nach dem Primärschlüssel zu sortieren. BLOB-Attribute werden in eigenen Tabellen ausgelagert und automatisch komprimiert.

Folgende Indextypen mit Angabe der verwendeten Algorithmen stehen zur Verfügung:

- B-Bäume [13]
- R-Bäume [14]

- Hashtabellen [15]
- Generalized Search Tree (GiST)[9]

GiST ist eine Generalisierung von Suchbäumen, die es ermöglicht, eigene Datentypen mit einem dazupassenden Indexzugriff zu definieren. Für PostgreSQL gibt es in diese Richtung bereits erste Implementierungen, die zum Beispiel Volltextsuche in Textattributen ermöglichen.
PostgreSQL erzeugt beim Anlegen einer neuen Tabelle automatisch einen Index auf den Primärschlüssel und weitere Indexe für jeden Sekundärschlüssel. Diese Indexe sind vom Typ B-Baum und werden für die Einhaltung der Eindeutigkeit verwendet, das heißt sie lassen keinen Eintrag mit einem Schlüssel, der bereits existiert, zu.

Eine Besonderheit von PostgreSQL sind partielle Indexe, die nur einen Teil einer Tabelle indexieren. Welche Datensätze indexiert werden, gibt man bei der Indexerzeugung mit einem SQL-WHERE-Prädikat an. Die Motivation von partiellen Indexen liegt darin, nur den interessanten Teil einer Tabelle zu indexieren. Angenommen wir haben in unserem Bibliotheksbeispiel hauptsächlich Bücher des Verlages „My Press“ und wir wollen auf die Spalte Verlag einen Index anlegen. Dann macht es Sinn, Bücher, die von „My Press“ herausgeben wurden, nicht im Index aufzunehmen, da in diesem Fall eine sequenzielle Suche vorzuziehen ist.

PostgreSQL bietet auch die Möglichkeit, indirekte Indexe basierend auf Funktionsergebnissen, die als Argumente Attributwerte übergeben bekommen, zu definieren. Es können auch die Vergleichsoperatoren, die die Ordnung des Index festlegen, durch benutzerdefnierte Funktionen umkonfiguriert werden.

3.3.3.3 MySQL

MySQL unterscheidet verschiedene Tabellentypen:

- ISAM
- MyISAM
- Heap
- Merge
- BerkeleyDB (BDB)
- InnoDB

[9] Weitere Informationen über GiST befinden sich unter http://gist.cs.berkeley.edu:8000/gist/.

ISAM ist die Vorgängerversion von MyISAM. In beiden werden die Daten sequenziell in einer Datei abgelegt und B-Bäume als Indextyp verwendet. Der Tabellentyp Merge bildet die Relationenvereinigung von mehreren, in Bezug auf das Schema identischen, bereits existierenden MyISAM-Tabellen. Heap-Tabellen sind temporäre Tabellen, die sich nur im Hauptspeicher befinden und einen Hashindex verwenden. BDB und InnoDB sind Embedded Datenbanksysteme, die in MySQL integriert worden sind und Transaktionsverarbeitung unterstützen. Im Folgenden soll auf den Standardtabellentyp MyISAM näher eingegangen werden.

MyISAM-Tabellen gibt es in drei Formaten:

- **Statisch**: Die Datensätze haben eine fixe Länge.
- **Dynamisch**: Die Datensätze haben eine variable Länge.
- **Komprimiert**: Die Datensätze werden komprimiert gespeichert. Die Tabelle kann nur gelesen werden.

Ob die Tabelle statisch oder dynamisch ist, hängt davon ab, ob BLOB- oder VARCHAR-Datentypen in der Tabellendefinition vorkommen. Dynamische Tabellen haben den Nachteil, dass sie fragmentieren. Dagegen hilft nur ein regelmäßiger Aufruf von OPTIMIZE in mysql bzw. von myisamchk in der Kommandozeile. Der Zugriff auf die betroffenen Tabellen wird für die Dauer der Operation blockiert. Statische und dynamische Tabellen können in MySQL komprimiert werden. Dann sind aber nur noch Lesezugriffe erlaubt.

MySQL bietet als besonderes Feature, Texte, die zum Beispiel in Attributen vom Typ TEXT gespeichert sind, zu indexieren, und ermöglicht dadurch eine Volltextsuche nach Wörtern. Dadurch lässt sich ohne viel Aufwand beispielsweise eine Stichwortsuche im Buchtitel realisieren, die mit einem normalen Index nicht möglich wäre.

Kapitel 4
Abfrageverarbeitung

In den vorigen Kapiteln war des Öfteren von SQL[10] die Rede, die Abfragesprache in Datenbanksystemen schlechthin. In diesem Kapitel gehen wir deshalb auf den SQL-Standard und die Erweiterungen der OSDB ein. Mit SQL können Daten nicht nur abgefragt, sondern auch definiert, eingefügt und manipuliert werden. Auch viele andere Features sind über diese Sprache zugänglich. Nachdem wir dem Leser einen Überblick über SQL gegeben haben, betrachten wir das Konzept der Abfrageverarbeitung von Datenbanksystemen und sehen uns die Optimierungsmöglichkeiten der OSDB an. Dazu zählen verschiedene Suchstrategien und Verbundalgorithmen, die von zuvor definierten Indexstrukturen profitieren können.

4.1 Die Abfragesprache SQL

Bevor wir näher auf die eigentliche Abfrageverarbeitung eingehen werden, wollen wir die relationale Sprache betrachten, die von allen gängigen Datenbanksystemen verwendet wird. Die Entwicklung von SQL (Structured Query Language) begann in den 1970-er Jahren bei IBM, bis sie erstmals 1982 zum ANSI-Standard erklärt wurde. Für die untersuchten OSDB ist der ISO/ANSI-SQL/92-Standard von Bedeutung, denn es geben alle drei an, diesen zu unterstützen. Neuere Entwicklungen sind im SQL/99-Standard aufgenommen worden, der aber unter den OSDB noch nicht vollständig umgesetzt wurde.

SQL basiert auf dem in Kapitel 1 vorgestellten Relationen-Tupel-Kalkül. Aus praktischen Gründen kann SQL aber mehr als RTK. In SQL kann man neben Abfragen auch Schemata definieren, Tupel einfügen, löschen und modifizieren, arithmetische Ausdrücke berechnen,

[10] Die offizielle Aussprache ist [es kju: el]. Historisch bedingt wird es manchmal auch wie das englische Wort „sequel“ ausgesprochen, da eine Vorgängerversion SEQUEL (Structured English Query Language) hieß [12].

Tupeln gruppieren, Aggregatfunktionen wie eine Summenberechnung durchführen und in manchen Fällen auch prozedural programmieren. Eine umfassende Darstellung von SQL würde den Rahmen dieser Arbeit bei weitem sprengen. Deshalb versuchen, wir die wichtigsten Konzepte anhand von Beispielen zu präsentieren und verweisen auf die einschlägige Literatur [12].

4.1.1 Datendefinition

Wir wollen das Bibliotheksbeispiel aus Kapitel 1 nun in SQL umsetzen und um die Tabellen *Rueckgabe* und *Mahnung* erweitern. Zur Erinnerung hier die Schemata der Tabellen:

Buch(BuchNr, Autor, Titel, Verlag, Erscheinungsjahr)
Leser(LeserNr, Name)
Ausleihe(BuchNr, LeserNr, Entlehndat ,faelligam)
Rueckgabe(BuchNr, Entlehndat, Rueckgabedat)
Mahnung(BuchNr, Entlehndat, Spesen)

Die unterstrichenen Attribute bilden den Primärschlüssel. Weiters sollen folgende Inklusionsabhängigkeiten zwischen den Tabellen bestehen:

Ausleihe[BuchNr] $\subseteq$ Buch[BuchNr], Ausleihe[LeserNr] $\subseteq$ Leser[LeserNr],
Rueckgabe[BuchNr, Entlehndat] $\subseteq$ Ausleihe[BuchNr, Entlehndat],
Mahnung[BuchNr, Entlehndat] $\subseteq$ Ausleihe[BuchNr, Entlehndat]

Beispiel 4.1 CREATE TABLE

Mit dem Befehl CREATE TABLE können in SQL neue Tabellen angelegt werden. Auszugsweise zeigen wir als Beispiel das Anlegen der Tabellen Buch, Ausleihe und Mahnung. Dazu müssen in einer durch Komma getrennten Liste der Attributname, der Datentyp und eventuelle Integritätsbedingungen angegeben werden.

```
CREATE TABLE Buch (
     BuchNr INTEGER PRIMARY KEY,
     Autor VARCHAR(50),
     Titel VARCHAR(100) NOT NULL,
     Verlag VARCHAR(100),
     Erscheinungsjahr SMALLINT
);
```

```
CREATE TABLE Ausleihe (
     BuchNr INTEGER,
     LeserNr INTEGER,
     Entlehndat DATE,
     faelligam DATE,
PRIMARY KEY (BuchNr,Entlehndat),
FOREIGN KEY (BuchNr) REFERENCES Buch ON DELETE RESTRICT,
FOREIGN KEY (LeserNr) REFERENCES Leser ON DELETE RESTRICT
);

CREATE TABLE Mahnung (
     BuchNr INTEGER,
     Entlehndat DATE,
     Spesen NUMERIC (10,2) NOT NULL,
PRIMARY KEY (BuchNr,Entlehndat),
FOREIGN KEY (BuchNr,Entlehndat) REFERENCES Ausleihe
ON DELETE RESTRICT
);
```

Primärschlüssel werden mit dem Schlüsselwort PRIMARY KEY, Sekundärschlüssel mit UNIQUE definiert. Inklusionsbeziehungen sind in SQL durch FOREIGN KEY … REFERENCES-Konstrukte repräsentiert. ON DELETE RESTRICT bewirkt, dass das Löschen von referenzierten Datensätzen verhindert wird. Alternativ könnte man angeben, dass referenzierte Datensätze automatisch gelöscht werden. Das nennt man kaskadiertes Löschen (ON DELETE CASCADE). Die Bedingung NOT NULL verlangt, dass beim Einfügen oder Verändern von Datensätzen das dazugehörige Feld einen konkreten Wert zugewiesen bekommt. Im Beispiel soll also der Buchtitel immer gesetzt werden. Zu beachten ist, dass durch NOT NULL Leerstrings jedoch erlaubt sind. In MySQL haben Fremdschlüssel-Bedingungen nur in Tabellen vom Typ InnoDB eine Wirkung, ansonsten werden sie einfach überlesen.

Beispiel 4.2 CREATE INDEX

```
CREATE INDEX idx_buchtitel ON Buch (Titel);
```

Um die Suche nach dem Buchtitel zu beschleunigen, können wir einen Index auf die Spalte Titel anlegen. CREATE INDEX benötigt dazu einen Indexnamen (idx_buchtitel), den Tabellennamen (Buch) und die Spalten, die indexiert werden sollen (Titel).

Beispiel 4.3 CREATE VIEW

```
CREATE VIEW BuchKurz AS SELECT BuchNr, Titel FROM Buch;
```

Manchmal ist es sinnvoll, die Sicht auf (Basis-)Tabellen einzuschränken oder aus den Ergebnistabellen beliebiger Abfragen virtuelle Tabellen zu erstellen. Solche virtuellen Tabellen werden mit CREATE VIEW definiert und entstehen aus der angegebenen SELECT-Anweisung. Das Beispiel erstellt eine virtuelle Tabelle BuchKurz, die aus den Attributen BuchNr und Titel besteht. Die Sicht BuchKurz kann in SELECT-Anweisungen in der FROM-Klausel genauso wie eine Basistabelle verwendet werden (siehe Abschnitt 4.1.3 und 7.20).

Das Löschen von Tabellen, Indexen und anderen Objekten ist in SQL einheitlich durch einen Befehl der Form DROP TABLE *Tabellenname*, DROP INDEX *Indexname*, DROP VIEW *Sichtname* etc. möglich. Schemainformationen können bis zu einem gewissen Grad auch noch nachträglich verändert werden. Einzelheiten finden sich dazu in Abschnitt 7.22 Modifikation des Datenbankschemas.

4.1.2 Datenmanipulation

Unter Datenmanipulation fallen die Operationen Einfügen, Modifizieren und Löschen von Datensätzen aus der Datenbank. In der Literatur werden auch Abfragen als Datenmanipulation aufgefasst. Wir widmen diesem Thema aber einen eigenen Abschnitt. Auch hier wollen wir wieder Beispiele für derartige Operationen angeben.

Beispiel 4.4 INSERT

```
INSERT INTO Leser (LeserNr, Name) VALUES (1,'Claudia');
```

Falls keine Integritätsbedingung verletzt wird, wird der Datensatz in die Tabelle Leser mit den Werten LeserNr = 1 und Name = „Claudia“ eingetragen.

Beispiel 4.5 UPDATE

```
UPDATE Leser SET Name='Claudia H.' WHERE LeserNr=1;
```

Mit dem Befehl UPDATE *Tabellenname* SET *Attribut = neuer Wert*, … WHERE *Bedingung* können gleich mehrere Datensätze auf einmal modifiziert werden. Welche Datensätze von der

Zuweisung betroffen sind, hängt von der WHERE-Bedingung ab. Es werden nur jene Datensätze modifiziert, auf die die Bedingung zutrifft.

Beispiel 4.6 DELETE FROM

DELETE FROM *Tabellenname* WHERE *Bedingung* löscht alle Datensätze aus der angegebenen Tabelle, welche die Bedingung erfüllen.

```
DELETE FROM Buch WHERE BuchNr = 100; -- Buch 100 löschen
```

4.1.3 SELECT-Abfragen

Der wohl wichtigste Befehl von SQL ist die SELECT-Anweisung, mit der Projektionen, Selektionen, Verbunde und weitere Operationen der relationalen Algebra[11] auf Basistabellen ausgeführt werden können. Die Syntax sieht wie folgt aus:

SELECT [ALL | DISTINCT] { * | *Ausdruck*$_1$ [AS *Alias*$_1$] [, …, *Ausdruck*$_n$ [AS *Alias*$_n$]] }
FROM *Tabellenname*$_1$ [*Tabellenalias*$_1$] [, …, *Tabellenname*$_n$ [*Tabellenalias*$_n$]]
[WHERE *Bedingung*]
[GROUP BY *Attributliste* [HAVING *Bedingung*]]
[{UNION | INTERSECT | EXCEPT} SELECT …]
[ORDER BY *Attribut_oder_Alias*$_1$ [ASC|DSC] [, …, *Attribut_oder_Alias*$_n$ [ASC|DSC]]] ;

Ausdruck$_1$ bis Ausdruck$_n$ geben an, welche Spalten aus den angegebenen Tabellen projiziert werden. Der Stern * steht für alle Spalten der Tabelle. Es ist möglich, Aliasbezeichnungen für die Ergebnisspalten und die Tabellen anzugeben. Die WHERE-Bedingung erlaubt, Datensätze zu filtern und mit ORDER BY nach bestimmten Attributen aufsteigend (ASC) oder absteigend (DSC) zu sortieren. Einfache Bedingungen können mit den boolschen Operatoren AND, OR und NOT zu komplexeren zusammengefasst werden. UNION, INTERSECT und EXCEPT bilden die mengentheoretischen Operatoren Vereinigung, Durchschnitt und Differenz nach. Mit dem Schlüsselwort DISTINCT werden keine Duplikate im Ergebnis angezeigt. Mit der GROUP BY-Klausel können die Datensätze partitioniert und für jede Gruppe ein aggregierter Wert berechnet werden, wobei zwei Datensätze genau dann in derselben Gruppe sind, falls die Attributwerte der angegebenen Attributliste bei beiden übereinstimmen. HAVING ermöglicht – ähnlich wie WHERE – aggregierte Datensätze auf Gruppenebene zu filtern.

Im Folgenden sollen einige Beispiele in SQL formuliert werden, um die Anwendungsmöglichkeiten der einzelnen Konstrukte zu verdeutlichen:

[11] Vgl. Definition 1.9.

Beispiel 4.7 Einfache Abfrage

```
SELECT * FROM Buch WHERE BuchNr > 100;
```

Diese Anweisung selektiert alle Bücher mit sämtlichen Attributen, die eine BuchNr größer als 100 haben.

Beispiel 4.8 Alias und ORDER BY

```
SELECT Name AS Leser FROM Leser ORDER BY Leser;
```

Dieses Beispiel selektiert alle Lesernamen in aufsteigender Reihenfolge.

Beispiel 4.9 Verbund

```
SELECT DISTINCT Titel, Name FROM Buch, Leser, Ausleihe
WHERE Buch.BuchNr = Ausleihe.BuchNr AND
Leser.LeserNr = Ausleihe.LeserNr;
```

Dieser Befehl führt einen Verbund der Tabellen Buch, Leser und Ausleihe aus und zeigt an, welche Bücher von welchen Lesern entlehnt werden bzw. wurden. Würde man die WHERE-Bedingung weglassen, würde stattdessen das kartesische Produkt gebildet werden.

Beispiel 4.10 Aggregatfunktionen

Mit Aggregatfunktionen kann eine ganze Spalte einer Tabelle oder einer Gruppe, die durch GROUP BY gebildet wurde, zu einem aggregierten Wert zusammengefasst werden. Die Aggregatfunktionen von SQL/92 sind COUNT, SUM (Summe), MAX, MIN und AVG (Durchschnitt). COUNT zählt die Anzahl der Werte in einer Spalte ungleich NULL, der Spezialfall COUNT(*) die Anzahl der Datensätze.

```
SELECT COUNT (DISTINCT BuchNr) FROM Ausleihe;
```

Das Beispiel zählt, wie viele unterschiedliche Bücher entlehnt wurden.

Beispiel 4.11 GROUP BY

Wir wollen eine Abfrage formulieren, die uns für jeden Leser ausgibt, wie oft er gemahnt wurde. Ein erster Versuch könnte folgende SELECT-Abfrage hervorbringen:

```
SELECT Name, COUNT(M.BuchNr) AS Mahnungen
FROM Mahnung M, Ausleihe A, Leser L
WHERE M.BuchNr = A.BuchNr AND  M.Entlehndat = A.Entlehndat
AND L.LeserNr = A.LeserNr
GROUP BY L.LeserNr, Name;
```

Zu beachten ist, dass Attribute, die nicht in der GROUP BY-Klausel vorkommen, nur aggregiert ausgegeben werden können. Daher ist das Attribut Name ebenfalls in die GROUP By-Klausel aufgenommen worden, obwohl es für die Unterscheidung der Partitionen nichts mehr beiträgt. Der (innere) Verbund, so wie er hier zum Einsatz kommt, hat den Nachteil, dass Leser, die noch nie gemahnt wurden, in der Ergebnisrelation nicht aufscheinen. Daher kennt SQL den so genannten äußeren Verbund zwischen zwei Tabellen, der solche „verlorenen" Datensätze, die sich mit der jeweils anderen Tabelle nicht verknüpfen lassen, entweder aus der linken, rechten oder aus beiden Tabellen dennoch mit ins Ergebnis aufnimmt. Die Attribute der jeweils anderen Tabelle werden dann allerdings mit NULL-Marken aufgefüllt. Syntaktisch sieht das in SQL so aus:

```
SELECT Name, COUNT(Mahnung.BuchNr) AS Mahnungen
FROM (Ausleihe INNER JOIN Mahnung USING (BuchNr,Entlehndat))
RIGHT OUTER JOIN Leser USING (LeserNr)
GROUP BY Leser.LeserNr, Name;
```

Hier werden die notwendigen Verbunde bereits in der FROM-Klausel mit dem Schlüsselwort JOIN gebildet. In USING wird angegeben, welche Spalte(n) den Verbund steuern soll(en). Ein rechter äußerer Verbund bewirkt, dass Datensätze aus der rechten Tabelle (im Beispiel Leser) auf alle Fälle in der Ergebnisrelation aufscheinen sollen. Alternativen zum rechten Verbund sind der LEFT- und der FULL-JOIN.
In SAP DB sind derartige JOIN-Ketten mit mehr als zwei Tabellen nicht möglich. Stattdessen muss man mit dem aus Oracle bekannten Join-Operator (+) jene Attribute in der WHERE-Bedingung kennzeichnen, die bei einem gescheiterten Vergleich mit NULL aufzufüllen wären. Die korrekte Abfrage lautet wie folgt:

```
SELECT Name, COUNT(M.BuchNr) AS Mahnungen
FROM Mahnung M, Ausleihe A, Leser L
WHERE M.BuchNr = A.BuchNr  AND  M.Entlehndat = A.Entlehndat
AND L.LeserNr = A.LeserNr (+)
GROUP BY L.LeserNr, Name
```

Beispiel 4.12 Arithmetische Ausdrücke

```
SELECT *, Spesen * 13.7603 AS ATS, Spesen * 1.95583 AS DM
FROM Mahnung;
```

SQL erlaubt im Kopf und in den Bedingungsklauseln arithmetische Ausdrücke. Das Beispiel rechnet die Mahnspesen von Euro in österreichische Schillinge und Deutsche Mark um.

Beispiel 4.13 Verschachtelte SELECT-Anweisungen

Im SQL-Kopf ($Argument_1$ bis $Argument_n$), in der FROM-Klausel oder in den Bedingungsklauseln (WHERE) ist es möglich, verschachtelte SELECT-Anweisungen zu verwenden. Die verschachtelte SELECT-Anweisung muss allerdings entweder immer nur genau einen Wert zurückliefern, um sie mit einem anderen Attribut vergleichen oder im Kopf verwenden zu können, oder man verwendet in Bedingungen ein Mengenprädikat wie EXISTS, MATCH, IN, ALL, ANY oder UNIQUE. EXISTS beispielsweise ist genau dann wahr, falls die verschachtelte SELECT-Anweisung mindestens einen Datensatz enthält. Erläuterungen zu den anderen Prädikaten finden sich in [12]. Das folgende Beispiel selektiert alle Leser, die noch nie eine Mahnung erhalten haben:

```
SELECT Name FROM Leser WHERE LeserNr NOT IN
(SELECT LeserNr FROM Mahnung NATURAL JOIN Ausleihe);
```

Abfragen mit GROUP BY-Klauseln können immer zu verschachtelten SELECT-Anweisungen ohne GROUP BY umgeformt werden. Das Beispiel 4.11 lässt sich wie folgt umformulieren:

```
SELECT Name, (SELECT COUNT(Mahnung.BuchNr) FROM Mahnung
JOIN Ausleihe USING (BuchNr,Entlehndat)
WHERE Ausleihe.LeserNr = Leser.LeserNr) AS Mahnungen
FROM Leser;
```

Verschachtelte SELECT-Anweisungen sind in MySQL gar nicht, in SAP DB nur in FROM- und Bedingungsklauseln möglich.

Beispiel 4.14 Vordefinierte Variablen

Der SQL-Standard kennt einige vordefinierte Variablen wie zum Beispiel CURRENT_DATE, in der das aktuelle Systemdatum gespeichert ist. Damit lässt sich eine Abfrage realisieren, die alle Bücher auflistet, welche länger als 4 Wochen entlehnt, aber noch nicht zurückgegeben wurden:

```
SELECT Titel, Name FROM Ausleihe, Buch, Leser
WHERE Ausleihe.BuchNr = Buch.BuchNr
AND Ausleihe.LeserNr = Leser.LeserNr
AND CURRENT_DATE - Ausleihe.EntlehnDat > 28
AND (Ausleihe.BuchNr, Ausleihe.Entlehndat)
NOT IN (SELECT BuchNr, Entlehndat FROM Rueckgabe);
```

4.2 Abfrageverarbeitung

In Abbildung 4.1 ist der Weg einer SQL-Abfrage bis zur Ausführung konzeptionell dargestellt. In der ersten Stufe übersetzt ein Parser die Abfrage in eine interne Darstellung. In diesem Schritt wird die Abfrage auf ihre Gültigkeit hin überprüft und im Systemkatalog nachgesehen, ob die referenzierten Objekte (Tabellen, Views, Funktionen, etc.) existieren.
Das Rewrite-System führt alle notwendigen Ersetzungen in der Abfrage durch. So werden beispielsweise Views durch ihre Definition ersetzt und in PostgreSQL eventuell definierte Ersetzungsregeln (Rules) angewendet.
In der nächsten Ausführungsstufe wird ein konkreter Zugriffsplan (Query Execution Plan) erstellt, der vom Executor abgearbeitet wird. In dieser Phase findet auch die Optimierung statt, das heißt die Auswahl der Zugriffsstrategie und eine eventuelle Vereinfachung der Abfrage. Die Optimierung erfolgt entweder anhand von festen Regeln oder einer Kostenabschätzung. Dabei werden verschiedene Zugriffspläne erstellt und miteinander verglichen, wobei der Plan mit den geringsten Kosten dem Executor übergeben werden soll.

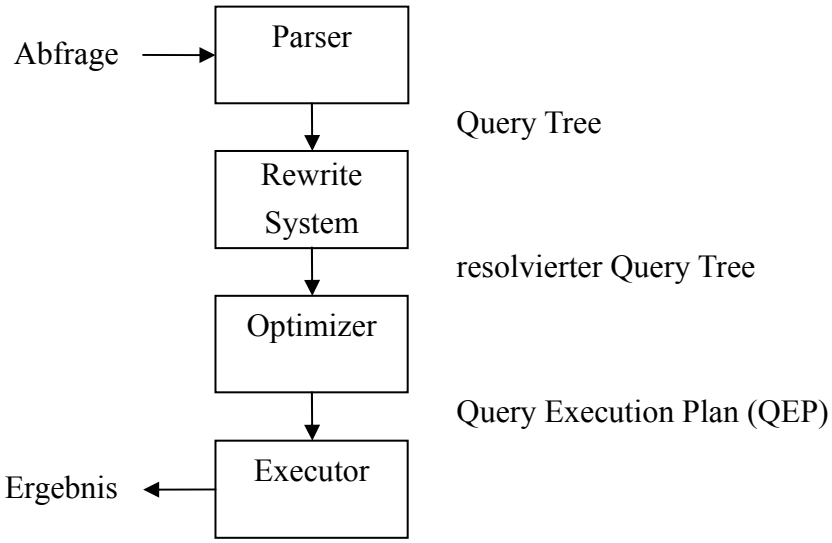

Abbildung 4.1 Ablauf einer Abfrageverarbeitung [4]

Eine Kostenfunktion schätzt für einen gegebenen Zugriffsplan Parameter wie die Wartezeit für den Benutzer, I/O-Zeit (Zugriffe auf Blöcke bzw. Datensätze), benötigte CPU-Zeit oder den Speicherverbrauch ab. Um eine effiziente Kostenabschätzung durchzuführen, speichert das DBMS Statistikwerte für jede Tabelle ab. Solche Statistikwerte umfassen beispielsweise die Anzahl der Datensätze, Anzahl der Blöcke und die Verteilung der Schlüsselwerte einer Tabelle. Die Statistikwerte müssen in allen drei untersuchten OSDB manuell aktualisiert werden. In SAP DB lautet der zugehörige Befehl UPDATE STATISTICS, in PostgreSQL ANALYZE und in MySQL ANALYZE TABLE. Die beiden ersteren Befehle können sowohl auf Tabellenebene als auch auf Datenbankebene (= alle Tabellen) die Statistik aktualisieren.

4.3 Optimierungsansätze

Die drei OSDB wollen im Wesentlichen I/O-Zugriffe, CPU-Zeit und Speicherplatz einsparen. Die Maxime besteht darin, so wenig Datensätze bzw. Blöcke wie möglich von dem Sekundärspeicher zu lesen und – falls möglich – keine temporären Tabellen zu erzeugen. Alle drei OSDB verwenden einen Cache, um Blöcke im Hauptspeicher zwischenzulagern.
Über eine High-Level-Optimierung, das ist die logische Umformung von Abfragen in Äquivalente mit weniger Kosten, finden sich in den Manuals und der Literatur kaum Hinweise. SAP DB kann beispielsweise Suchbedingungen in die konjunktive Normalform bringen, scheitert aber an den DeMorgan'schen Regeln. PostgreSQL formt die Bedingungen in die disjunktive Normalform um [18] und versucht auch alle NOTs so weit wie möglich, nach innen zu bringen. MySQL macht dies nicht, kann aber Bedingungen mit Konstanten vereinfachen.

Beispiel 4.15 Constant Folding und Constant Removal in MySQL [3]

a<b AND b=c AND a=5 $\rightarrow$ b>5 AND b=c AND a=5
(B>=5 AND B=5) OR (B=6 AND 5=5) OR (B=7 AND 5=6) $\rightarrow$ B=5 OR B=6

Unter Low-Level-Optimierung verstehen wir die Wahl einer effizienten Such- und Verbundstrategie unter Zuhilfenahme eines oder mehrerer Indexe. Der Optimizer muss also entscheiden, ob und wenn ja, welche Indexe für eine Abfrage in Frage kommen.

4.3.1 Suchstrategien

Das Suchen ist neben dem Verbund, der letztendlich auch eine Suche nach gemeinsamen Paaren darstellt, eine der aufwändigsten Operationen in SQL. Deshalb wollen wir uns ansehen, wie die OSDB dabei vorgehen.
Die einfachste und zugleich aufwändigste Suche in einer Tabelle ist die sequenzielle Suche, die immer dann zum Einsatz kommt, falls der Optimizer keine bessere Möglichkeit findet. Wenn wir zum Beispiel das Buch mit der Nummer 100 suchen, wäre es zu aufwändig, die ganze Tabelle sequenziell zu durchsuchen. Stattdessen wird der Optimizer festlegen, den Index auf den Primärschlüssel BuchNr zu verwenden und direkt auf den korrespondierenden Datensatz zuzugreifen. Ein Index muss aber nicht immer schneller als eine sequenzielle Suche sein und zwar genau dann, wenn die Bedingung auf einen großen Teil der Tabelle zutrifft. In diesem Fall würde es nämlich auf Grund des permanenten Zugriffs auf die Datensätze bzw. Blöcke der Tabelle zu sehr vielen Plattenkopfbewegungen kommen, was in Summe langsamer wäre, als die Tabelle als Ganzes sequenziell durchzugehen. Man nennt das Verhältnis Anzahl der gefundenen Datensätze zur Gesamtanzahl an Datensätzen in der Tabelle Selektivität der

Suchbedingung. Die Selektivität einer Punkt- oder Bereichsabfrage kann vom Optimizer aber lediglich abgeschätzt werden.
Ein aus mehreren Spalten zusammengesetzter Index kann nur dann verwendet werden, falls nach einem Präfix des Index gesucht wird. Falls man nur Attribute selektiert, die im Index vorkommen, verzichten die OSDB sogar darauf, auf den Datensatz zuzugreifen, und beschränken den Zugriff nur auf den Index.
Die OSDB unterscheiden sich in der Indexausnutzung erst dann, wenn mehrere Bedingungen mittels OR oder AND verknüpft werden. Zum einen kommt es zu der besagten Normalformbildung, zum anderen stellt sich die Frage, falls mehrere Indexe möglich wären, welche verwendet werden. Prinzipiell werden die Indexe mit höherer Selektivität verwendet, das heißt bei einer Punktabfrage der Primärindex vor einem Sekundärindex. SAP DB kann auch mehrere Indexe gleichzeitig verwenden und je nach AND oder OR den Durchschnitt oder die Vereinigung bilden. PostgreSQL macht das nur im Spezialfall *Primärschlüssel = Wert* OR *Indexattribut = Wert*, während MySQL überhaupt nur einen Index pro Tabelle verwenden kann. Nur in MySQL kann man explizit angeben, welcher Index verwendet werden soll.

Beispiel 4.16 Anzeige der Suchstrategie mittels EXPLAIN

```
EXPLAIN SELECT * FROM Buch WHERE Verlag LIKE 'VEB%' AND
Erscheinungsjahr=1989
```

Mit dem Befehl EXPLAIN gefolgt von der SQL-Abfrage kann man sich anzeigen lassen, welche Suchstrategie und welche Indexe verwendet werden. Im Beispiel suchen wir nach allen Büchern, die 1989 unter einem VEB-Verlag erschienen sind. Wir nehmen an, dass für die Spalten Verlag und Erscheinungsjahr je ein Index existiert.

Ergebnis in SAP DB (gekürzt)

```
Table Index       Strategy                              Page Count
BUCH  VERLAG      INTERSECTION OF COLUMN INDEXES           3
      ERSCHEINUNGSJAHR
                  RESULT IS NOT COPIED , COSTVALUE IS      1
```

Ergebnis in PostgreSQL

```
Index Scan using idx_verlag on Buch  (cost=0.00..5.97 rows=1
width=90)
```

Ergebnis in MySQL (gekürzt)

```
 table | type | key       | key_len | ref   | rows | Extra
 Buch  | ref  | idx_jahr |       3 | const |    6 | where used
```

SAP DB verwendet den Durchschnitt der beiden Indexe, während PostgreSQL und MySQL abwechselnd jeweils nur einen der Indexe verwenden. EXPLAIN zeigt in allen drei Fällen auch die Kostenabschätzung an.

4.3.2 Verbundstrategien

Ähnlich wie bei der Suche kann der Verbund zwischen zwei oder mehreren Tabellen mit Indexen wesentlich beschleunigt werden. Ein Index kann nämlich anstatt einer sequenziellen Suche zur Auffindung der korrespondierenden Datensätze verwendet werden. Wesentlich zur Ausführungsgeschwindigkeit trägt auch die Reihenfolge der Tabellen bei, die dem Verbundoperator zugeführt werden. Deshalb versuchen die drei OSDB die Tabellen so zu reihen, dass möglichst kleine Tabellen mit einer hohen Selektivität (= möglichst wenige assoziierte Datensätze) zuerst an die Reihe kommen, um den Suchraum einzuschränken. In MySQL kann mit dem Schlüsselwort STRAIGHT_JOIN die gleiche Reihenfolge, wie in der FROM-Klausel angegeben, erzwungen werden. In PostgreSQL kann die Reihenfolge der Verbundpaare mittels verschachtelter JOIN-Befehle festgelegt werden.

Folgende Verbundstrategien werden von den OSDB verwendet:

- **Nested-Loop-Join (PostgreSQL, MySQL)**
  Für jeden Datensatz aus der rechten Tabelle werden alle Datensätze aus der linken Tabelle durchlaufen, um die assoziierten Datensätze zu finden. Der Aufwand ist in dieser Variante quadratisch.
- **Hash-Join (PostgreSQL, MySQL)**
  Anstatt wie beim Nested-Loop-Join alle Datensätze der linken Tabelle zu durchlaufen, wird auf die assoziierten Datensätze direkt mittels eines Hashindex, der gegebenenfalls vorher erstellt werden muss (PostgreSQL), zugegriffen.
- **Sort-Join (SAP DB)**
  Laut Manual [1] sortiert SAP DB die linke Tabelle, bevor sie mit der rechten Tabelle verknüpft wird. Vermutlich wird die Sortierung verwendet, um schneller korrespondierende Paare zu finden. Falls Indexe existieren, werden jedenfalls diese verwendet.
- **Merge-Sort-Join (PostgreSQL)**
  Zuerst werden beide Tabellen nach den Verbundattributen sortiert. Anschließend wird mit einem Merge-Algorithmus der Verbund berechnet, der die Sortierung ausnutzen kann. Beide Tabellen brauchen in der Merge-Phase nur einmal durchlaufen werden. Das Sortieren erübrigt sich, falls Indexe auf den Verbundattributen existieren.

Mit den obigen Algorithmen werden immer genau zwei Tabellen miteinander verbunden. Um mehr als zwei Tabellen zu verbinden, werden die Verfahren sukzessive angewendet. Gestartet wird mit zwei Tabellen, dann wird das Ergebnis mit der dritten verknüpft, usw. Etwas unterschiedlich geht dabei MySQL vor:

- **One-Sweep-Multi-Join (MySQL)**
  Es werden alle Tabellen in einem Durchgang verbunden. Dazu wird für jeden Datensatz der ersten Tabelle ein assoziierter Datensatz in der zweiten Tabelle gesucht, dann in der dritten, usw. Mittels Backtracking werden alle möglichen Kombinationen errechnet. Natürlich verwendet MySQL auch Indexe für das schnelle Suchen nach geeigneten Datensätzen.

4.3.3 Sonstige Optimierungen

Folgende Operationen in SQL profitieren auch von Indexen:

- **Mininmum- und Maximum-Berechnung**
  SAP DB und MySQL verwenden einen vorhandenen Index, um den Wert zu berechnen.
- **Sortieren mit ORDER BY**
  PostgreSQL verwendet den Index, um die Datensätze sortiert auszugeben. In MySQL wird nur dann der Index verwendet, falls dieser ausreicht, um die projizierten Attribute auszugeben. SAP DB verwendet, anders als im Manual beschrieben, außer für den Primärschlüssel den Index nicht.
- **Gruppieren mit GROUP BY**
  Nur PostgreSQL verwendet einen allfälligen Index, um die Gruppierung rascher vorzunehmen.
- **Entfernung von Duplikaten mit DISTINCT**
  PostgreSQL und MySQL verwenden – falls möglich – den Index, um die Duplikate zu filtern.
- **Zählen von Datensätzen mit Count(Spalte)**
  SAP DB und MySQL zählen nur im Index, während PostgreSQL die gesamte Tabelle durchläuft. Count(*) ist in SAP DB und MySQL für die gesamte Tabelle aus der Tabellenstatistik sofort abrufbar.

Kapitel 5
Transaktionsverarbeitung

Eine Transaktion ist eine logische Arbeitseinheit, bestehend aus einem oder mehreren SQL-Befehlen, die konsistent ausgeführt werden sollen. Das heißt, dass sie sich nicht von parallel laufenden Datenmanipulationen beeinflussen lassen und entweder vollständig oder gar nicht ausgeführt werden. Ein Beispiel dafür ist das Erhöhen der Mahnspesen aller noch nicht zurückgegebenen Bücher um einen fixen Tagessatz. Ein Zugriff auf die veränderten Datensätze soll erst möglich sein, nachdem die Transaktion ordnungsgemäß abgeschlossen wurde.

In diesem Kapitel gehen wir auf die Probleme ein, die sich bei gleichzeitigen Zugriffen auf die Datenbank ergeben können. Wir definieren, welche Eigenschaften die Ausführungsreihenfolge der Befehle erfüllen muss, um die Konsistenz nicht zu verletzen und ein beliebiges Abbrechen von Transaktionen zu ermöglichen. Unter Konsistenz verstehen wir hier einerseits, dass sich die in der Datenbank gespeicherten Daten jederzeit in einem konsistenten Zustand befinden (vgl. Definition 1.8) und andererseits, dass es während der Abarbeitung einer Transaktion nicht dazu kommt, dass sich (aus der Sichtweise der Transaktion) der für die Transaktion relevante Teil des Datenbankzustandes unerwartet ändert. Letzteres bedarf einer Form von Synchronisation zwischen den Transaktionen. Als relevanten Teil des Datenbankzustandes betrachten wir alle Datenbankobjekte, auf die die Transaktion zugreift.

Das angestrebte Ziel ist, Transaktionen unter Einhaltung eines gewissen Konsistenzgrades, der angibt, in welchem Ausmaß eine inkonsistente Sicht auf die Daten gestattet ist, parallel abzuarbeiten und damit den Durchsatz zu erhöhen. Konsistenz und Parallelität stehen dabei in einem konkurrierenden Verhältnis. Je größer die Parallelität, desto geringer die Konsistenz.

5.1 Transaktionskonzept

Die Aufgabe des Transaktionsmanagers ist es, das Verhalten bei parallelem Zugriff und im Fehlerfall zu regeln, wobei jede Transaktion die Datenbank von einem konsistenten Zustand in einen (nicht notwendigerweise verschiedenen) konsistenten Zustand überführt. Eine Transaktion besteht aus einer Folge von logisch zusammenhängenden Operationen, die als Ganzes oder gar nicht ausgeführt werden. Eine Transaktion endet entweder erfolgreich mit dem Befehl COMMIT oder sie wird entweder durch den Benutzer oder durch das DBMS abgebrochen. Ein expliziter Abbruch ist in SQL mit dem Befehl ROLLBACK jederzeit möglich. Laut SQL-Standard [12] beginnt eine neue Transaktion implizit nach dem Verbindungsaufbau und nach jedem COMMIT bzw. ROLLBACK. In MySQL und PostgreSQL gibt es zusätzlich den Befehl BEGIN, der explizit eine Transaktion startet und gleichzeitig den Auto-Commit-Modus, falls aktiviert, vorübergehend verlässt. Ist Auto Commit aktiviert, stellt jeder SQL-Befehl eine Transaktion für sich dar, die mit COMMIT automatisch abgeschlossen wird.

Beispiel 5.1 Eine Transaktion in Pseudocode

```
SELECT Spesen FROM Mahnung WHERE ...
Spesen = Spesen + 0.50
UPDATE Mahnung SET ... WHERE ...
COMMIT
```

In Beispiel 5.1 ist eine einfache Transaktion dargestellt, die einen Datensatz aus der Tabelle Mahnung liest und die Spesen erhöht. Im Folgenden wollen wir uns überlegen, was alles passieren kann, falls nicht nur diese Transaktion läuft, sondern mehrere Transaktionen gleichzeitig ausgeführt werden und sich diese auch gegenseitig unterbrechen können. Diese Annahme ist durchaus realistisch, da aus Performance-Gründen eine serielle Abarbeitung, das heißt eine Transaktion nach der anderen, nicht zweckmäßig wäre, da die Dauer einer Transaktion vom Benutzer abhängig ist und ein Warten auf das Ende der jeweils vorherigen Transaktion damit *beliebig* lange dauern könnte. Stellen wir uns vor, dass zwischen dem SELECT und UPDATE die Spesen genau desselben Datensatzes von einer anderen Transaktion geändert werden. Dann würde dieses Update wieder überschrieben werden und dadurch „verloren“ gehen. Dieses Problem bezeichnen wir gemäß der Literatur [4,19] mit Lost-Update-Problem.

5.2 Probleme ohne Synchronisation

Bevor wir uns den Problemfällen, die bei der verzahnten Ausführung von Transaktionen entstehen, zuwenden, wollen wir über die Operationen in Transaktionen abstrahieren. Wir verwenden das gängige Read-Write-Modell [4], wo uns nur noch die eigentlichen Lese- und Schreiboperationen, abgekürzt mit r(x) und w(x), auf die Datenbankobjekte (Block, Tabelle, Datensatz, …) und die speziellen Operationen Commit und Rollback interessieren. Als Argument von w und r geben wir an, auf welches Objekt sich die Operation bezieht. Wir nehmen an, dass jede Schreiboperation von allen vorher durchgeführten Leseoperationen in der Transaktion abhängt, das heißt dass die Ergebnisse der vor w(x) stehenden Abfragen dazu verwendet werden, den Wert für x zu berechnen. Diese Semantik wird auch in der Praxis verwendet, da es zu aufwändig wäre, die Beziehungen zwischen einzelnen Lese- und Schreiboperationen zu erfassen.

5.2.1 Verlorengegangene Änderungen (Lost Update)

In den folgenden Abbildungen stellen wir den zeitlichen Verlauf von jeweils zwei parallel laufenden Transaktionen dar, die auf gemeinsame Datensätze zugreifen. Die Lost-Update-Anomalie tritt dann auf, wenn ein Wert von einer Transaktion aktualisiert wird, aber anschließend von einer anderen Transaktion, die den alten Wert gelesen hat, überschrieben wird. Der Effekt der Transaktion 1 geht dabei verloren.

Zeitpunkt	Transaktion 1	Transaktion 2
0	r(x)	
1		r(x)
2	w(x)	
3		w(x)

Abbildung 5.1 Lost-Update-Anomalie

5.2.2 Zugriff auf nicht freigegebene Daten (Dirty Read/Write)

Dieses Problem ist dadurch gekennzeichnet, dass eine Transaktion Daten liest, die von einer anderen Transaktion modifiziert wurden, die noch nicht erfolgreich mit Commit beendet wurde. Es besteht die Gefahr, dass der gelesene Wert für x wieder ungültig wird, falls die modifizierende Transaktion später abgebrochen wird. Dies kann fatale Folgen haben, da der nun ungültige Wert von x bereits zu weiteren Datenänderungen geführt hat.

In Abbildung 5.2 sind die beiden verwandten Phänomene Dirty Read und Dirty Write dargestellt. Die Transaktion 1 steht dabei jeweils mit den Transaktionen 2a und 2b in Konflikt.

Zeitpunkt	Transaktion 1	Transaktion 2a	Transaktion 2b
0	r(x)		
1	w(x)		
2		r(x)	r(y)
3		w(y)	w(x)
4		Commit	Commit
5	Rollback		

Abbildung 5.2 Dirty-Read- und Dirty-Write-Anomalie

In Transaktion 2a wird ein nicht freigegebenes Objekt gelesen (Dirty Read), in Transaktion 2b ein nicht freigegebenes Objekt geschrieben (Dirty Write). Letzteres kann dazu führen, dass bei einem Rollback der Transaktion 1 der Wert von x zum Zeitpunkt 0 wiederhergestellt wird, was den Effekt von Transaktion 2b zunichte machen würde.

5.2.3 Nicht wiederholbares Lesen (Non-repeatable Read)

Diese Anomalie liegt vor, wenn eine Transaktion bedingt durch Änderungen paralleler Transaktionen während ihrer Ausführung unterschiedliche Werte eines Objektes sehen kann. Zum Unterschied von Dirty Read wird der Wert von x diesmal durch Commit freigegeben.

Zeitpunkt	Transaktion 1	Transaktion 2
0		r(x)
1	w(x)	
2	Commit	
3		r(x)

Abbildung 5.3 Non-repeatable-Read-Anomalie

5.2.4 Phantom-Phänomen

Eine besondere Form des nicht wiederholbaren Lesens ist das Phantom-Phänomen. Hierbei ändert sich auf Grund von parallelen Datenmanipulationen auf einer Tabelle die Menge der Datensätze, die von einer WHERE-Bedingung erfasst werden. Wird die WHERE-Bedingung zu zwei unterschiedlichen Zeitpunkten ausgewertet, kann es zu Inkonsistenzen kommen.

Zeitpunkt	Transaktion 1	Transaktion 2	Bemerkung
0		r(X)	$X=\{x_1,\ldots,x_n\}$
1	w(X)		fügt y in X ein
2		r(X)	$X=\{x_1,\ldots,x_n,y\}$

Abbildung 5.4 Phantom-Phänomen

Beispielsweise wird in Transaktion 2 zweimal die Summe aller Mahnspesen eines bestimmten Lesers berechnet. Dazwischen wird jedoch von der Transaktion 1 eine den Leser betreffende neue Mahnung eingetragen. Dies führt zu einer Inkonsistenz zwischen dem ersten und zweiten Auswerten der Mahnspesen aus der Sicht von Transaktion 2.

5.3 Prinzipien der Transaktionsverarbeitung

Die im vorigen Abschnitt beschriebenen Problemfälle sind in den meisten Fällen nicht tolerabel, da sie zu einer falschen Sicht auf die gespeicherten Werte führen können. Damit besteht auch die Gefahr, basierend auf den inkonsistenten Daten falsche Werte in die Datenbank zu schreiben. In diesem Abschnitt stellen wir anzustrebende Eigenschaften der Transaktionsverarbeitung vor, um solche Anomalien zu vermeiden.

5.3.1 ACID-Prinzip

Die folgenden vier Eigenschaften für Transaktionen gewährleisten korrekte Synchronisation und Fehlertoleranz, das heißt sie ermöglichen ein korrektes Rollback von fehlerhaften oder abgestürzten Transaktionen. ACID [4] ist ein Akronym und steht für:

- **Atomarität (Atomicity)**
  Die Transaktion wird entweder vollständig oder gar nicht ausgeführt. Das bedeutet: Wenn die Transaktion aus welchem Grund auch immer abgebrochen wird, werden keine Modifikationen nach außen sichtbar.

- **Konsistenz (Consistency)**
  Alle Integritätsbedingungen der Datenbank werden eingehalten, das heißt eine Transaktion hinterlässt stets einen konsistenten Zustand. Trifft dies für eine Transaktion nicht zu, da sie beispielsweise eine Schlüssel-Bedingung verletzen würde (verursacht etwa durch eine „falsche“ Einfügung), wird sie abgebrochen und ihr Effekt auf Grund der Atomarität aufgehoben. Dies ist bei PostgreSQL der Fall, in SAP DB und MySQL führen fehlerhafte Befehle zu keinem Transaktionsabbruch.

- **Isolation (Isolation)**
  Jede Transaktion soll unabhängig von anderen ausgeführt werden. Insbesondere darf der Effekt einer Transaktion erst dann nach außen und damit für andere Transaktionen sichtbar werden, falls diese erfolgreich mit einem Commit abschließt. Diese Eigenschaft verhindert die im vorigen Abschnitt erläuterten Anomalien im Mehrbenutzerbetrieb.

- **Persistenz (Durability)**
  Der Effekt erfolgreich abgeschlossener Transaktionen soll auch im Falle eines unmittelbaren Absturzes nach Commit dauerhaft erhalten bleiben. Dies impliziert die Verwendung eines Sekundärspeichers und Mechanismen für Recovery, die in Kapitel 6 beschrieben werden.

5.3.2 Serialisierbarkeit

Das ACID-Prinzip ist noch relativ abstrakt. In diesem Abschnitt wollen wir untersuchen, welche Auswirkung ACID auf die Transaktionsverarbeitung hat. Die Situation in einem DBMS ist in Abbildung 5.5 dargestellt. Der Scheduler bekommt als Eingabe kontinuierlich Transaktionen (t_1, t_2, t_3, …), die aus den Operationen $p_i \in t_1$, $q_i \in t_2$ und $o_i \in t_3$ bestehen, wobei Operationen r(x), w(x), Commit oder Rollback sein können. Die Aufgabe des Schedulers ist es, eintreffende Operationen so zu reihen, dass ihre Ausführung ACID-konform ist.

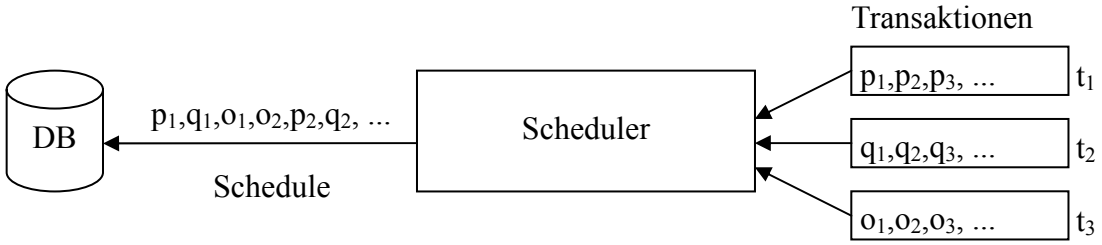

Abbildung 5.5 Transaktionsverarbeitung [4]

Wir weisen darauf hin, dass es sich hierbei um ein dynamisches Scheduling-Problem handelt, da der Scheduler im Voraus weder ein Wissen über zukünftige Transaktionen besitzt, noch vorhersagen kann, welche Operationen innerhalb einer aktiven Transaktion zu erwarten sind.

Definition 5.1 Schedule, serielles Schedule, Serialisierbarkeit

Unter einem *Schedule* bezeichnen wir eine konkrete Reihung der Operationen aus den Transaktionen, die dem DB-Writer als Eingabe übergeben werden. Ein trivialer Scheduler, der das ACID-Prinzip erfüllt, erzeugt ein Schedule, das *seriell* ist. Dies ist einfach eine sequenzielle, unverzahnte Zusammensetzung der Transaktionen. Da ein solcher Scheduler auf das Ende der aktuellen Transaktion warten müsste, um die nächste zu verarbeiten, widerspricht diese Lösung den Anforderungen an ein DBMS, möglichst effizient zu sein. Stattdessen wünscht man die Ausgabe eines *serialisierbaren* Schedules, der in seinem Effekt einem seriellen Schedule äquivalent ist, aber eine verzahnte, parallele Ausführung von Transaktionen zulässt. Das Testen dieses Serialisierbarkeitsbegriffs erweist sich in der Praxis jedoch als nicht durchführbar, da die Semantik von Transaktionen dem DBMS nicht bekannt sein muss. Stattdessen definieren wir einen auf dem Read-Write-Modell aufbauenden Serialisierbarkeitsbegriff, die so genannte *Konflikt-Serialisierbarkeit*.

Definition 5.2 Konflikte

Zwei Operationen p,q ∈ s unterschiedlicher Transaktionen stehen genau dann in einem Schedule s in Konflikt zueinander, falls beide auf ein gemeinsames Datenbankobjekt zugreifen und eine von beiden schreibt.

Definition 5.3 Konflikt-Serialisierbarkeit

Ein Schedule s ist genau dann konflikt-serialisierbar, wenn s durch Vertauschen von je zwei Operationen in s, die weder aus derselben Transaktion stammen noch zueinander in Konflikt stehen, in einen seriellen Schedule umgeformt werden kann.

Diese Eigenschaft kann für ein Schedule s in polynomieller Zeit entschieden werden, indem der zu s korrespondierende *Konfliktgraph* k aufgebaut wird. Die Knoten von k sind die Transaktionen, die in s vorkommen. Eine gerichtete Kante zwischen den Transaktionen t_i und t_j (t_i, t_j sind in s enthalten und $i \neq j$) existiert genau dann, wenn es in t_i eine Operation p und in t_j eine Operation q gibt, die in Konflikt stehen, und die Operation p vor q in s vorkommt. Der Schedule s ist genau dann konflikt-serialisierbar, falls der zu s gehörige Konfliktgraph k azyklisch ist.[12]

Konflikt-Serialisierbarkeit ist eine hinreichende Bedingung für Serialisierbarkeit. Dass sie nicht notwendig ist, beweist folgendes Beispiel, das davon ausgeht, dass es zu keinem Transaktionsabbruch kommt.

Beispiel 5.2 Serialisierbar, aber nicht konflikt-serialisierbar [6]

Eingabe:

t_1: $w_1(y)$, $w_1(x)$

t_2: $w_2(y)$, $w_2(x)$

t_3: $w_3(x)$

mögliche Schedules:

s1: $w_1(y)$, $w_1(x)$, $w_2(y)$, $w_2(x)$, $w_3(x)$

s2: $w_1(y)$, $w_2(y)$, $w_2(x)$, $w_1(x)$, $w_3(x)$

Der Schedule s_2 ist zwar serialisierbar, da der Effekt von s_2 mit dem seriellen Schedule s_1 ident ist. Der Wert von x wird letztendlich von der Transaktion t_3 bestimmt, der Wert von y von t_2. Der Schedule s_2 ist aber nicht konflikt-serialisierbar, da s_2 durch Vertauschen nicht in einen seriellen Schedule überführt werden kann. Der Grund dafür ist, dass $w_1(y)$ zu $w_2(y)$ und $w_1(x)$ zu $w_2(x)$ in Konflikt stehen.

[12] Beweis siehe [4] Satz 17.2.

5.3.3 Striktheit

Bisher wurde stillschweigend angenommen, dass es zu keinen Transaktionsabbrüchen kommt. Der Ausgang einer Transaktion durch Commit oder Rollback wurde von der Konflikt-Serialisierbarkeit bewusst ausgeklammert. Es müssen daher von einem Schedule weiter reichende Eigenschaften als Konflikt-Serialisierbarkeit gefordert werden, damit die Konsistenz nicht durch abgebrochene Transaktionen gefährdet wird.
Die stärkste Eigenschaft bezüglich Fehlersicherheit, die ein korrektes Schedule erfüllen muss, ist die *Striktheit*, die sicherstellt, dass durch andere Transaktionen modifizierte Werte erst nach dem Ausgang dieser Transaktion gelesen und geschrieben werden dürfen. Die Striktheit verhindert damit Probleme, die durch Dirty-Read-(Write-)Anomalien entstehen.

Die drei OSDB erzeugen unter Rücksichtnahme auf den eingestellten Konsistenzgrad konflikt-serialisierbare, strikte Schedules, die dem ACID-Prinzip gerecht werden.

5.3.4 Konsistenzgrade

Striktheit und Konflikt-Serialisierbarkeit stehen in Konkurrenz zur Parallelität, da sie das Verzahnen von Transaktionen, die auf gemeinsame Datenobjekte zugreifen, einschränken. Da in der Praxis nicht immer hundertprozentige Konsistenz erforderlich ist, sieht SQL vor, die Konsistenzeigenschaft, falls gewünscht, aufzulockern. Zum Beispiel kann es je nach Anwendung nichts ausmachen, wenn Dirty Reads erlaubt sind, dafür aber die Geschwindigkeit steigt. In SQL kann man für jede Transaktion mit dem Befehl SET TRANSACTION ISOLATION LEVEL einen Isolationsgrad (= Konsistenzgrad) aus der Tabelle 5.1 festlegen. In der Tabelle ist angegeben, welche Anomalien bei welchem Isolationsgrad auftreten können.

Tabelle 5.1 Isolationsgrade des SQL-Standards [12]

Isolationsgrad (Isolation Level)	Dirty Read	Non-repeatable Read	Phantom-phänomen
READ UNCOMMITTED	Ja	Ja	Ja
READ COMMITTED	Nein	Ja	Ja
REPEATABLE READ	Nein	Nein	Ja
SERIALIZABLE	Nein	Nein	Nein

Der Isolationsgrad SERIALIZABLE gewährleistet das ACID-Prinzip und ist laut SQL-Standard die Voreinstellung, an die sich nur MySQL hält. PostgreSQL untersützt nur SERIALIZABLE und READ COMMITTED. Letzteres allerdings in einer etwas anderen Form. SAP DB deckt alle SQL/92-Isolationsgrade ab.

5.4 Synchronisationskontrolle in den OSDB

Die Eigenschaften Serialisierbarkeit und Striktheit werden von den OSDB durch Sperrprotokolle auf Datenbankobjekten sichergestellt. Die zu Grunde liegende Idee ist, vor jeder Leseoperation eine Lese-Sperre (Read oder Shared Lock), vor jeder Schreiboperation eine Schreib-Sperre (Write oder Exclusive Lock) zu beantragen und diese erst am Ende der Transaktion wieder freizugeben. Sperren steuern grundsätzlich, ob auf ein Objekt zugegriffen werden darf oder nicht. Übertragen wir nun die Konfliktdefinition auf Sperren, besagt dies, dass eine Lese-Sperre auf ein Objekt gesetzt werden kann, falls das Objekt entweder noch gar nicht oder nur lesend gesperrt ist. Eine Schreib-Sperre kann nur dann vergeben werden, wenn das betreffende Objekt freigegeben ist, also keine andere Transaktion eine Lese- oder Schreib-Sperre hält. Entsteht bei der Anforderung einer neuen Sperre ein Konflikt, muss die beantragende Transaktion solange warten, bis die Konflikt verursachende Sperre aufgehoben wird.

Das Setzen und Löschen von Sperren übernimmt der Scheduler für den Benutzer implizit, obwohl es uns die OSDB auch gestatten, explizite Sperren innerhalb einer Transaktion zu setzen und wieder freizugeben. Vom Isolationsgrad der Transaktion hängt es ab, ob und wie lange implizite Sperren gesetzt werden. Die nachfolgenden Bedingungen beziehen sich deshalb auf den höchsten Isolationsgrad SERIALIZABLE.

Fassen wir die Bedingungen für ein korrektes Sperrprotokoll zusammen [19]:

- Jedes zu referenzierende Objekt muss vor dem Zugriff mit einer Sperre belegt werden.
- Die Sperren anderer Transaktionen sind zu beachten. Das bedeutet, dass eine mit gesetzten Sperren unverträgliche Sperranforderung auf die Freigabe unverträglicher Sperren warten muss.

angeforderte Sperre \ aktuelle Sperre	NL	S	X
S	+	+	-
X	+	-	-

+ verträglich
- unverträglich
S = Shared Lock, X = Exclusive Lock
NL = No Lock

Abbildung 5.6 Kompatiblitätsmatrix von Lese- und Schreibsperren [19]

- Keine Transaktion fordert eine Sperre an, die sie bereits besitzt. Sie darf jedoch eine eigene Lese-Sperre, sofern keine anderen Transaktionen Lese-Sperren auf das Objekt innehaben, auf eine Schreib-Sperre erweitern.

- Sperren werden strikt zweiphasig angefordert und freigegeben. Das strikte Zwei-Phasen-Sperrprotokoll sieht vor, dass in der ersten Phase Sperren beantragt werden (Growing Phase), diese aber erst in der zweiten Phase (Shrinking Phase) in einem Zug (zum Zeitpunkt von Commit oder Rollback) wieder freigegeben werden. Damit wird erreicht, dass wenn zwei Transaktionen in Konflikt stehen, die spätere auf den Ausgang der ersteren warten muss, bevor auf das Objekt zugegriffen werden darf.

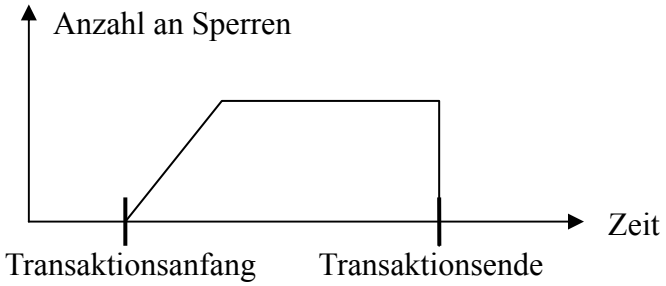

Abbildung 5.7 Growing- und Shrinking-Phase eines strikten Zwei-Phasen-Schedulers [19]

Ein Problem der Sperrprotokolle ist, dass es zu Deadlocks kommen kann, bei denen sich zwei oder mehr Transaktionen gegenseitig behindern, indem sie zyklisch auf Sperren warten. Ein Beispiel mit zwei Transaktionen ist in Abbildung 5.8 dargestellt. Transaktion 1 und 2 warten auf die Freigabe des jeweils durch die andere Transaktion gesperrten Objektes. Dieses Warten ist ohne geeignete Maßnahmen potenziell unendlich. Alle drei OSDB verwenden Deadlock-Erkennung und lösen das Problem, indem die Transaktion, die zu einem Deadlock geführt hat, abgebrochen wird. Einen solchen Abbruch nennt man Synchronisationsfehler. Die Applikation muss mit solchen Fehlern immer rechnen und gegebenenfalls die Transaktion zu einem späteren Zeitpunkt neu starten.

Zeitpunkt	Transaktion 1	Transaktion 2	Bemerkung
0	r(x)		Lese-Sperre auf x
1		r(y)	Lese-Sperre auf y
2	w(y)		wartet auf Freigabe von y
3		w(x)	wartet auf Freigabe von x

Abbildung 5.8 Deadlock auf Grund gegenseitiger Abhängigkeit

5.4.1 SAP DB

SAP DB verwendet ein Sperrprotokoll mit unterschiedlicher Granularität. Es können entweder die gesamte Tabelle oder einzelne Zeilen einer Tabelle gesperrt werden. Falls eine Transaktion mehr als eine vordefinierte Anzahl von Zeilensperren (Parameter MAXLOCKS) auf eine Tabelle hält, versucht der Scheduler diese automatisch in eine Tabellensperre umzuwandeln. Dies ist nur dann möglich, falls die Umwandlung in eine Tabellensperre keinen Konflikt mit anderen Tabellensperren mit sich bringen würde. Es werden drei Sperrtypen unterschieden:

- SHARE
- EXCLUSIVE
- OPTIMISTIC ROW

SHARE- und EXCLUSIVE-Sperren haben die gleiche Kompatibilität wie in der Einleitung beschrieben. OPTIMISTIC ROW-Sperren beziehen sich immer nur auf eine Zeile und sind nur mit EXCLUSIVE-Sperren unverträglich. Sie werden nur explizit verwendet und machen nur unter den Isolationsgraden 0 bis 15 (siehe unten) Sinn, wo keine permanenten SHARE-Sperren auf die gelesenen Objekte gesetzt werden. EXCLUSIVE-Sperren auf Zeilen können nur gesetzt werden, falls sie mit den anderen Zeilensperren und mit den Sperren auf die die Zeile enthaltende Tabelle verträglich sind.

Vom eingestellten Isolationsgrad hängt es ab, welche Sperren für wie lange implizit gesetzt werden. Nur EXCLUSIVE-Sperren werden unabhängig vom Isolationsgrad vor jeder Schreiboperation bis zum Transaktionsende gesetzt. Für SHARE-Sperren gilt Folgendes:

- **Isolationsgrad 0 (entspricht READ UNCOMMITTED)**
  Es werden implizit keine SHARE-Sperren gesetzt.
- **Isolationsgrad 1 (entspricht READ COMMITTED)**
  SHARE-Sperren auf Zeilenebene werden so lange implizit gesetzt, bis die Abfrage die nächste Zeile liest. Falls nichts anderes spezifiziert, ist dies die Standardeinstellung für neue Transaktionen.
- **Isolationsgrad 15**
  In diesem Isolationsgrad wird zusätzlich zum Isolationsgrad 1 noch eine SHARE-Sperre auf Tabellenebene gesetzt. Diese bleiben entweder bis zum Transaktionsende aufrecht oder, falls eine Ergebnistabelle erzeugt wird, bis diese geschlossen wird.

- **Isolationsgrad 2 (entspricht REPEATABLE READ)**
  SHARE-Sperren werden auf Tabellenebene wie im Isolationsgrad 15 gesetzt. Auf Zeilenebene wird vor jedem Zugriff eine SHARE-Sperre beantragt, die bis zum Transaktionsende bestehen bleibt.
- **Isolationsgrad 3 (entspricht SERIALIZABLE)**
  SHARE-Sperren werden vor jedem Zugriff auf eine Tabelle beantragt und bleiben bis zum Transaktionsende aufrecht.

Sperren können aber auch explizit mit den Befehlen LOCK und UNLOCK beantragt oder freigegeben werden. Erwähnenswert ist, dass bei einer SELECT-Anweisung direkt angegeben werden kann, welcher Sperrtyp für die gelesenen Zeilen angefordert werden soll. Wenn man kurz davor ist, einen Datensatz zu modifizieren, ihn aber vorher noch auslesen möchte, kann man gleich beim Auslesen eine EXCLUSIVE-Sperre beantragen. Wird diese gesetzt, kann danach keine andere Transaktion den Wert des Datensatzes lesen oder schreiben.

Beispiel 5.3 Explizite Sperre bei SELECT

```
SELECT * FROM Mahnung WHERE (BuchNr,Entlehndat) IN
(SELECT BuchNr,Entlehndat FROM Ausleihe WHERE LeserNr=1)
WITH LOCK (NOWAIT) EXCLUSIVE
```

Diese Abfrage setzt eine EXCLUSIVE-Sperre auf alle Mahnungen des Lesers mit der Nummer 1. NOWAIT bewirkt, dass die Transaktion nicht auf die Freigabe eventuell vorhandener Sperren wartet, sondern gleich mit einer Fehlermeldung „Lock collision“ beendet werden soll.

SAP DB unterstützt auch Sub-Transaktionen. Das sind verschachtelte Transaktionen in Transaktionen, die eine Transaktion logisch in weitere Einheiten unterteilen. Sub-Transaktionen können rückgängig gemacht werden ohne die Haupttransaktion zu gefährden. Wird die Haupttransaktion abgebrochen, werden natürlich auch sämtliche Sub-Transaktionen unabhängig davon, ob sie mit Commit abgeschlossen wurden oder nicht, zurückgefahren.

5.4.2 PostgreSQL

PostgreSQL verwendet ein kombiniertes Synchronisationsverfahren, das auf dem bisher kennen gelernten Sperrprotokoll und einem Multiversion-Concurrency-Control-System (MVCC) beruht. MVCC legt bei jeder Modifikation eines Datensatzes unter Beibehaltung des ursprünglichen Eintrages eine neue Version desselben an. Damit kann nachvollzogen werden, welchen Zustand die Datenbank zu einem bestimmten Zeitpunkt gehabt hat. Dadurch kann eine Entkopplung von Lese- und Änderungstransaktionen erreicht werden, da eine Lese-

transaktion während ihrer Laufzeit ungeachtet neuer Versionen eine konsistente Sicht auf die Daten beibehalten kann. Dies ermöglicht eine höhere Parallelität, da das Warten auf Sperren deutlich verringert werden kann. Der Nachteil liegt im Verwaltungsaufwand von den Versionen und darin, dass MVCC nur im Isolationsgrad READ COMMITTED Sinn macht. Nicht mehr benötigte Versionen werden von PostgreSQL nicht automatisch beseitigt. Hierfür ist die Vacuum-Routine aufzurufen. PostgreSQL hat nur die folgenden zwei Isolationsgrade:

- **READ COMMITTED**
  Lesezugriffe werden über das Versionssystem abgewickelt und sehen einen so genannten konsistenten Snapshot der Datenbank, wie er zum Zeitpunkt des Transaktionsanfangs bestanden hat. Eigene Änderungen innerhalb der Transaktion sind dieser natürlich sichtbar. Fremde Änderungen hingegen werden erst nach einem Commit lesbar. Schreibzugriffe müssen im Falle eines Konfliktes auf die Freigabe der betroffenen Zeilen warten. READ COMMITTED ist die Standardeinstellung unter PostgreSQL.

- **SERIALIZABLE**
  Lesezugriffe sehen diesmal nur Daten, die zum Zeitpunkt des Transaktionsanfangs freigegeben waren. Änderungen, die nachträglich mit Commit von anderen Transaktionen freigegeben werden, bleiben im Gegensatz zu READ COMMITTED verborgen. Schreibzugriffe, die mit anderen Transaktionen in Konflikt stehen, warten auf deren Ausgang. Terminiert die konkurrierende Transaktion erfolgreich, wird die wartende Transaktion mit einer Synchronisationsfehlermeldung abgebrochen und muss neu gestartet werden.

Wegen des MVCC können zwar konsistente, jedoch unter Umständen alte Daten gelesen werden und auf Grund dieser eine falsche Entscheidung getroffen oder ein inkorrekter Schreibzugriff auf ein anderes Datenbankobjekt getätigt werden. Um dieses Problem zu lösen, gibt es bei der SELECT-Anweisung die Möglichkeit durch Angabe von FOR UPDATE sicherzustellen, dass die gelesenen Werte auch tatsächlich aktuell sind. Gleichzeitig wird eine ROW SHARE-Sperre auf Tabellenebene und eine EXCLUSIVE-Sperre auf die gelesenen Zeilen gesetzt.

Auf Tabellenebene gibt es eine ganze Reihe von unterschiedlichen SHARE- und EXCLUSIVE-Sperren, die zum Beispiel die Aufgabe haben, eine Tabelle während der Vacuum-Routine vor Schemaänderungen zu schützen. Wir verweisen an dieser Stelle auf das Manual [2] Abschnitt 9.3, wo sich eine vollständige Liste der Tabellensperren befindet.

In PostgreSQL gibt es gemäß dem SQL-Standard [12] die Möglichkeit, das Überprüfen von Integritätsbedingungen auf das Ende der Transaktion zu verschieben. PostgreSQL unterstützt dieses Feature im Moment jedoch nur für die Fremdschlüssel-Integrität. Mit dieser gelockerten Integritätsüberprüfung können Datensätze in beliebiger Reihenfolge eingefügt werden. Hauptsache, zum Commit-Zeitpunkt ist wieder alles in Ordnung.

Beispiel 5.4 Integritätsüberprüfung auf das Ende der Transaktion aufschieben

```
SET CONSTRAINTS ALL DEFFERED
```

5.4.3 MySQL

MySQL, das ursprünglich nicht für Transaktionsverarbeitung konzipiert wurde, wartet mit einem alternativen Konzept auf, mit dem Konsistenz gewährleistet werden soll. Betrachten wir zunächst die Situation bei MyISAM-Tabellen. Einzelne SQL-Operationen werden isoliert abgearbeitet. Der Benutzer hat die Möglichkeit, folgende explizite Tabellen-Sperren zu setzen, die Isolation über mehrere Befehle hinweg gewährleisten:

- **READ**
  Auf die gesperrte Tabelle darf nur noch lesend zugegriffen werden.

- **READ LOCAL**
  Lesen und Einfügen (INSERT) auf die gesperrte Tabelle ist erlaubt.

- **WRITE**
  Nur der aktuelle Thread (= Session) hat vollen Zugriff auf die Tabelle, alle anderen müssen auf die Freigabe dieser Sperre warten. Eine beantragte WRITE-Sperre hat höhere Priorität als eine READ-Sperre und wird gegenüber wartenden READ-Sperren bevorzugt.

- **LOW_PRIORITY WRITE**
  Das ist eine WRITE-Sperre, die jedoch in der Warteschlange nicht vorgereiht wird.

Diese Sperren werden mit den Befehlen LOCK TABLE und UNLOCK TABLE angefordert und verwaltet. Um Deadlocks zu vermeiden, muss der Benutzer alle Sperren, die er in einer Session benötigt, mit einem einzigen LOCK TABLE-Befehl anfordern. MySQL führt diese Anforderung dann atomar aus, so dass es zu keinen Deadlocks kommen kann. Treten Deadlocks auf, falls der Benutzer Sperren stufenweise anfordert, bricht MySQL die Sperranforderungen ab.

MySQL erlaubt verzögertes Einfügen (Befehl INSERT DELAYED) in eine Tabelle, während diese gesperrt ist. Ein eigener Thread ist dafür zuständig, die Datensätze sobald wie möglich in die Tabelle einzufügen. Der Client kann auf diese Weise zwar weiterarbeiten, hat aber keine hundertprozentige Gewährleistung, dass das Einfügen erfolgreich sein wird. Wenn MySQL beispielsweise abstürzt, sind noch nicht eingefügte Datensätze verloren.

Tabellen vom Typ InnoDB und BDB unterstützen Transaktionen. InnoDB verwendet ähnlich wie PostgreSQL Multiversionen-Synchronisation gepaart mit Sperren auf Zeilenebene. BDB verwendet Sperren auf Blockebene. Verwendet man einen der beiden Tabellentypen, wird davon abgeraten, die MySQL eigenen Tabellensperren zu benutzen, da sie nicht in das Transaktionskonzept passen und eine laufende Transaktion beenden. Ab Version 4.0.5 unterstützen InnoDB-Tabellen alle vier Isolationsgrade, BDB-Tabellen hingegen nur SERIALIZABLE. Im Gegensatz zu PostgreSQL besteht bei InnoDB nicht die Gefahr, alte Daten zu lesen, da jeder Lesezugriff eine Lese-Sperre setzt.

Kapitel 6
Recovery und Replikation

In diesem Kapitel befassen wir uns mit den Möglichkeiten, während des Betriebes auftretende Fehler des DBMS zu beheben (Recovery) und einem Ausfall durch Redundanz vorzubeugen (Replikation). Beides ist wichtig, um einen potenziellen Datenverlust zu vermeiden und eine hohe Verfügbarkeit zu erreichen. In vielen Anwendungsbereichen führt ein auch nur kurzzeitiger Ausfall des DBMS zu einem beträchtlichen wirtschaftlichen Verlust.
Am Anfang des Kapitels klassifizieren wir mögliche Ausfälle und untersuchen die Möglichkeiten, diese abzuwenden. Im Vergleich zu Oracle bieten die drei OSDB von Haus aus keine oder nur geringe Replikationsunterstützung. Wir zeigen aber Möglichkeiten, wie man dennoch die Ausfallsicherheit zum Beispiel durch den Betrieb von zwei redundanten Servern verbessern kann.

6.1 Fehlerklassen

In der Literatur [4,5,6] wird zunächst differenziert, ob der Fehler in der Applikation oder im DBMS auftritt. Aus der Sichtweise des DBMS unterscheidet man folgende Fehlerklassen:

- **Transaktionsfehler**
  Dieser Fehler tritt auf, wenn eine Transaktion ihren Commit-Punkt nicht erreicht. Dies kann mehrere Gründe haben. Eine Transaktion kann von sich aus abbrechen, zum Beispiel durch ein Rollback-Kommando oder wenn die Applikationssoftware abstürzt und die Verbindung trennt. Sie kann aber auch durch das DBMS abgebrochen werden, wenn in der Transaktion ein fehlerhafter SQL-Befehl ausgeführt wird, oder wenn es auf Grund von Concurrency zu einem Transaktionsabbruch kommt, um die Serialisierbarkeit zu gewährleisten.

- **Systemfehler**
  Das System, auf dem das DBMS läuft, oder das DBMS selbst fallen auf Grund eines Absturzes aus. Sämtliche Information, die sich zum Zeitpunkt des Ausfalls im Hauptspeicher, insbesondere in den Puffern, befunden haben, gehen verloren. Der Sekundärspeicher bleibt aber intakt. Weiters nehmen wir an, dass ein Systemfehler nicht zu einer Beschädigung des Dateisystems des Sekundärspeichers führt.

- **Medienfehler**
  Das sind Fehler des Sekundärspeichers wie Plattenausfälle (Headcrashes), unlesbare Sektoren oder Zerstörung des Dateisystems. Ein Medienfehler kann auch einen Systemfehler bewirken, falls das DBMS nicht mehr in der Lage ist, seine Puffer auf die Platte zu schreiben, oder gar das gesamte Betriebssystem abstürzt. Medienfehler können daher als Verallgemeinerung von Systemfehlern aufgefasst werden.

Im Folgenden untersuchen wir Maßnahmen, um die oben aufgezählten Fehler in den Griff zu bekommen und Ausfälle oder Datenverlust zu vermeiden.

6.2 Recovery

Unter Recovery verstehen wir Maßnahmen, um die oben beschriebenen Fehler zu beheben, also insbesondere einen gültigen Zustand des DBMS wiederherzustellen.

6.2.1 Recovery von Transaktionsfehlern

Das Problem von Transaktionsabbrüchen kann man sich am besten vorstellen, wenn wir uns die allgemeine Organisation eines Data-Managers ansehen. Der Recovery-Manager schreibt und liest Informationen in den instabilen Puffer im Hauptspeicher und fordert gegebenenfalls neue Datensätze vom Puffer-Manager an, der diese von der DB liest.

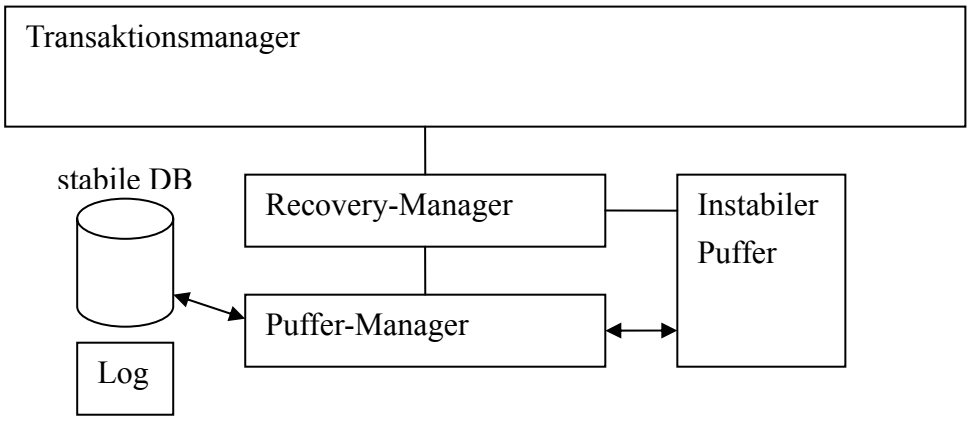

Abbildung 6.1 Organisation eines Data-Managers [3]

Da der Speicherplatz im instabilen Puffer beschränkt ist, müssen periodisch Bereiche davon in die DB zurückgeschrieben werden. Das ist ein Grund, warum nach einem Transaktionsabbruch bereits Änderungen auf die Platte geschrieben sein könnten.
Hier kommt das Log ins Spiel, das alle Änderungen mitprotokolliert. Genauer wird vermerkt, welche Transaktion an welcher Stelle der DB welche Änderung vornimmt. Das Log wird dabei sequenziell beschrieben, sodass es die Ausführungsreihenfolge, die durch den Transaktionsmanager vorgegeben wird, widerspiegelt. Werden beispielsweise alle Schreibzugriffe einer Transaktion sofort in die DB geschrieben und bricht diese Transaktion später ab, müssen die geänderten Werte rückgängig gemacht werden. Dazu sucht der Recovery-Manager nach Einträgen der abgebrochenen Transaktion im Log rückwärts und ersetzt die Änderungen durch den alten Wert (Before Image). Man spricht hierbei von einem Undo-Log.
Ein Redo-Log hingegen ist ein Log, das nicht den alten Wert, sondern den neuen Wert (After Image) speichert. Änderungen dürfen hierbei erst nach dem Commit in die DB geschrieben werden. Kommt es nach dem Commit, aber bevor die neuen Werte in die DB geschrieben wurden, zu einem Absturz, können mittels eines Redo-Logs die noch nicht ausgeführten Änderungen wiederholt werden. In der Praxis wird oft eine Kombination dieser beiden Verfahren, das so genannte Undo-Redo-Log, verwendet.

Die OSDB verwenden alle eine Variante von Logging, um sich gegen Transaktionsfehler und Systemfehler abzusichern.

6.2.2 Recovery von Systemfehlern

Kommt es zu einem Systemfehler, könnte ein mögliches Szenario wie folgt aussehen:

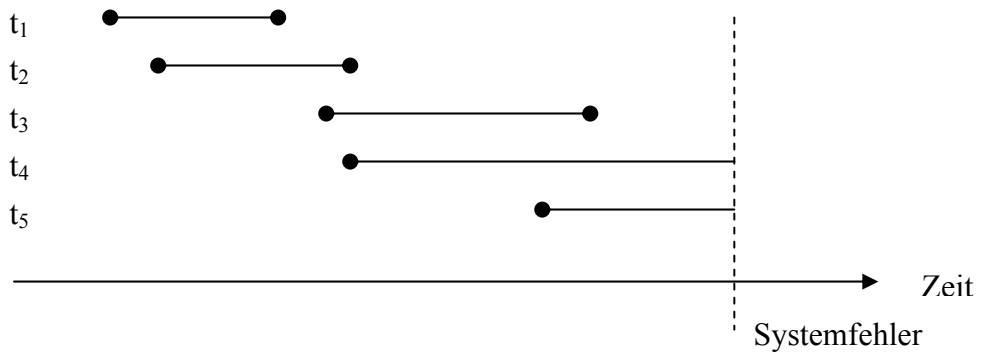

Abbildung 6.2 Szenario eines Systemfehlers [3]

Zum Zeitpunkt des Fehlers existieren noch aktive Transaktionen (t_4,t_5). Diese müssen bei einem Neustart des DBMS rückgängig gemacht werden. Hierbei wird, sofern es zu keinem Medienfehler gekommen ist, die Loginformation, wie im vorigen Abschnitt beschrieben, verwendet, um zu einem konsistenten Zustand der DB zurückzukehren.

6.2.3 Recovery von Medienfehlern

Bei einem Verlust der persistenten DB-Dateien muss man auf ein Backup zurückgreifen. Klar ist, dass man je nach benötigter Aktualität Backup-Intervalle festlegen und Backup-Medien an einem sicheren Ort aufbewahren sollte. Als Ergänzung für höhere Ausfallsicherheit ist der Einsatz eines RAID-Systems [5] bzw. Replikation (siehe Abschnitt 6.3) zu erwägen.
In diesem Unterkapitel befassen wir uns mit den Möglichkeiten der OSDB, Backups zu erstellen und diese wiedereinzuspielen.

6.2.3.1 SAP DB

In SAP DB können nur DBM-Benutzer mit den Server-Rechten BACKUP und RECOVERY Datenbankinstanzen sichern und wiederherstellen. Dazu muss zuerst ein Backup-Medium (normalerweise eine Datei) angelegt werden. Es können separat sowohl die Datendateien als auch die Logdateien gesichert werden. Es werden folgende Arten von Backup unterschieden:

- **Daten-Backup**
  Es werden folgende Backup-Varianten angeboten:
  - **Vollständig** (= die gesamte DB wird gesichert)
  - **Inkrementell** (= nur geänderte Seiten werden gesichert)
  - **Mit Checkpoint**
    Ein Daten-Backup mit Checkpoint wartet bis alle aktiven Transaktionen beendet sind, und beginnt erst dann mit dem Backup. Während der Backup-Dauer sind nur Lesezugriffe auf die DB Instanz erlaubt. Dies führt zu einem konsistenten Backup.
  - **Ohne Checkpoint**
    Ein Backup ohne Checkpoint startet sofort und erlaubt auch während des Backups Lese- und Schreibzugriffe. Da Daten noch während des Backups geändert werden, kann es zu einem inkonsistenten Abbild der DB kommen. Um ein konsistentes Restore durchzuführen, wird deshalb zusätzlich noch die Logdatei benötigt, die den Zeitraum zwischen Backup-Beginn und Backup-Ende abdecken muss.

- **Log-Backup**
  Um ein Daten-Backup auf den neusten Stand bzw. konsistent zu machen, benötigt man Log-Backups. Ein weiterer Grund, warum Log-Backups regelmäßig durchgeführt werden müssen, ist der, dass beim Backup nicht mehr benötigte Log-Segmente wieder freigegeben werden. Deswegen gibt es den Modus *Automatic Log Backup*, der – sobald ein Log-Segment voll ist – ein Log-Backup durchführt. Ein *Interactive Log Backup* (= manuelles Backup) wird hingegen nicht empfohlen.

Eine Besonderheit von SAP DB besteht darin, dass es erlaubt ist, das Log Devspace zu spiegeln. Besitzt man ein Daten-Backup mit einem aktuellen Log-Backup, kann man den Datenbankzustand retten, der zu dem Zeitpunkt existiert hat, als das Log-Backup durchgeführt wurde, ausgenommen der aktiven Transaktionen, die natürlich rückgängig gemacht werden. Zu erwähnen ist noch, dass ein Restore nur offline, das heißt bei heruntergefahrener Datenbankinstanz, möglich ist.

Zur Erstellung von Sicherungskopien kann auch noch der Replication Manager verwendet werden, der zum Beispiel das Ergebnis einer beliebigen SQL-Anweisung oder eine ganze Tabelle in eine Datei extrahieren bzw. importieren kann. Folgende Dateiformate werden unterstützt:

- **COMPRESSED**
  Datensätze sind durch Zeilenumbruch, Attribute mit Komma getrennt. Dieses Format bezeichnet man auch als CSV (Comma Separated Values).

- **FORMATTED**
  Datensätze sind wieder durch Zeilenumbruch getrennt. Die Attributwerte beginnen jedoch an einer fixen, absoluten Position und benötigen daher kein Trennzeichen mehr.

- **COMPACT**
  Die Daten werden in einem proprietären Binärformat gespeichert.

Beispiel 6.1 Kommandodatei für den Replication Manager

```
DATAEXTRACT * FROM Ausleihe, Buch, Leser
WHERE Ausleihe.BuchNr = Buch.BuchNr
AND Ausleihe.LeserNr = Leser.LeserNr
OUTFIELDS Entlehndat 1
          Titel 2
          Name 3
OUTFILE 'buch.data' COMPRESSED
DATE 'DD.MM.YYYY'
```

Das Kommando DATAEXTRACT führt einen SQL-Befehl aus und schreibt die Attribute, die nach OUTFIELDS angegeben wurden, in die Datei *buch.data*. Die Zahlen hinter den Attributnamen geben die Reihenfolge in der Ausgabedatei an.

6.2.3.2 PostgreSQL

PostgreSQL bietet im Unterschied zu SAP DB keine explizite Möglichkeit, Daten- und Logdateien zu sichern. Stattdessen werden Tools zur Verfügung gestellt, die alle benutzerdefinierten Daten einer Datenbank in Form eines SQL-Skriptes extrahieren. Um eine Datenbank wiederherzustellen, wird lediglich das erzeugte Skript in psql ausgeführt. Weiters kann mit dem SQL-Befehl COPY der Inhalt einzelner Tabellen in Form von Text- oder Binärdateien importiert und exportiert werden.

Die für die Sicherung zur Verfügung gestellten Tools arbeiten konsistent und können während des laufenden Betriebes erstellt werden. Insbesondere wirken sich Datenänderungen nach dem Start der Sicherung auf diese nicht mehr aus. Die Tools sind im Folgenden kurz beschrieben:

- **pg_dump**
  Erzeugt ein SQL-Skript oder ein Archiv (proprietäres Format) einer einzelnen Datenbank. Archive werden mit pg_restore wiederhergestellt. Das Programm pg_dump extrahiert alle benutzerdefinierten Typen, Funktionen, Tabellen, Indexe, Aggregatfunktionen und Operatoren.

- **pg_dumpall**
  Dieses Tool ist eine Erweiterung zu pg_dump und extrahiert alle Datenbanken in einem Cluster. Zusätzlich werden auch alle angelegten Benutzer und Gruppen extrahiert.

- **pg_restore**
  Wurde statt eines SQL-Skriptes ein Archiv erstellt, kann dieses mit pg_restore widerhergestellt werden. Das Tool pg_restore bietet die Möglichkeit, bestimmte Inhalte der Sicherung selektiv wiederherzustellen. Zum Beispiel kann man angeben, nur die Schemainformation oder nur eine einzelne Tabelle wiederherzustellen.

Ein SQL-Skript, das zur Regenerierung einer Datenbank erstellt wird, hat den Vorteil, dass es versionsunabhängiger als ein binäres Backup der DB-Dateien ist und daher auch für Migration verwendet werden kann. Der Nachteil ist hingegen eine längere Laufzeit beim Erstellen und Interpretieren des SQL-Skriptes.

6.2.3.3 MySQL

In MySQL hängt das Backup-Verfahren wesentlich von den verwendeten Tabellentypen ab. Für InnoDB-Tabellen gibt es ein eigenes kostenpflichtiges Hot-Backup-Tool[13], das aber nicht im Umfang von MySQL enthalten ist. Die folgenden Backup-Strategien beziehen sich primär auf den MySQL eigenen Tabellentyp MyISAM, können aber auch für andere Tabellentypen verwendet werden:

- **Backup auf Dateisystemebene**
  Diese Backup-Variante ist natürlich die simpelste und prinzipiell auch bei den anderen OSDB anwendbar. In MySQL brauchen aber pro Tabelle lediglich drei Dateien (*.frm, *.myd und *.myi) kopiert werden. Selbstverständlich muss während des Kopierens sichergestellt sein, dass keine Schreibzugriffe auf die Tabelle stattfinden und die Puffer im Hauptspeicher auf Platte geschrieben worden sind. Dies wird mit den Befehlen LOCK TABLES und FLUSH TABLES bewerkstelligt.

- **mysqlhotcopy**
  Für das oben beschriebene Verfahren gibt es sogar ein eigenes Skript mysqlhotcopy, das diese Aufgabe automatisiert.

- **mysqldump**
  Dieses Tool generiert von der DB ein SQL-Skript, das die Tabellendefinitionen und die Tabelleninhalte in SQL-Form beinhaltet. Ein Backup dieser Art stellt die sicherste Variante dar, da ein binäres Kopieren der DB-Dateien auch eventuelle Fehler in den Indexdateien mitkopieren würde.

- **SELECT * INTO**
  Dieser SQL-Befehl kann dazu verwendet werden, selektiv Teile der Datenbank zu exportieren. Das Rekonstruieren von Tabellen erfolgt mit dem SQL-Befehl LOAD DATA *Datei*.

- **BACKUP TABLE / RESTORE TABLE / CHECK TABLE / REPAIR TABLE**
  Dies sind eine Reihe von SQL-Befehlen, die auf Dateisystemebene Tabellen sichern, Tabellen wiederherstellen, Indexe überprüfen und reparieren können.

[13] Siehe http://www.innodb.com/hotbackup.html.

Für Änderungen, die nach dem Backup gemacht wurden, ist das *Binary Log* bzw. *Update Log* hilfreich. Dieses kann dazu verwendet werden, falls es auch gesichert wurde oder nach einem Medienfehler noch erhalten geblieben ist, das Backup auf den Stand vor dem Ausfall zu bringen.

6.3 Replikation

Unter Replikation verstehen wir ein redundantes, homogenes Verteilen einer Datenbank auf mehrere Systeme (Cluster), die mit einem Netzwerk miteinander verbunden sind. Es soll also das gleiche DBMS mit den gleichen Daten auf mehreren Systemen laufen. Ziel von Replikation ist es, die Zuverlässigkeit, Erweiterbarkeit und die Leistung des Gesamtsystems [4] zu erhöhen. Folgende Replikationsszenarien [8] sind denkbar:

- **Read-only-Replikation** (Master-Slave)
  Unter der Menge der redundanten Systeme existiert ein Master, der einzig und alleine Änderungen annehmen und durchführen darf. Der Rest, Slaves genannt, kann auf die Daten nur lesend zugreifen. Der Spezialfall, in dem es nur einen Slave und einen Master gibt, wird auch Hot-Standby-Replikation genannt. Diese Variante dient dann primär dazu, hohe Verfügbarkeit zu gewährleisten. Anzumerken ist auch, dass Datensicherung, wie im vorigen Abschnitt beschrieben, anstatt am Master dann am Slave-System durchgeführt werden kann, ohne den laufenden Betrieb zu beeinflussen.

- **Peer-to-Peer-Replikation**
  Bei dieser Variante dürfen alle beteiligten Systeme Änderungen an den Daten vornehmen. Diese müssen aber konsistent im Cluster verteilt werden.

Wesentlich bei Replikation ist, dass Änderungen des Datenbestandes möglichst rasch und konsistent im System verteilt werden, das heißt spätestens bei einem Commit sind die geänderten Daten an die Replikate zu übertragen. Dabei unterscheidet man, ob die Replikation *synchron* oder *asynchron* erfolgt. Erstere Variante propagiert Transaktionen Commit-synchron zu allen anderen Replikaten und sorgt dafür, dass die Transaktionen im Gesamtsystem entweder überall oder gar nicht ausgeführt werden, während zweitere Variante die Replikate erst nach dem lokalen Commit informiert. Dabei kann es zu inkonsistenten Zuständen kommen, wenn zum Beispiel zwei Transaktionen auf unterschiedlichen Systemen ausgeführt werden, die beide das gleiche Tupel verändert haben.
Die Replikationsunterstützung der OSDB, um es gleich vorweg zu nehmen, ist leider nur marginal. Hier punkten kommerzielle Datenbanken wie Oracle oder MS SQL Server. Nur MySQL bietet von sich aus eine Möglichkeit, Hot Standby zu realisieren.

Was man aber immer versuchen kann, ist Master-Slave-Replikation auf Dateisystemebene zu imitieren. Auf eine solche Vorgehensweise wird exemplarisch bei SAP DB näher eingegangen.

6.3.1 SAP DB

SAP DB unterstützt keine der oben beschriebenen Replikationsvarianten. Um die Verfügbarkeit zu erhöhen, kann man aber einen Hot-Standby-Server betreiben, der mittels NFS und DRBD [9] auf dem aktuellen Stand gehalten wird.
Das High Availability Howto [9] beschreibt, wie man die Konfiguration vornehmen könnte. Es existieren die beiden Rechner *dbmain* und *dbbackup*, auf denen SAP DB installiert ist. Eine Partition wird für die Logdateien verwendet und diese wird mittels DRDB auf beiden Systemen gespiegelt.

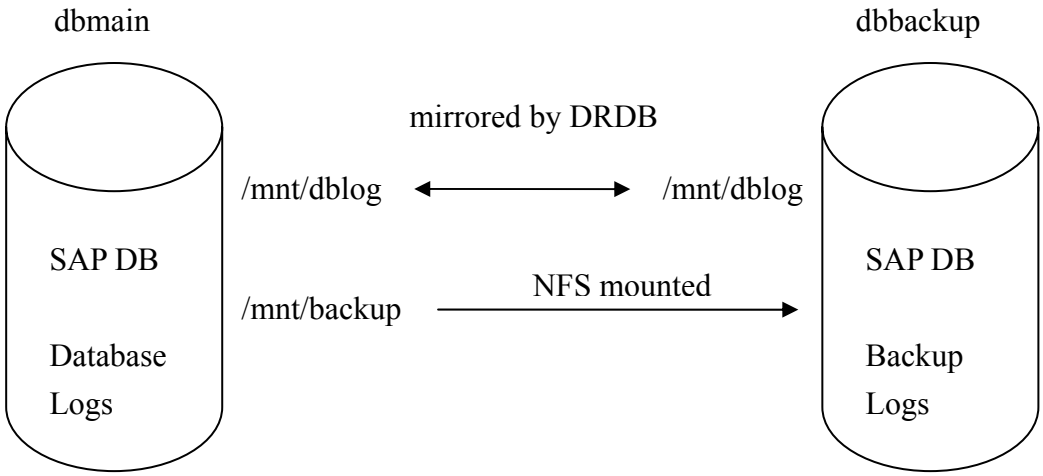

Abbildung 6.3 High Availability durch Spiegeln auf Dateisystemebene

Initialisiert wird der Backup-Server mit dem Kopieren der Datenbankdateien von dbmain auf dbbackup. Der Trick besteht nun darin, zum Beispiel jeden Tag ein Daten-Backup zu machen, dieses aber über NFS auf den Backup-Server dbbackup zu spielen.
Ein Fail-over auf den dbbackup-Server funktioniert wie folgt:

1. DRDB wird deaktiviert.
2. Die Partition mit den Logdateien wird gemountet.
3. Eventuell wird die IP-Adresse des ausgefallenen dbmain übernommen.
4. SAP DB wird neu gestartet.
5. Die Datenbank wird geöffnet.
6. Mit Hilfe des täglichen Backups und der aktuellen Logdateien kann der aktuelle Zustand unmittelbar vor dem Fail-over wiederhergestellt werden.

Die Verwendung des DUAL-Log-Modes, um die Logdateien zu spiegeln, wird nicht empfohlen, da diese Funktionalität laut Aussagen[14] von SAP AG in zukünftigen Versionen nicht mehr unterstützt werden soll.

6.3.2 PostgreSQL

Es laufen momentan Projekte [8], Replikation in PostgreSQL zu integrieren. Hier soll exemplarisch das Postgres-R Projekt[15] [10] vorgestellt werden, das derzeit als einziges synchrone Replikation unterstützt.
Postgres-R integriert eine effiziente synchrone Peer-to-Peer-Replikationsunterstützung in PostgreSQL Version 6.4.2. Basis von Postgres-R ist eine Group-Kommunikationssoftware, die Multicast-Nachrichten an die partizipierenden Server zuverlässig schickt und für die Einhaltung der gleichen Reihenfolge der Nachrichten im Gesamtsystem sorgt.

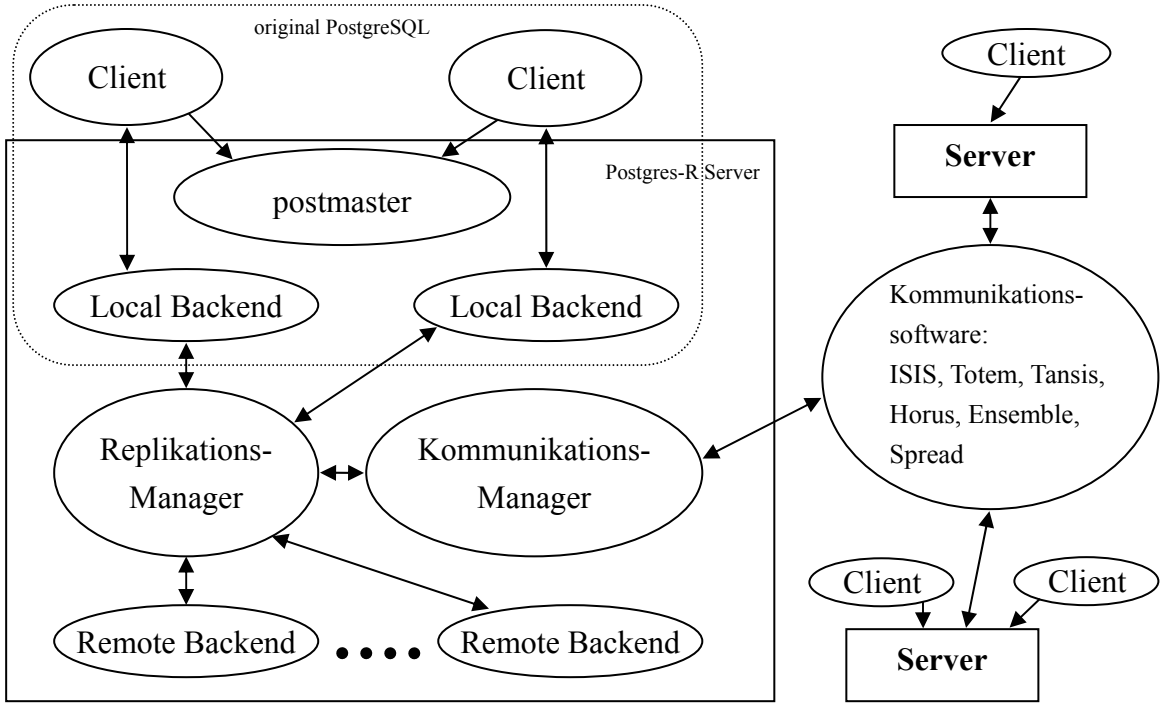

Abbildung 6.4 Architektur von Postgres-R [10]

Für jede Clientverbindung erzeugt der postmaster einen *Local Backend*-Prozess, der die Verbindung verwaltet und Transaktionen entgegennimmt. Jede schreibende Transaktion wird über den Replikations- und Kommunikationsmanager an die Replikate geschickt. Die *Remote Backend*-Prozesse sind dafür zuständig, Transaktionen von anderen Servern zu empfangen und zu verarbeiten.

---

[14] Gemäß einem Posting in der offiziellen SAP DB Mailinglist vom 14.08.2002 von Torsten Strahl (SAP AG).

[15] Siehe http://gborg.postgresql.org.

Transaktionen, die nur lesend auf die Datenbank zugreifen, werden lokal bearbeitet. Das Protokoll für schreibende Transaktionen sieht wie folgt aus:

1. **Local Phase**: Alle Operationen der Transaktion werden zunächst lokal abgearbeitet. Dadurch werden fehlerhafte Transaktionen bereits lokal erkannt und zurückgewiesen.
2. **Send Phase**: Am Ende der Transaktion werden alle Schreibzugriffe (Write Set), die die Transaktion lokal ausgeführt hat, in eine Nachricht gepackt und diese dann an alle aktiven Replikate und auch an sich selbst verschickt. Die geschriebenen Daten werden in Schattenkopien gehalten und sind außerhalb der Transaktion noch nicht sichtbar. Die Kommunikationssoftware stellt eine totale Ordnung der Transaktionen im Gesamtsystem sicher.
3. **Lock Phase**: In dieser Phase versucht die Transaktion alle notwendigen Sperren zu setzen. Gelingt das auf Grund eines Konfliktes nicht, wird die Transaktion abgebrochen und es wird an die Replikate eine Abort-Nachricht gesendet. Die Sperren müssen dabei atomar und im Gesamtsystem in der gleichen Reihenfolge bearbeitet werden. Das wird durch die Ordnungssemantik des Kommunikationssystems erreicht.
4. **Write Phase**: Wurden alle Sperren erfolgreich gesetzt, wird ein Commit gesendet. Erst jetzt wird die Schattenkopie gültig und die Replikate dürfen das *Write Set* in die Datenbank übernehmen.

Postgres-R benötigt kein aufwändiges Zwei-Phasen-Commit, sondern nützt die Möglichkeiten eines darunterliegenden, zuverlässigen Group-Kommunikationssystems aus. Beim Zwei-Phasen-Commit-Protokoll wird jeder Teilnehmer im Cluster in der ersten Phase gefragt, ob ein Commit aus seiner Sicht zulässig ist. In der zweiten Phase, nachdem alle Antworten eingetroffen sind, wird dann entschieden, ob es zu einem Commit kommt oder nicht. Die Entscheidungsfindung bei obigem Protokoll ist hingegen lokal möglich. Leider ist Postgres-R bis jetzt nur in einer älteren PostgreSQL-Version integriert. Am Einbinden in die aktuelle Version wird derzeit gearbeitet.

6.3.3 MySQL

MySQL ist das einzige der drei untersuchten Datenbanksysteme, das von Vornherein eine asynchrone Master-Slave-Replikation unterstützt. Diese funktioniert über das Binary Log des Masters, in dem alle Update-Operationen chronologisch mitprotokolliert werden. Ausschließlich die Slaves sind dafür verantwortlich, sich mit dem Master zu synchronisieren.

Die Vorgehensweise ist dabei folgende:

1. Anlegen eines speziellen Benutzers, der für die Replikation zuständig ist. Dieser benötigt auf allen Tabellen das REPLICATION SLAVE-Recht.
2. Es muss ein Backup des Masters erstellt werden.
3. Jeder Rechner muss eine eindeutige Server-ID zugewiesen bekommen und das Binary Log aktiviert haben, damit jeder grundsätzlich in der Lage ist, die Rolle des Masters zu übernehmen.
4. Wiederherstellen des Backups auf den Slaves.
5. Konfigurieren und Starten der Slaves.

Hat man einmal einen Master-Slave-Cluster aufgebaut, hat dies mehrere Vorteile. Die Ausfallsicherheit kann durch ein Fail-over auf einen Slave, der neuer Master wird, drastisch erhöht werden. Für dieses Fail-over muss aber ein Watchdog implementiert werden, der von außen den Ausfall erkennt und die Slaves umprogrammiert. Slaves können für beliebige Zeit aus dem Cluster genommen werden und zu einem späteren Zeitpunkt wieder reintegriert werden. Sämtliche Backups können auf einem beinahe aktuellen Slave durchgeführt werden, und behindern durch die obligatorischen Schreibsperren nicht den laufenden Betrieb. Die Slaves können zwecks Entlastung des Masters für Read-only-Zugriffe verwendet werden. Allerdings muss dann in Kauf genommen werden, dass die gelesenen Daten nicht unbedingt auf dem neusten Stand sind.

Die Replikationsunterstützung von MySQL steht trotz nützlicher Ansätze jedoch noch am Anfang der Entwicklung und ist in manchen Punkten noch nicht ganz ausgereift.[16]

[16] Vgl. Punkt 4.10.4 in [3].

Kapitel 7
Kriterienkatalog

Bisher wurden die wichtigsten Konzepte der OSDB vorgestellt und miteinander verglichen. In diesem Kapitel stellen wir einen Kriterienkatalog zusammen, der auf die wichtigsten Merkmale der OSDB eingeht und als Entscheidungshilfe für die Auswahl einer geeigneten Datenbank dienen soll. Die hier erarbeiteten Kriterien beschränken sich nicht unbedingt nur auf die drei OSDB, sondern können auch für einen Vergleich mit anderen, auch kommerziellen DBMS, herangezogen werden. Der Schwerpunkt der Kriterien liegt bei technischen Aspekten aus der Sichtweise des Datenbankanwenders und Administrators. Natürlich ist dieser Kriterienkatalog so wie jener in [17] dem gerade aktuellen Stand der Technik unterworfen und damit Veränderungen am Markt ausgesetzt, die ein regelmäßiges Überarbeiten des Katalogs notwendig machen.

Der Kriterienkatalog soll auch ein genaueres Bild über die (unterschiedliche) Funktionsweise der getesteten Systeme machen. Der Vollständigkeit halber führen wir hier auch Kriterien an, die schon in den vorhergehenden Kapiteln ausführlich erläutert wurden und verweisen in einem solchen Fall auf diese.

7.1 Unterstützte Plattformen

Alle drei OSDB unterstützen eine ganze Reihe von UNIX-Plattformen, unter anderem Linux. Echte Windowsversionen gibt es von SAP DB und MySQL, während PostgreSQL mit dem UNIX-Emulator cygwin[17] unter Windows betrieben werden kann. Binärdistributionen gibt es von SAP DB für Windows, Linux (Intel), Solaris (Sparc), AIX (PPC), Tru64 (Alpha) und HP-

[17] http://www.cygwin.com

UX (PA), von PostgreSQL nur für Intel-basiertes Linux und Solaris und von MySQL für Windows, Linux (Intel, Alpha, Sparc, S/360), Solaris (Sparc), FreeBSD (Intel), MacOS X, Novell NetWare (Intel), HP-UX (PA), AIX (PPC), QNX (Intel), SCO OpenUnix (Intel), SGI Irix und Dec OSF (Alpha). SAP DB unterstützt neben den hier aufgezählten Systemen offiziell keine weiteren, während sich in den Handbüchern von PostgreSQL und MySQL eine ausführliche Liste von erfolgreich getesteten Plattformen befindet.

7.2 Transaktionsverarbeitung

Transaktionen werden von SAP DB, PostgreSQL und auch von MySQL (Tabellentypen InnoDB und BDB) unterstützt. Alle drei Datenbanksysteme erfüllen im Isolationsgrad SERIALIZEABLE die ACID-Eigenschaften. PostgreSQL verhält sich auf Grund der Multiversionen-Synchronisation allerdings ein wenig anders. Für einen genauen Vergleich verweisen wir auf das Kapitel 5.

7.3 Backup und Restore

PostgreSQL und MySQL erstellen Backups der Datenbank in Form von SQL-Skripten, die einfach wiederhergestellt werden können und weit gehend versionsunabhängig sind. SAP DB verwendet so wie kommerzielle DBMS primär Binär-Backups seiner Daten- und Logdateien zur Sicherung. Zusätzlich stellt der Replication Manager die Möglichkeit bereit, Tabellen und Daten als Textdatei zu exportieren und importieren. Backup und Restore wurde bereits in Kapitel 6 behandelt.

7.4 Recovery

Dazu gehört das korrekte Wiederanlaufen im Fehlerfall zum Beispiel nach Systemfehlern. Vor allem muss nach einem Absturz automatisch ein korrekter Zustand der Datenbank wiederhergestellt werden. Das ist bei den OSDB der Fall, sofern Transaktionsverarbeitung verwendet wird. Siehe diesbezüglich auch Kapitel 6.

7.5 Replikation

Replikation ist das Verteilen der Daten auf mehrere Server, um die Last besser aufzuteilen und die Ausfallsicherheit auf Grund von Redundanz zu erhöhen. MySQL ist das einzige der drei OSDB, das eine rudimentäre Master-Slave-Replikation anbietet. Beispiele für eine Hot-Standby-Replikation wurden in Kapitel 6 vorgestellt.

7.6 Optimierung

Mögliche Optimierungsmaßnahmen umfassen das Umformulieren von SQL-Anweisungen in äquivalente, einfachere Anweisungen, die Erzeugung eines optimierten Ausführungsplans und eine effiziente Ausführung desselben, zum Beispiel mittels Indexen. Derartige Laufzeitverbesserungen sind in Abschnitt 4.3 beschrieben.

7.7 Performance

Für einen Geschwindigkeitstest sind folgende Entscheidungen zu treffen:

- **Was soll getestet werden?**
  Definition des zu verwendenden Datenbankschemas, der Abfragen und der Datenmanipulationen in Form von Transaktionen in einem realistischen oder künstlichen Szenario.

- **Single-User- versus Multi-User-Test**
  Bei einem Single-User-Test ist nur ein Benutzer mit dem DBMS verbunden und dieser führt die Abfragen sequenziell aus. Bei einem Multi-User-Test führen mehrere Benutzer gleichzeitig möglicherweise konkurrierende Transaktionen aus, um das Verhalten der Synchronisationskontrolle und damit die erzielbare Parallelität zu testen.

- **Schnittstelle zur Datenbank**
  Wie wird auf das DBMS zugegriffen? Entscheidend für die Performance sind die Programmierschnittstelle und die Clientprogrammiersprache, die zum Einsatz kommen. Das Verwenden einer datenbankspezifischen API ist in der Regel schneller als der Zugriff über einen darauf basierenden ODBC-Treiber (siehe Abschnitt 7.27).

- **Datenbankparameter**
  Dazu zählen Einstellungen wie Cachegröße oder der verwendete Isolationsgrad.

- **Hard- und Softwareumgebung**
  Jeder Test ist natürlich von der eingesetzten Hard- und Software abhängig. Greifen die Clients beispielsweise über ein Netzwerk auf das DBMS zu, fließt die Netzwerkperformance in das Testergebnis mit ein.

Auf Grund der hier angedeuteten Fragestellungen lässt sich erahnen, dass das Testen von Datenbanksystemen keine triviale Aufgabe ist, da eine Vielzahl von Aspekten berücksichtigt werden muss. Am Markt finden sich eine Reihe von Standardwerkzeugen zum Testen der Datenbankperformance. Wir verwenden den *Open Source Database Benchmark*[18], der auf dem synthetischen AS3AP-Benchmark [20] basiert. Wir haben das Programm dahingehend angepasst, dass es neben PostgreSQL und MySQL auch SAP DB unterstützt. Der Test besteht aus drei Teilen: Zuerst wird die Testdatenbank angelegt, danach wird ein Single-User- und Mutli-User-Test ausgeführt. Nach dem Anlegen der Tabellen und Indexe haben wir eine Funktion *optimize* eingefügt, die die datenbankabhängige Tabellenstatistik aktualisiert.
Die Testkonfiguration kann aus Tabelle 7.1 entnommen werden. Alle Tests wurden dreimal ausgeführt und der Mittelwert in den folgenden Ergebnistabellen eingetragen. Wenn nicht anders angegeben, handelt es sich bei den Angaben um Sekunden. Falls eine Operation gescheitert ist, ist die Zeit angegeben, bis der Fehler aufgetreten ist. Gründe dafür können nicht vorhandene Funktionalität oder Deadlock-Probleme sein.

Tabelle 7.1 Benchmark-Testkonfiguration

Benchmark	Open Source Database Benchmark Version 0.14 Datenbasis: data-4mb, Anzahl der Benutzer: 15 (--users 15)
SAP DB	Version: 7.3.23, Isolation Level: 2, Schnittstelle: Embedded SQL
PostgreSQL	Version: 7.2.3, Isolation Level: Read Committed, Schnittstelle: API
MySQL	Version: 4.0.3, Tabellentyp: MyISAM und InnoDB, Isolation Level: Repeatable Read, Schnittstelle: API
Betriebssystem	Linux 2.4.1 (i586)
Hardware	Pentium 166, 80MB RAM, 45GB Festplatte (IBM-DTLA-307045)

Tabelle 7.2 Testergebnis: Datenbankerzeugung

	SAP DB	**PostgreSQL**	**MyISAM**	**InnoDB**
create_tables()	0.09	0.22	0.04	0.29
load()	82.15 [19]	22.63	8.90	28.11
create_idx_uniques_key_bt()	3.57	7.49	4.36	8.64
create_idx_updates_key_bt()	4.35	7.40	4.16	9.41
create_idx_hundred_key_bt()	4.66	7.45	4.23	9.08
create_idx_tenpct_key_bt()	4.65	7.60	4.21	8.42

[18] http://osdb.sourceforge.net

[19] Wir verwenden das Kommando DATALOAD, um die Daten zu laden. Mit FASTLOAD würde das Laden zwar nur ein Viertel der Zeit beanspruchen, dafür ist danach eine Datenbanksicherung notwendig, um mit der Datenbank weiterarbeiten zu können.

create_idx_tenpct_key_code_bt()	4.76	1.17	4.96	9.61
create_idx_tiny_key_bt()	0.02	0.09	0.02	0.07
create_idx_tenpct_int_bt()	3.54	0.76	5.71	12.47
create_idx_tenpct_signed_bt()	3.29	0.88	5.82	16.31
create_idx_uniques_code_h()	3.19	2.11	4.87	11.31
create_idx_tenpct_double_bt()	4.01	1.11	8.12	20.19
create_idx_updates_decim_bt()	3.90	2.42	5.03	11.17
create_idx_tenpct_float_bt()	4.68	1.21	7.15	23.52
create_idx_updates_int_bt()	2.31	0.81	5.42	13.02
create_idx_tenpct_decim_bt()	3.97	2.18	7.74	26.92
create_idx_hundred_code_h()	3.09	1.64	4.98	11.48
create_idx_tenpct_name_h()	3.65	2.11	8.73	29.43
create_idx_updates_code_h()	3.19	1.80	6.08	18.17
create_idx_tenpct_code_h()	3.16	1.78	9.79	33.30
create_idx_updates_double_bt()	3.27	1.20	6.96	20.45
create_idx_hundred_foreign()	0.97	17.50	4.92	failed 0.05
optimize()	0.45	13.04	0.01	0.01
populateDataBase()	150.92	104.60	122.22	321.43

Tabelle 7.3 Testergebnis: Single-User-Test

	SAP DB	**PostgreSQL**	**MyISAM**	**InnoDB**
sel_1_cl()	0.04	0.11	0.01	0.03
join_3_cl()	0.05	0.11	0.02	0.05
sel_100_ncl()	0.17	0.23	0.04	0.06
table_scan()	0.14	0.34	0.26	1.54
agg_func()	6.37	3.49	0.70	1.38
agg_scal()	0.03	0.62	0.00	0.00
sel_100_cl()	0.17	0.20	0.04	0.07
join_3_ncl()	0.06	0.13	0.02	0.05
sel_10pct_ncl()	1.35	2.32	0.44	1.52
agg_simple_report()	0.43	12.79	failed 0.00	failed 0.00
agg_info_retrieval()	0.04	0.16	0.01	0.08
agg_create_view()	0.08	0.18	0.00	0.00
agg_subtotal_report()	0.49	0.50	failed 0.00	failed 0.01
agg_total_report()	0.34	0.20	failed 0.00	failed 0.00
join_2_cl()	0.04	0.02	0.00	0.00
join_2()	0.21	0.42	0.59	0.73

sel_variable_select_low()	0.04	0.03	0.00	0.01
sel_variable_select_high()	2.44	4.94	0.88	0.97
join_4_cl()	0.05	0.04	0.00	0.00
proj_100()	13.94	12.02	1.63	1.72
join_4_ncl()	0.09	0.08	0.02	0.01
proj_10pct()	17.06	17.67	1.01	1.29
sel_1_ncl()	0.04	0.02	0.00	0.01
join_2_ncl()	0.05	0.02	0.01	0.00
integrity_test()	0.45	0.68	failed 1.50	failed 2.93
drop_updates_keys()	1.29	0.08	failed 0.00	failed 0.01
bulk_save()	2.33	0.50	0.31	1.37
bulk_modify()	62.22	334.46	0.91	4.86
upd_append_duplicate()	failed 0.02	0.08	0.00	0.00
upd_remove_duplicate()	0.03	0.01	0.01	0.01
upd_app_t_mid()	0.06	0.02	0.00	0.01
upd_mod_t_mid()	0.22	0.36	0.01	0.00
upd_del_t_mid()	0.19	0.35	0.00	0.02
upd_app_t_end()	0.02	0.01	0.01	0.02
upd_mod_t_end()	0.19	0.35	0.00	0.01
upd_del_t_end()	0.19	0.34	0.01	0.01
create_idx_updates_code_h()	3.18	2.02	0.00	0.00
upd_app_t_mid()	0.05	0.02	failed 0.01	failed 0.01
upd_mod_t_cod()	0.04	0.03	0.00	0.00
upd_del_t_mid()	0.23	0.36	0.01	0.01
create_idx_updates_int_bt()	2.25	0.81	failed 0.00	failed 0.00
upd_app_t_mid()	0.04	0.03	0.01	0.00
upd_mod_t_int()	0.05	0.02	0.00	0.01
upd_del_t_mid()	0.24	0.35	0.01	0.00
bulk_append()	4.49	1.86	1.50	2.51
bulk_delete()	62.78	330.64	3.31	2.01
Single User Test	184.37	730.10	13.30	23.35

Tabelle 7.4 Testergebnis: Mulit-User-Test

	SAP DB	**PostgreSQL**	**MyISAM**	**InnoDB**
Mixed IR (tup/sec)	6.02	5.64	12.01	12.66
sel_1_ncl()	0.54	0.39	0.04	0.00
agg_simple_report()	16.98	225.07	failed 0.00	failed 0.01

mu_sel_100_seq()	0.44	1.76	0.76	1.95
mu_sel_100_rand()	0.44	1.22	0.84	1.68
mu_mod_100_seq_abort()	0.50	4.55	failed 1.53	failed 4.30
mu_mod_100_rand()	0.35	4.51	1.40	4.10
mu_unmod_100_seq()	0.33	4.77	1.41	3.22
mu_unmod_100_rand()	0.37	4.85	1.65	2.98
crossSectionTests(Mixed IR)	19.95	247.14	7.64	18.23
mu_checkmod_100_seq()	0.20	0.40	failed 0.05	failed 0.06
mu_checkmod_100_rand()	0.19	0.26	failed 0.06	failed 0.10
Mixed OLTP (tup/sec)	27.43	61.70	11.75	14.75
sel_1_ncl()	0.12	1.99	0.04	0.00
agg_simple_report()	13.42	25.05	failed 0.00	failed 0.00
mu_sel_100_seq()	0.23	0.28	0.77	1.08
mu_sel_100_rand()	0.22	0.14	0.71	1.60
mu_mod_100_seq_abort()	15.95	18.09	failed 1.57	4.45
mu_mod_100_rand()	0.58	5.04	1.35	2.03
mu_unmod_100_seq()	0.56	failed 10.79	1.62	1.78
mu_unmod_100_rand()	0.48	failed 6.32	1.15	2.41
crossSectionTests(Mixed OLTP)	31.56	68.18	7.22	13.35
mu_checkmod_100_seq()	0.63	failed 0.91	failed 0.70	failed 0.92
mu_checkmod_100_rand()	0.18	failed 2.51	failed 1.03	failed 1.32
Multi User Test	355.27	600.15	291.67	308.83

Am schnellsten lädt MySQL (Tabellentyp MyISAM) die Testdaten in die Datenbank, erweist sich aber bei der Erstellung der Indexe langsamer als SAP DB und PostgreSQL, die dabei in Summe etwa gleich schnell sind. Im Single-User-Test führt MySQL deutlich und ist verglichen mit SAP DB fast 14-mal und mit PostgreSQL fast 54-mal schneller. SAP DB und PostgreSQL verlieren vor allem bei den Operationen Bulk Modify und Bulk Delete erheblich viel Zeit.

Auch im Multi-User-Test sind die beiden MySQL-Probanden zwar die Schnellsten, erzielen aber einen wesentlich geringeren Durchsatz beim Mixed-OLTP-Test, bei dem getestet wird, wie viele Tupeln innerhalb von einer Minute von 15 parallel laufenden, konkurrierenden Prozessen aktualisiert werden können. Bei diesem Test schneidet PostgreSQL mit etwa 62 Tupeln pro Sekunde besonders gut ab, braucht aber für die Auswertung einer Aggregatfunktion mit Sub-SELECT (agg_simple_report) sehr lange, was für das schlechte Gesamtergebnis verantwortlich ist.

Gemessen an der reinen Laufzeit ist MySQL besonders mit MyISAM-Tabellen das schnellste DBMS (427.19 bzw. 653.61 Sekunden), gefolgt von SAP DB (690.56 Sekunden) und etwas abgeschlagen PostgreSQL (1434.85 Sekunden). Fairerweise muss dazugesagt werden, dass MySQL auf Grund mangelnder Funktionalität nicht alle Tests bestanden hat, was zu einer leichten Verfälschung der Testergebnisse führt.

In der Praxis empfiehlt es sich, neben einem solchen Standardtest auch Testfälle aus der intendierten Anwendung herauszunehmen, um realistische Messungen durchzuführen, anhand derer eine Entscheidung für ein bestimmtes DBMS getroffen werden kann. Für die Auswahl eines geeigneten Datenbanksystems werden neben der Geschwindigkeit auch noch weitere Faktoren eine Rolle spielen.

7.8 Rechteverwaltung

Datensicherheit, das heißt den Schutz vor unerlaubtem Zugriff, gewährleisten die drei OSDB durch eine Benutzer- und Rechteverwaltung. Grundsätzlich haben Benutzer nur auf jene Datenbankobjekte Zugriff, die sie selber erstellt haben, das heißt wenn sie Eigentümer sind. Damit auch andere Benutzer auf fremde Objekte zugreifen dürfen, können ihnen bestimmte Rechte, Privilegien genannt, zuerkannt werden. Solche Privilegien können zum Beispiel sein:

- **SELECT, INSERT, UPDATE, DELETE** auf Tabellen
  Diese Rechte erlauben das Lesen, Einfügen, Modifizieren oder Löschen von Datensätzen in Tabellen.

- **EXECUTE** auf Stored Procedures
  Dieses Recht erlaubt das Ausführen einer Stored Procedure.

- **GRANT_OPTION** auf Rechte
  Wird diese Option beim Zuweisen von Rechten gesetzt, kann der Inhaber die zugewiesenen Rechte an andere Benutzer weitergeben, auch wenn er nicht der Eigentümer ist.

Um die Rechtevergabe zu vereinfachen, gibt es die Konzepte von Rollen und Gruppen. Eine Rolle ist eine Menge von Privilegien, die einem Benutzer (oder einer Gruppe) auf einmal (mit einem Befehl) zugewiesen werden kann. Eine Gruppe hingegen ist eine Menge von Benutzern, denen als Gruppe Privilegien (oder Rollen) erteilt werden können.

Im Folgenden stellen wir die Rechteverwaltung der drei OSDB vor.

7.8.1 SAP DB

SAP DB unterscheidet zwischen folgenden Benutzerklassen, die unterschiedliche Rechte innehaben:

- **Database Manager Operator**
  Ein Benutzer dieser Klasse hat Zugriff auf den Database Manager. Abhängig von den zugewiesenen Server-Autorisierungen (SERVER_RIGHTS), ist es einem solchen Benutzer gestattet, Datenbankinstanzen zu starten und zu beenden, Backups durchzuführen und Datenbankparameter zu ändern.

- **Database User**
  Datenbankbenutzer sind in vier Unterklassen aufgeteilt, die berechtigt sind, eine Datenbankinstanz zu verwenden, aber (außer SYSDBA) nicht zu administrieren. Die Benutzerklasse regelt, welche Datenbankobjekte von einem Benutzer angelegt werden dürfen.

  - **SYSDBA** (Database System Administrator)
    Pro Datenbankinstanz gibt es nur einen SYSDBA, der Eigentümer der Systemtabellen ist und neue Datenbankbenutzer anlegen darf.
  - **DBA**
    Ein Benutzer dieser Klasse darf neue Datenbankobjekte anlegen, Rollen definieren, Benutzer in Gruppen zusammenfassen und RESOURCE- und STANDARD-Benutzer anlegen.
  - **RESOURCE**
    Diese Benutzer dürfen nur STANDARD-Benutzer anlegen und neue Datenbankobjekte erstellen.
  - **STANDARD**
    Solche Benutzer haben die wenigsten Rechte. Sie dürfen nur Views, Synonyme und temporäre Tabellen anlegen.

SAP DB unterstützt sowohl das Konzept von Rollen als auch von Gruppen, wobei jeder Benutzer maximal nur einer Gruppe zugeordnet werden kann. Ist ein Benutzer einer Gruppe zugeordnet und erzeugt dieser neue Datenbankobjekte, wird automatisch die gesamte Gruppe Eigentümer des neu angelegten Objektes. Auf Grund dieser Einschränkung ist das Konzept nicht so flexibel wie jenes von PostgreSQL (siehe nächster Abschnitt). Rollen können auch per Passwort geschützt werden.

UPDATE- und SELECT-Privilegien können auf einzelne Spalten einer Tabelle beschränkt werden. Zusätzlich werden noch folgende Privilegien angeboten:

- SELUPD: Kombiniertes SELECT- und UPDATE-Privileg
- INDEX: Erzeugen von Indexen
- ALTER: Verändern des Tabellenschemas
- REFERENCES: Verwenden der Tabelle in einer Fremdschlüssel-Bedingung
- SELECT ON *Sequenz*: Gestattet den Zugriff auf Sequenzen.

In SAP DB ist es auch möglich, für jeden Benutzer den Speicherplatz für temporäre und private Tabellen und die maximal erlaubten Kosten für SQL-Abfragen zu beschränken. Es kann auch festgelegt werden, ob sich ein Benutzer gleichzeitig mehrmals einloggen darf.

7.8.2 PostgreSQL

Beim Anlegen neuer Benutzer können zwei spezielle Attribute gesetzt werden:

- **CREATEUSER** (= Superuser)
  Benutzer, die mit diesem Attribut ausgestattet sind, umgehen alle Zugriffsbeschränkungen. Außerdem können nur Superuser neue Benutzer anlegen.

- **CREATEDB** (= Database Creation)
  Dieses Attribut gestattet dem Benutzer das Anlegen von neuen Datenbanken. Superuser dürfen dies ohnehin.

Datenbezogene Privilegien in PostgreSQL beziehen sich immer nur auf die ganze Tabelle. Dieses Manko kann aber mit Views aufgehoben werden. Folgende Privilegien werden weiters unterstützt:

- RULE: Erzeugen von Rules auf Tabellen/Views
- REFERENCES: Um eine Fremdschlüssel-Bedingung anzulegen, muss dieses Recht sowohl auf die referenzierte als auch auf die referenzierende Tabelle gesetzt sein.
- TRIGGER: Erzeugen von Triggern auf Tabellen
- CREATE: Erstellen von Datenbankobjekten
- TEMP: Erzeugen von temporären Tabellen
- USAGE: Erlaubt die Verwendung von prozeduralen Sprachen (SQL LANGUAGE-Definition) oder das Lesen von Schemata (SQL SCHEMA-Definition).

Mehrere Benutzer können zu Gruppen zusammengefasst werden, wobei ein Benutzer durchaus auch Mitglied bei mehreren Gruppen sein kann. Privilegien können einzelnen Benutzern individuell oder einer gesamten Gruppe zuerkannt werden. Das Rollenkonzept wird dagegen nicht unterstützt.

PostgreSQL bietet verschiedene Authentifizierungsmethoden an. Neben Passwort-Authentifizierung können Kerberos, UNIX IDENT und PAM verwendet werden. Auf IP-Netzwerkebene lässt sich sowohl der Zugriff auf den Server beschränken, als auch die Authentifizierungsmethode für ein Netz wählen, das heißt es ist konfigurierbar, welche Rechner auf den Server zugreifen dürfen und wie die Authentizität der Benutzer festgestellt werden soll. Optional können Verbindungen mit SSL verschlüsselt werden, falls der Server mit dieser Option kompiliert wurde.

7.8.3 MySQL

In MySQL gibt es keinen CREATE USER-Befehl. Stattdessen werden Benutzer implizit bei der Verwendung von GRANT angelegt, wenn der Benutzer, dem Privilegien zugewiesen werden, noch nicht existiert. Pro Benutzer kann eingeschränkt werden, wie oft er sich in der Stunde einloggen und wie viele Updates und SQL-Abfragen er ausführen darf.

Privilegien können auf unterschiedlichen Ebenen gesetzt werden:

- Globale Ebene (gelten für alle Datenbanken auf einem MySQL-Server)
- Datenbankebene (gelten für alle Tabellen innerhalb einer Datenbank)
- Tabellenebene
- Spaltenebene

Die MySQL spezifischen Privilegien sind:

- ALTER: Ausführen von ALTER TABLE
- CREATE: Ausführen von CREATE TABLE
- CREATE TEMPORARY TABLE: Ausführen des gleichnamigen Befehls
- DROP: Löschen von Tabellen
- FILE: Exportieren und Importieren von Daten über Dateien
- INDEX: Erstellen von Indexen
- LOCK TABLES: Setzen von Sperren
- PROCESS: Anzeigen der gerade laufenden Prozesse
- RELOAD: Löschen des Caches mittels FLUSH-Befehl

- REPLICATION CLIENT: Erlaubt dem Benutzer zu fragen, wer die Master- und Slave-Rechner sind.
- REPLICATION SLAVE: Wird von den Slaves benötigt, um das Log des Masters zu lesen.
- SHOW DATABASE: Anzeigen der verfügbaren Datenbanken
- SHUTDOWN: Ausführen von mysqladmin shutdown
- SUPER: Erlaubt das Einloggen, auch wenn die Beschränkung überschritten wurde und zusätzlich das Ausführen folgender Befehle: CHANGE MASTER, KILL *thread*, mysqladmin debug, PURGE MASTER LOGS and SET GLOBAL
- USAGE: Synonym für keine Rechte

MySQL unterstützt weder das Rollen- noch das Gruppenkonzept.

Der Zugang zum Server kann auf Hostebene beschränkt werden, indem ein neuer Benutzer in der Form *Name@Host* angelegt wird. Der Zusatz Host gibt an, von welchem Rechner er sich einloggen darf. Wildcards sind in der Hostangabe erlaubt. MySQL 4.0 bietet die Möglichkeit, Verbindungen mit SSL zu verschlüsseln.

7.9 Datentypen

Eine Gegenüberstellung der unterstützten Datentypen haben wir bereits in Kapitel 3 angeführt. In PostgreSQL ist es sogar möglich, neue Basisdatentypen zu erstellen, indem dafür eigene Input- und Outputfunktionen in einer externen Programmiersprache wie C implementiert werden. Es ist auch möglich, Operatoren für den neu erstellen Datentyp zu programmieren.

7.10 Funktionen und Operatoren

Ein Vergleich des SQL-Funktionsumfangs ergab, dass PostgreSQL und MySQL deutlich mehr Funktionen anbieten als SAP DB. Den Unterschied machen Bit-Operationen, reguläre Ausdrücke bei Suchmasken, Formatieren von DATE-/TIME-Feldern, zusätzliche Stringfunktionen, Zufallszahlen und Fallunterscheidungen (CASE WHEN-Befehl) aus. SAP DB und MySQL bieten einen phonetischen Vergleich von Textfeldern mittels der Funktion SOUNDEX an. PostgreSQL wartet speziell für die geometrischen Datentypen (POINT, BOX, LSEG, LINE, PATH, POLYGON und CIRCLE) mit nützlichen Operationen und Funktionen auf. Zum Beispiel kann man Rechtecke verschieben, skalieren oder die Überlappung mit anderen Rechtecken bestimmen. MySQL punktet mit Mengenfunktionen für den Datentyp SET,

Funktionen für das Ver- und Entschlüsseln von Passwörtern und dem Berechnen von Fingerprints mittels MD5 und SH1.

Die folgenden Tabellen stellen den Funktionsumfang der OSDB gegenüber. Für eine detaillierte Beschreibung der Funktionssemantiken verweisen wir auf die entsprechenden Benutzerhandbücher [1,2,3].

7.10.1 Numerische Funktionen und Operatoren

Tabelle 7.5 Numerische Funktionen und Operatoren

Funktion / Operator	**SAP DB**	**PostgreSQL**	**MySQL**
Arithmetische Operatoren	+, -, *, /	+, -, *, /	+, -, *, /
Integerdivision	DIV	/	CAST auf Divisionsergebnis
Divisionsrest	MOD	MOD, %	MOD, %
Vergleichsoperatoren	<, >, <=, >=, =, !=, <>	<, >, <=, >=, =, !=, <>	<, >, <=, >=, =, !=, <>, <=>
Logische Operatoren	AND, OR, NOT	AND, OR, NOT	AND (&&), OR (||), NOT (!), XOR
Bereichsabfrage	BETWEEN x AND y	BETWEEN x AND y	BETWEEN x AND y
Betrag	ABS	ABS, @	ABS
Fakultät		!, !!	
Pi (π)	PI	PI	PI
Quadratwurzel	SQRT	SQRT, |/	SQRT
Kubikwurzel		CBRT, |//	POW(*x*,1/3)
Signum	SIGN	SIGN	SIGN
Nächst größere Zahl	CEIL	CEIL	CEILING
Nächst kleinere Zahl	FLOOR	FLOOR	FLOOR
Runden und Abschneiden	ROUND, TRUNC	ROUND, TRUNC	ROUND, TRUNCATE
Potenz	POWER[20]	POW, ^	POW
Logarithmen	LOG, LN	LOG, LN	LOG, LOG10, LOG2, LN
Exponentialfunktion	EXP	EXP	EXP

[20] In SAP DB muss der Exponent vom Typ INTEGER sein.

Trigonometrische Funktionen	COS, SIN, TAN, COT, ACOS, ASIN, ATAN, ATAN2, SINH, COSH, TANH	COS, SIN, TAN, COT, ACOS, ASIN, ATAN, ATAN2	COS, SIN, TAN, COT, ACOS, ASIN, ATAN, ATAN2
Radianten nach Grad	DEGREES	DEGREES	DEGREES
Grad nach Radianten	RADIANS	RADIANS	RADIANS
Minimum	LEAST		LEAST
Maximum	GREATEST		GREATEST
Zufallszahlen		RANDOM	RAND
Intervallermittlung			INTERVAL[21]
Bit-Operationen		&, |, #, ~, <<, >>	&, |, ^, ~, <<, >>
Formatierte Ausgabe einer Zahl als String		TO_CHAR	FORMAT
Geometrische Funktionen und Operatoren		+, -, *, /, #, ##, &&, &<, &>, <->, <<, >>, <^, >^, ?#, ?-, ?-| , @-@, ?|, ?||, @, @@, ~=, AREA, BOX, CENTER, DIAMETER, HEIGHT, ISCLOSED, ISOPEN, LENGTH, NPOINTS, PCLOSE, POPEN, RADIUS, WIDTH	

7.10.2 Stringfunktionen

Tabelle 7.6 Stringfunktionen

Funktion / Operator	**SAP DB**	**PostgreSQL**	**MySQL**
Stringkonkatenation	||, &	||	CONCAT, CONCAT_WS
Zeichen zu ASCII-Wert		ASCII	ASCII

[21] Mit dieser Funktion kann festgestellt werden, in welches Intervall ein gegebener Wert fällt.

Zeichen zu Multi-Byte-Code			ORD
ASCII-Code zu Zeichen		CHR	CHAR
Zeichensatz-konvertierung	ASCII, EBCDIC, MAPCHAR, ALPHA	CONVERT, TO_ASCII	
Konvertierung zwischen Binärdaten und ASCII-Text		ENCODE, DECODE	
Stringlänge	LENGTH	LENGTH	LENGTH
Stringlänge in Zeichen		CHAR_LENGTH, CHARACTER_ LENGTH	CHAR_LENGTH, CHARACTER_ LENGTH
Stringlänge in Bits		BIT_LENGTH	BIT_LENGTH
Stringlänge in Oktetten			OCTET_LENGTH
Konvertierung in Groß- und Kleinbuchstaben	UPPER, LOWER	UPPER, LOWER	UPPER (UCASE), LOWER (LCASE)
Erstes Zeichen jedes Wortes groß schreiben	INITCAP	INITCAP	
Beliebige Zeichen vom Anfang/Ende entfernen	TRIM, RTRIM, LTRIM	TRIM, RTRIM, LTRIM, BTRIM	TRIM, RTRIM, LTRIM
String mit beliebigen Zeichen auffüllen	RPAD, LPAD, EXPAND	LPAD, RPAD	LPAD, RPAD, (SPACE)[22]
Substring extrahieren	SUBSTR	SUBSTR, SUBSTRING[23]	SUBSTRING, MID, LEFT, RIGHT
Position eines Substrings ermitteln	INDEX	POSITION, STRPOS	POSITION, LOCATE, INSTR
Stringtoken extrahieren		SPLIT_PART	SUBSTRING_ INDEX
Substring überschreiben		OVERLAY	INSERT
Zeichen ersetzen	TRANSLATE	TRANSLATE	
Substring ersetzen	REPLACE	REPLACE	REPLACE
String wiederholen		REPEAT	REPEAT

[22] SPACE erzeugt einen String bestehend aus *n* Leerzeichen, entspricht LPAD('', *n*, ' ').

[23] SUBSTRING extrahiert Substrings, die mit einem regulären Ausdruck übereinstimmen.

String umkehren			REVERSE
SQL-Literal erzeugen		QUOTE_LITERAL	QUOTE
SQL-Identifier erzeugen		QUOTE_IDENT	
String für phonetischen Vergleich erzeugen	SOUNDEX		SOUNDEX
Zahlen in andere Basis umwandeln	HEX	TO_HEX	CONV, BIN, OCT, HEX
Stringvergleich	<, >, <=, >=, =, !=, <>	<, >, <=, >=, =, !=, <>	<, >, <=, >=, =, !=, <>, STRCMP
Mustersuche	LIKE	LIKE, ILIKE	LIKE
Suche mit regulären Ausdrücken		SIMILAR TO, ~, ~*, !~, !~*	REGEXP, RLIKE
Volltextsuche			MATCH (*Spalten*) AGAINST (*Ausdruck*)
Mengenfunktionen			FIND_IN_SET, MAKE_SET, EXPORT_SET, FIELD, ELT

7.10.3 Datums- und Zeitfunktionen

Tabelle 7.7 Datums- und Zeitfunktionen

Funktion / Operator	**SAP DB**	**PostgreSQL**	**MySQL**
Formatierungs-funktionen	CHAR[24]	TO_CHAR	DATE_FORMAT, TIME_FORMAT
Datumsarithmetik (mit Intervallen)	ADDDATE, SUBDATE [25]	+, -	+, -, ADDDATE (DATE_ADD), SUBDATE (DATE_SUB)
Zeitarithmetik (mit Intervallen)	ADDTIME, SUBTIME	+, -	
Intervallarithmetik		+, -, *, /	

[24] Eingeschränkt auf fünf vordefinierte Formate: EUR, INTERNAL, ISO, JIS und USA.

[25] Erlaubt nur das Hinzuzählen und Abziehen von Tagen, da der Datentyp INTERVAL nicht unterstützt wird.

Datums- und Zeitdifferenz	DATEDIFF, TIMEDIFF	-, AGE	
Periodenarithemtik (Format: Jahr und Monat)			PERIOD_ADD, PERIOD_DIFF
Extraktion von Datums- und Zeitkomponenten	DAYOFWEEK, WEEKOFYEAR, DAYOFMONTH, DAYOFYEAR, DAYNAME, MONTHNAME	EXTRACT, DATE_PART (extrahiert: century, day, decade, dow, doy, epoch, hour, microseconds, millennium, milliseconds, minute, month, quarter, second, timezone_hour, timezone_minute, week, year)	EXTRACT, DAYOFWEEK, WEEKDAY, DAYOFMONTH, DAYOFYEAR, MONTH, DAYNAME, MONTHNAME, QUARTER, WEEK, YEAR, YEARWEEK, HOUR, MINUTE, SECOND
Konvertierungs-funktionen	MAKEDATE, MAKETIME	TO_DATE, TO_TIMESTAMP	TO_DAYS, FROM_DAYS, UNIX_ TIMESTAMP, FROM_ UNIXTIME, SEC_TO_TIME, TIME_TO_SEC
Aktuelles Datum und Systemzeit	DATE, TIME, TIMESTAMP	CURRENT_DATE, CURRENT_TIME, CURRENT_ TIMESTAMP, LOCALTIME, LOCAL-TIMESTAMP, NOW, TIMEOFDAY	CURRENT_DATE (CURDATE), CURRENT_TIME (CURTIME), CURRENT_ TIMESTAMP (NOW, SYSDATE), UNIX_ TIMESTAMP
Komponente abschneiden		DATE_TRUNC	

Testen auf gültiges und endliches Datum bzw. Intervall		ISFINITE	
Konvertieren von Zeitzonen		AT TIME ZONE	

7.10.4 Datentypkonvertierung

Tabelle 7.8 Datentypkonvertierung

Funktion / Operator	**SAP DB**	**PostgreSQL**	**MySQL**
Type Cast		CAST	CAST, CONVERT
String zu Zahl	NUM	TO_NUMBER	
Zahl zu String	CHR	TO_CHAR	FORMAT
String zu Datum/Zeit	CHAR	TO_DATE, TO_TIMESTAMP	
Datum/Zeit zu String		TO_CHAR	DATE_FORMAT, TIME_FORMAT
Geometrische Konvertierungs-funktionen		BOX, CIRCLE, LSEG, PATH, POINT, POLYGON	

7.10.5 Sonstige Funktionen

Tabelle 7.9 Sonstige Funktionen

Funktion / Operator	**SAP DB**	**PostgreSQL**	**MySQL**
NULL-Abfragen	IS NULL, IS NOT NULL	IS NULL, ISNULL, IS NOT NULL, NOTNULL, NULLIF	IS NULL, ISNULL, IS NOT NULL, NULLIF, IFNULL
Erstes Nicht-NULL-Argument zurückliefern	VALUE	COALESCE	COALESCE
„Element von“-Prädikat	IN	IN	IN
Bedingung und Fallunterscheidung	DECODE[26]	CASE WHEN	IF, CASE, CASE WHEN

[26] DECODE ist weniger flexibel als CASE WHEN, da die Bedingungen der einzelnen Zweige nicht variieren können.

Funktionen und Operatoren für Netzwerkadressen		<, <=, >, >=, =, <>, <<, <<=, >>, >>=, BROADCAST, HOST, MASKLEN, SET_MASKLEN, NETMASK, NETWORK, TEXT, ABBREV, TRUNC	INET_NTOA, INET_ATON
Verbindungsspezifische Funktionen	USER, USERGROUP, DATABASE, TRANSACTION	CURRENT_ DATABASE, CURRENT_ SCHEMA[S], CURRENT_USER, USER, SESSION_ USER, VERSION, CURRENT_ SETTINGS, SET_CONFIG	DATABASE, USER, SYSTEM_USER, SESSION_USER, VERSION, CONNECTION_ ID
Passwortverschlüsselung und Fingerprints			PASSWORD, ENCRYPT, ENCODE, DECODE, AES_ENCRYPT, AES_DECRYPT, DES_ENCRYPT, DES_DECRYPT, MD5, SHA (SHA1)
Sperrfunktionen			GET_LOCK, RELEASE_LOCK, IS_FREE_LOCK
Benchmark			BENCHMARK
Funktionen für Sequenzen	NEXTVAL, CURRVAL [27]	NEXTVAL, CURRVAL, SETVAL	LAST_INSERT_ID

[27] Hierbei handelt es sich in Wirklichkeit um Attribute von Sequenzen und nicht um Funktionen.

Master-Slave-Synchronisation			MASTER_POS_WAIT

7.11 Aggregatfunktionen

Aggregatfunktionen fassen alle Werte einer Spalte bzw. einer Gruppe, falls eine GROUP BY-Klausel verwendet wird, zu einem Wert zusammen.

Tabelle 7.10 Aggregatfunktionen

Aggregatfunktion	**SAP DB**	**PostgreSQL**	**MySQL**
Anzahl der Zeilen	COUNT(*)		
Anzahl der Einträge ungleich NULL	COUNT([DISTINCT | ALL[28]] *Ausdruck*)		
Durchschnitt	AVG(*Ausdruck*)		
Minimum	MIN(*Ausdruck*)		
Maximum	MAX(*Ausdruck*)		
Summe	SUM(*Ausdruck*)		
Standardabweichung	STDDEV	STDDEV	STDDEV (STD)
Varianz	VARIANCE	VARIANCE	
Binäre OR-Verknüpfung			BIT_OR
Binäre AND-Verknüpfung			BIT_AND

In PostgreSQL können sogar neue benutzerdefinierte Aggregatfunktionen erstellt werden.

7.12 Sequenzen

Sequenzen sind Zähler, die mit einem bestimmten Wert starten und hinauf- oder hinunterzählen können. Sie erleichtern das automatische Erzeugen von eindeutigen, numerischen Primärschlüsseln, indem für neu eingefügte Datensätze ein Wert von der Sequenz angefordert und gleichzeitig weitergezählt wird. Alternativ zu explizit erstellten Sequenzen, gibt es auch die Möglichkeit, das Schlüsselattribut mit dem Pseudo-Sequenzdatentyp (SERIAL) anzulegen. Im Hintergrund wird dann implizit eine Sequenz erstellt und der DEFAULT-Wert des Attributes von der Sequenz bezogen.

[28] Das Schlüsselwort ALL wird von MySQL nicht angeboten.

7.12.1 SAP DB

Definition 7.1 CREATE SEQUENCE-Syntax

```
CREATE SEQUENCE [Besitzer.]Sequenzname [INCREMENT BY Delta]
[START WITH Startwert] [MAXVALUE Maximum | NOMAXVALUE]
[MINVALUE Minimum| NOMINVALUE] [CYCLE | NOCYCLE]
[CACHE Größe | NOCACHE]
```

CREATE SEQUENCE erzeugt eine Sequenz, die mit dem Startwert zu zählen beginnt und bei jedem Schritt das Delta dazuaddiert. Ein negatives Delta bewirkt, dass die Sequenz abwärts zählt. Es kann ein Maximum bzw. Minimum angegeben werden, bis zu dem gezählt werden soll. Wird dieses erreicht und ist CYCLE angegeben, fängt der Zähler beim anderen Ende der Zahlengerade wieder zu zählen an. Der Cache wird dazu verwendet, mehrere Werte auf einmal anzufordern, im Hauptspeicher zu halten und bei Bedarf zu vergeben. Auf die Sequenz wird mit den Ausdrücken Sequenzname.CURRVAL und Sequenzname.NEXTVAL zugegriffen. CURRVAL liefert die zuletzt mit NEXTVAL angeforderte Sequenznummer zurück, während NEXTVAL den aktuellen Zähler übergibt und gleichzeitig weiterzählt.

Beim Anlegen von neuen Tabellen kann genau ein Attribut vom Pseudo-Datentyp SERIAL sein. SERIAL ist in SAP DB ein Synonym für FIXED(10) DEFAULT SERIAL und bewirkt, dass implizit eine für den Benutzer nicht sichtbare Sequenz erzeugt wird, die beim Anlegen von neuen Datensätzen, falls dem Attribut kein Wert[29] zugewiesen wird, diesem Attribut immer einen neuen Wert aus der Sequenz zuweist. Beim Pseudo-Datentyp SERIAL kann optional der Startwert in runden Klammern angegeben werden.

7.12.2 PostgreSQL

Definition 7.2 CREATE SEQUENCE-Syntax

```
CREATE [TEMP[ORARY]] SEQUENCE Sequenzname [INCREMENT Delta]
[START Startwert] [MAXVALUE Maximum] [MINVALUE Minimum]
[CYCLE] [CACHE Größe]
```

Die CREATE SEQUENCE-Syntax ist ähnlich der von SAP DB. Mit dem Schlüsselwort TEMPORARY können temporäre Sequenzen erzeugt werden, die nur während der Dauer

[29] Auch NULL oder der Wert 0 bewirkt, dass die Sequenz zum Zug kommt.

einer Session bestehen. Auf die Sequenz kann mittels der Funktionen NEXTVAL, CURRVAL und SETVAL zugegriffen werden. Es ist also auch möglich, den Zähler im Nachhinein zu verändern. Der Pseudo-Datentyp SERIAL erzeugt eine für den Benutzer sichtbare Sequenz mit dem Namen *Tabellenname_Spaltenname*_seq und setzt den DEFAULT-Wert der Spalte auf NEXTVAL(*Sequenzname*). Die Anzahl der SERIAL Attribute in einer Tabelle ist nicht beschränkt.

7.12.3 MySQL

MySQL bietet zwar keine Sequenzen an, dafür aber die Möglichkeit, Spalten mit dem Attribut AUTO_INCREMENT zu versehen. Wird nun ein neuer Datensatz eingefügt und die betreffende Spalte gar nicht, mit NULL oder dem Wert 0 initialisiert, wird ein im Hintergrund arbeitender Zähler verwendet. Den zuletzt eingefügten Wert kann man mit der Funktion LAST_INSERT_ID abfragen. Eine Besonderheit von AUTO_INCREMENT bei Verwendung in einem zusammengesetzten Primärschlüssel ist, dass der Zähler nur dann erhöht wird, falls der Präfix des Primärschlüssels ohne AUTO_INCREMENT nicht ausreichen würde, um den Datensatz eindeutig zu identifizieren. Bei jedem neuen Präfix wird wieder von eins zu zählen begonnen. Dies kann dazu verwendet werden, um Daten in geordnete Gruppen einzufügen.

Beispiel 7.1 AUTO_INCREMENT bei zusammengesetzten Schlüsseln [3]

Gruppe	ID (AUTO_INCREMENT)	Name
Fisch	1	Lachs
Säugetier	1	Hund
Säugetier	2	Katze
Säugetier	3	Wal
Vogel	1	Pinguin

7.13 Constraints

Integritätsbedingungen (Constraints) sind Prädikate, die festlegen, ob ein Datenbankzustand semantisch Sinn macht oder nicht. Sie schränken beispielsweise den möglichen Wertebereich von Attributen ein oder sorgen für die Eindeutigkeit von Schlüsselattributen. Aufgabe des DBMS ist es, das Hinzufügen, Modifizieren oder Löschen von Datensätzen zu verhindern, wenn dies zu einer Verletzung einer Integritätsbedingung führen würde. Der SQL-Standard [12] sieht folgende Arten von Constraints vor:

- **Primär- und Sekundärschlüssel-Bedingungen**
  Diese Bedingungen stellen die Eindeutigkeit von Schlüsseln sicher. Die OSDB legen bei der Definition von Primär- und Sekundärschlüsseln automatisch Indexe an.

Beispiel 7.2 Tabelle Buch mit der Signatur (BuchNr) als Primärschlüssel und der ISBN-Nummer als Sekundärschlüssel

```
... PRIMARY KEY (BuchNr), UNIQUE (ISBN) ...
```

- **Fremdschlüssel-Bedingungen (referenzielle Integrität)**
  Darunter verstehen wir die Bedingung, dass die durch einen Fremdschlüssel referenzierten Datensätze auch tatsächlich existieren. Problematisch wird es, falls referenzierte Datensätze gelöscht oder deren Primärschlüssel nachträglich geändert werden. In solchen Fällen entscheidet eine etwaige Lösch- oder Änderungsregel der Fremdschlüssel-Bedingung, wie das DBMS darauf reagieren soll. Standardmäßig werden solche problematischen Aktionen verhindert (Regel: RESTRICT bzw. NO ACTION) oder die referenzierten Datensätze mitgelöscht oder deren Fremdschlüssel auf den geänderten Wert aktualisiert (Regel: CASCADE). Weitere Regeln sind SET DEFAULT und SET NULL. In SAP DB kann der Benutzer nur eine Löschregel angeben, aber keine Änderungsregel. MySQL unterstützt Fremdschlüssel-Bedingungen überhaupt nur bei InnoDB-Tabellen.

- **NOT NULL-Bedingungen**
  Einem Attribut mit der NOT NULL-Bedingung kann die Nullmarke NULL nicht zugewiesen werden.

- **Spalten- und Tabellenbedingungen**
  Mit Hilfe von CHECK-Bedingungen können beliebige SQL-Prädikate dazu eingesetzt werden, Attributwerte zu überprüfen. SAB DB und PostgreSQL unterstützen aber leider keine SELECT-Anweisungen in solchen Bedingungen, was dazu führt, dass nur auf Tupelebene Attribute in Beziehung gebracht werden können. MySQL hingegen kennt zwar die Syntax von CHECK, ignoriert aber solche Angaben bei der Erstellung von neuen Tabellen.

Beispiel 7.3 Bücher sollen erst nach ihrer Entlehnung fällig werden

```
... CHECK (faelligam > Entlehndat) ...
```

- **Domain-Bedingungen**
  SQL sieht vor, einen Datentyp mit den dazugehörigen Constraints als Domain zu definieren, die dann später bei Spaltendefinitionen verwendet werden kann. Dies wird von SAP DB und PostgreSQL angeboten, wobei in PostgreSQL derzeit noch keine CHECK-Bedingung in die Domain-Definition aufgenommen werden kann.

Beispiel 7.4 Definition einer Domain (SAP DB)

```
CREATE DOMAIN Geschlecht CHAR(1) DEFAULT '?'
CHECK (Geschlecht IN ('F', 'M', '?'))
```

- **Allgemeine Bedingungen**
  Globale CHECK-Bedingungen beliebiger Komplexität (SQL-Anweisungen mit Sub-SELECTs) können gemäß SQL-Standard mit dem Befehl CREATE ASSERTION definiert werden. Dieser Befehl wird derzeit jedoch von keiner OSDB zur Verfügung gestellt. Stattdessen weisen wir darauf hin, dass mit dem Triggerkonzept (siehe Abschnitt 7.16) komplexere Bedingungen kontrolliert werden können.

Das Verzögern von Integritätsbedingungen auf das Transaktionsende ist nur in PostgreSQL möglich. Dort allerdings nur bei Fremdschlüssel-Bedingungen.

7.14 Sub-SELECTs

Verschachtelte SELECT-Anweisungen (Sub-SELECTs) werden nur in SAP DB und PostgreSQL verarbeitet. SAP DB unterstützt keine Sub-SELECTs im SELECT-Kopf. In MySQL muss man versuchen, verschachtelte SELECT-Anweisungen mit Hilfe von (temporären) Hilfstabellen und Joins aufzulösen.

Beispiel 7.5 Die aktivsten Leser herausfinden

```
SELECT Name FROM Leser JOIN Ausleihe USING (LeserNr)
GROUP BY Leser.LeserNr, Name
HAVING Count(*) >= ALL
(SELECT Count(*) FROM Ausleihe GROUP BY LeserNr)
```

Im Falle von MySQL nehmen wir an, dass das Ergebnis des Sub-SELECTs in der Tabelle LeserStatistik (LeserNr, Name, Anzahl) steht. Dann lässt sich die Abfrage umformulieren in:

```
SELECT L1.Name
FROM LeserStatistik L1 LEFT JOIN LeserStatistik L2
ON L1.Anzahl < L2.Anzahl WHERE L2.Anzahl IS NULL
```

7.15 Stored Procedures

SAB DB und PostgreSQL erlauben es, benutzerdefinierte Funktionen und Prozeduren zu erstellen, die serverseitig gespeichert und ausgeführt werden. Mit Hilfe von Stored Procedures ist es möglich, die Applikationslogik immer mehr von den Applikationen direkt in das Datenbanksystem zu integrieren. Diese Tendenz bringt folgende Vorteile mit sich:

- **Abstraktion von SQL-Befehlen auf logische Einheiten**
  Stored Procedures können als Schnittstelle zwischen den Applikationen und dem Datenbankmodell fungieren. Typische Geschäftsfälle können als Stored Procedure abgebildet werden und erleichtern den Zugriff aus den Applikationen.

- **Höhere Geschwindigkeit**
  Der Kommunikationsaufwand zwischen Client und Server sinkt in der Regel, da eine Stored Procedure mehrere SQL-Befehle beinhalten kann, deren Ergebnisse direkt am Server weiterverarbeitet werden können.

- **Portabilität**
  Stored Procedures werden gewöhnlich in einer eigenen DBMS abhängigen prozeduralen Sprache entwickelt, die auf allen Plattformen ausgeführt werden kann, auf denen auch das DBMS läuft.

- **Bessere Wartbarkeit**
  Durch die Zentralisierung in Richtung DBMS, braucht im Idealfall nur die Stored Procedure angepasst werden, nicht aber die Applikationen, die diese verwenden.

Nachteile können sich bei rechenintensiven Funktionen ergeben, da sich in diesem Fall die Rechenlast am Datenbankserver kumuliert.

7.15.1 SAP DB

SAP DB erweitert den Sprachumfang von reinen SQL-Anweisungen und bietet für das prozedurale Programmieren von Stored Procedures Variablen, Kontrollanweisungen und Ausnahmebehandlung an. Es können alle Datentypen außer LONG als Aus- und Eingangsparameter verwendet werden. SAP DB unterstützt Sub-Transaktionen, um Stored Procedures im Fehlerfall wieder rückgängig zu machen.

Das folgende Beispiel, das dazu dient, eine Buchrückgabe durchzuführen und etwaige Spesen zu verrechnen, soll einen kleinen Eindruck von benutzterdefinierten Prozeduren in SAP DB

vermitteln. Im VAR-Abschnitt werden lokale Variablen deklariert, die später mit FETCH initialisiert werden. Tritt im TRY-Block ein Fehler auf, wird der STOP-Befehl am Ende der Prozedur ausgeführt, der den Fehlercode des Fehler verursachenden Befehls ($RC) und einen Text an den Aufrufer zurückgibt.

Beispiel 7.6 Stored Procedure in SAP DB

```
CREATE DBPROC buch_rueckgabe (IN BuchNr INTEGER) AS
VAR  Entlehndat DATE;
     faelligam DATE; Spesen FIXED(10,2); AktuellesDatum DATE;
TRY
     SET AktuellesDatum = DATE;
     SELECT Entlehndat, faelligam FROM Test.Ausleihe
     WHERE BuchNr = :BuchNr ORDER BY Entlehndat DESC;
     FETCH INTO :Entlehndat, :faelligam;
     /* Rückgabe in die Tabelle eintragen */
     INSERT INTO Test.Rueckgabe
     VALUES (:BuchNr, :Entlehndat, :AktuellesDatum);
     /* etwaige Spesen berechnen */
     SET Spesen = DATEDIFF(AktuellesDatum, faelligam) * 0.15;
     IF Spesen > 0 THEN
     BEGIN
          SET Spesen = Spesen + 2.00; /* 2 EUR Mahngebühr */
          INSERT INTO Test.Mahnung
          VALUES (:BuchNr, :Entlehndat, :Spesen);
     END;
CATCH
     STOP($RC, 'Fehler in buch_rueckgabe');
```

Beispiel 7.7 Aufruf von *buch_rueckgabe* in SAP DB mit COMMIT nach erfolgreicher Ausführung

```
CALL buch_rueckgabe(123) WITH COMMIT
```

7.15.2 PostgreSQL

In PostgreSQL können Stored Procedures, hier Funktionen genannt, je nach Wunsch des Benutzers in verschiedenen prozeduralen Programmiersprachen geschrieben werden. Mit Hilfe von dynamischen Modulen ist es auch möglich, einen eigenen Interpreter für eine benutzerdefinierte Sprache in PostgreSQL zu integrieren. Standardmäßig bietet PostgreSQL folgende prozedurale Sprachen für die Entwicklung von benutzerdefinierten Funktionen an:

- PL/pgSQL
- PL/Tcl
- PL/Perl
- PL/Python

Wir gehen hier nur auf die PostgreSQL eigene prozedurale Programmiersprache PL/pgSQL ein, die der PL von SAP DB oder etwa Oracle PL/SQL ähnlich ist. Neben Variablenunterstützung, Kontrollanweisungen und Mechanismen, um Datensätze der Ergebnismenge der Reihe nach zu bearbeiten, punktet PL/pgSQL vor allem mit der Möglichkeit, dass Funktionen auch ganze Ergebnisrelationen zurückliefern können. PostgreSQL optimiert – falls erwünscht – Funktionsaufrufe, indem zum Beispiel idente Aufrufe nur einmal evaluiert werden. Im folgenden Beispiel zeigen wir, wie man die Funktion buch_rueckgabe in PostgreSQL implementieren könnte. Die Argumente stehen in den speziellen Variablen $1, $2, $3, usw. Variablen des Typs RECORD können mehrere Attribute eines Datensatzes aufnehmen. Mit so genannten Typreferenzen auf Typen im Datenbankmodell wird die Funktion flexibler in Bezug auf Schemaänderungen. Um PL/pgSQL in einer Datenbank zu verwenden, muss man vorher mit dem Shell-Skript *createlang* die Sprache definieren. PostgreSQL bietet nicht die Möglichkeit, Funktionen in Sub-Transaktionen zu verpacken.

Beispiel 7.8 Benutzerdefinierte Funktion in PostgreSQL[30]

```
CREATE FUNCTION buch_rueckgabe (Buch.BuchNr%TYPE)
RETURNS DATE AS '
DECLARE
     Details RECORD;
     Spesen Mahnung.Spesen%TYPE; /* Typreferenz */
     AktuellesDatum DATE;
BEGIN
```

[30] Aus Gründen der Übersichtlichkeit verzichten wir in diesem Beispiel auf eine explizite Fehlerbehandlung.

```
     AktuellesDatum := CURRENT_DATE;
     SELECT INTO Details Entlehndat, faelligam FROM Ausleihe
     WHERE BuchNr = $1 ORDER BY Entlehndat DESC;
     /* Rückgabe in die Tabelle eintragen */
     INSERT INTO Rueckgabe VALUES
     ($1, Details.Entlehndat, AktuellesDatum);
     /* etwaige Spesen berechnen */
     Spesen := (AktuellesDatum - Details.faelligam) * 0.15;
     IF Spesen > 0 THEN
          Spesen := Spesen + 2; /* 2 EUR Mahngebühr */
          INSERT INTO Mahnung VALUES
          ($1, Details.Entlehndat, Spesen);
     END IF;
     RETURN AktuellesDatum;
END
' LANGUAGE 'plpgsql';
```

Beispiel 7.9 Aufruf der Funktion *buch_rueckgabe* in PostgreSQL

```
SELECT buch_rueckgabe(123);
```

7.16 Trigger

Trigger sind Prozeduren, die bei einer Datenmanipulation (INSERT-, UPDATE- oder DELETE-Anweisung) automatisch ausgeführt werden, um zum Beispiel die Korrektheit der Operation zu prüfen, redundante Information in der Datenbank anzugleichen oder weitere Operationen anzustoßen, wie zum Beispiel ein Änderungslog zu führen. Mit Triggern können transitionale und dynamische Integritätsbedingungen realisiert werden (vgl. Abschnitt 1.5). Trigger beziehen sich immer auf eine Tabelle und werden entweder vor der Aktualisierung des Datensatzes oder danach ausgeführt. Mit Hilfe der speziellen Variablen OLD und NEW kann bei einer Aktualisierung auf die alten und neuen Werte eines Datensatzes zugegriffen werden. Trigger auf SQL-Anweisungsebene, die also nicht bei jedem betroffenen Datensatz, sondern nur einmal für die gesamte Anweisung „feuern“, werden weder von SAP DB noch von PostgreSQL angeboten. MySQL fehlt es noch an einer Triggerimplementierung.

7.16.1 SAB DB

In SAB DB kann pro Tabelle für jedes der drei Ereignisse (INSERT, UPDATE und DELETE) nur ein Trigger definiert werden. Dafür kann das Auslösen des Triggers an eine zusätzlich angegebene Bedingung geknüpft werden, die jedoch keine Sub-SELECTs beinhalten darf. Update-Trigger können auf das Aktualisieren von bestimmten Spalten eingeschränkt werden. Es können nur Trigger definiert werden, die nach erfolgreicher Ausführung der SQL-Anweisung ausgeführt werden.
Das folgende Beispiel zeigt, wie man einen Trigger definiert, der beim Löschen von Rückgaben automatisch eventuell vorhandene Mahnungen entfernt.

Beispiel 7.10

```
CREATE TRIGGER mahnung_entfernen FOR Rueckgabe AFTER DELETE
EXECUTE (
DELETE FROM Test.Mahnung M WHERE
M.BuchNr = :OLD.BuchNr AND M.Entlehndat = :OLD.Entlehndat; )
```

7.16.2 PostgreSQL

Trigger in PostgreSQL sind eine benutzerdefinierte Funktion in einer beliebigen prozeduralen Sprache, denen man sogar Argumente übergeben kann. Pro Tabelle und Ereignis können beliebig viele Trigger definiert werden, die nacheinander entweder vor der eigentlichen Datenmanipulation oder danach ausgeführt werden. Das Update-Ereignis kann nicht wie in SAP DB auf bestimmte Spalten eingeschränkt werden. Das Beispiel 7.10 könnte man in PostgreSQL wie folgt umsetzen:

Beispiel 7.11

```
CREATE FUNCTION mahnung_entfernen () RETURNS OPAQUE
AS 'BEGIN DELETE FROM Mahnung WHERE BuchNr = OLD.BuchNr
AND Entlehndat = OLD.Entlehndat; RETURN NULL; END;'
LANGUAGE 'plpgsql';
CREATE TRIGGER rueckgabe_delete AFTER DELETE ON Rueckgabe
FOR EACH ROW EXECUTE PROCEDURE mahnung_entfernen();
```

7.17 Ausnahmebehandlung (Exceptions)

Sowohl SAB DB als auch PostgreSQL (PL/pgSQL) bieten in ihren prozeduralen Programmiersprachen eine Ausnahmebehandlung an. Es können explizit Exceptions erzeugt werden. In SAP DB kann mit den Variablen $RC (Fehlercode), $ERRMSG (Fehlermeldung) und $COUNT (Anzahl der Zeilen in der Ergebnistabelle) der Erfolg einer Anweisung überprüft werden. In PostgreSQL gibt es die boolsche Variable FOUND und den Befehl GET DIAGNOSTICS, um die Anzahl der Ergebniszeilen in Erfahrung zu bringen. Einen TRY CATCH-Block wie in SAP DB sucht man aber im Manual von PL/pgSQL vergeblich.

7.18 Cursor-Unterstützung

Um die Kluft zwischen den mengenorientierten Ergebnissen von SQL-Abfragen und den Anforderungen von prozeduralen Sprachen, deren Variablen nur einen Wert oder maximal einen Datensatz speichern können, zu überwinden, sind im SQL-Standard so genannte Cursor vorgesehen, um zeilenweise in der Ergebnismenge zu manövrieren. Ein Cursor basiert auf einer beliebigen SQL-Anweisung. Den Cursor kann man sich wie einen Zeiger vorstellen, der auf einen bestimmten Datensatz in der Ergebnisrelation zeigt. Mit dem Befehl FETCH kann ein Datensatz eingelesen und der Zeiger weiterbewegt werden. Der SQL-Standard [12] sieht folgende Operationen vor:

- DECLARE CURSOR: Cursor anlegen.
- OPEN: Cursor öffnen (= SQL-Anweisung ausführen).
- FETCH: Datensatz einlesen.
- UPDATE und DELETE (positioniert): Aktuellen Datensatz ändern / löschen.
- CLOSE: Cursor wieder schließen.

Cursor waren ursprünglich nur für Embedded SQL vorgesehen. In PostgreSQL unterscheidet sich die Cursor-Untersützung je nach dem, ob man im interaktiven Modus arbeitet oder sie in der prozeduralen Programmiersprache PL/pgSQL verwendet. Im interaktiven Modus gibt es kein OPEN und CLOSE. Es reicht aus, den Cursor zu deklarieren. In PL/pgSQL gibt es neben dem Cursor-Konzept auch noch eine spezielle FOR-Schleife, um sequenziell eine SQL-Ergebnismenge zu bearbeiten. In SAP DB stehen alle oben aufgezählten Befehle zur Verfügung. Mit einem speziellen SELECT-Befehl, der einer Abfrage einen Namen gibt, kann man sich sogar das Anlegen, Öffnen und Schließen eines Cursors sparen. Der FETCH-Befehl wird in diesem Fall, genauso – als ob es sich um einen Cursor handeln würde – verwendet, nur dass nicht der Cursor-Name, sondern der Name der Abfrage angegeben wird. SAP DB erlaubt

unter folgenden Bedingungen das Modifizieren und Löschen des aktuellen Datensatzes, auf den ein Cursor zeigt:

- Der Cursor wurde mit der Option FOR UPDATE angelegt.
- In der Cursor-Definition kommt kein UNION, INTERSECT oder EXCEPT vor.
- In der FROM-Klausel der SELECT-Anweisung steht nur eine Basistabelle oder eine modifizierbare View (siehe Abschnitt 7.20).
- In der Abfrage ist kein DISTINCT oder GROUP BY.
- Es werden keine Aggregatfunktionen verwendet.

Trifft dies zu, können in SAP DB mit den Befehlen UPDATE und DELETE unter Angabe von WITH CURRENT *Cursor-Name bzw. SELECT-Name* der aktuelle Datensatz geändert oder gelöscht werden. In PostgreSQL ist dies nicht möglich. MySQL hat auf SQL-Ebene noch keine Cursor-Unterstützung im Programm.

Beispiel 7.12 Anlegen eines Cursors, um die Tabelle Buch zeilenweise auszulesen

```
DECLARE CBuch CURSOR FOR SELECT BuchNr,Autor,Titel FROM Buch;
```

7.19 Rekursive SQL-Anweisungen

SAP DB bietet überraschenderweise eine dem SQL/99-Standard angelehnte Rekursion in SQL-Anweisungen an. Dies allerdings nur in Cursor-Definitionen und nicht als eigenständiger SQL-Befehl. Die Syntax der rekursiven Anweisung ist folgende:

Definition 7.3 Rekrusvie Cursor-Defintion in SAP DB

```
DECLARE Name CURSOR FOR
WITH RECURSIVE Referenztabellenname (Spaltennamen) AS
( initiale Abfrage UNION ALL rekursive Abfrage )
abschließende SELECT-Abfrage
```

Die Ausführung läuft in etwa so ab: Zunächst wird die initiale Abfrage ausgewertet. Das Ergebnis wird in die Tabelle mit dem Referenztabellennamen geschrieben. Danach wird diese Tabelle solange mit dem Ergebnis der rekursiven Abfrage erweitert, bis die rekursive Abfrage keine neuen Einträge mehr liefert, das heißt eine leere Tabelle als Antwort zurückgibt. Zum Schluss wird die Tabelle mit der abschließenden SELECT-Abfrage noch für die Ausgabe aufbereitet. In der rekursiven und der abschließenden Abfrage kann natürlich die aktuelle Ergebnistabelle referenziert werden.

Um den Vorgang besser zu verstehen, wollen wir zum Abschluss noch ein kleines Beispiel angeben, das einer Tabelle Kind_von (Kind, Elternteil) zu Grunde liegt und als Ergebnis die Relation Vorfahre_von (a,b) produziert, sodass a Vorfahre von b ist.

Beispiel 7.13 Rekursive SQL-Anweisung [12]

```
DECLARE CVorfahre_von CURSOR FOR
WITH RECURSIVE Vorfahre_von (Vorfahre, Nachfahre)
AS (SELECT Elternteil, Kind FROM Kind_von UNION ALL
SELECT V.Elternteil, K.Kind FROM Vorfahre_von V, Kind_von K
WHERE V.Kind = K.Elternteil)
SELECT * FROM Vorfahre_von
```

7.20 Views

Views sind virtuelle Tabellen, die das Ergebnis einer beliebigen SQL-Anweisung repräsentieren. Sie werden dazu verwendet, um komplexen SQL-Anweisungen bzw. Joins einen Namen zu geben, die Sicht auf Basistabellen einzuschränken (zum Beispiel Projektion oder Selektion auf die relevanten Spalten oder Zeilen) oder um eine Schnittstelle zum Datenbankmodell zu definieren. Sowohl SAP DB als auch PostgreSQL erlauben Views zu definieren. Sie unterscheiden sich in der Möglichkeit, Datenmanipulationen auf Views auszuführen. PostgreSQL erstellt für jede View implizit eine Dummy-Tabelle mit einer SELECT-Transformationsregel (siehe nächster Abschnitt) und erlaubt deshalb per se keine Updates. In SAP DB können Datensätze aktualisiert, gelöscht und hinzugefügt werden, falls folgende Bedingungen für die zugrundeliegende SQL-Anweisung zutreffen:

- Es wird kein DISTINCT verwendet.
- Die Datensätze werden nicht mit GROUP BY gruppiert.
- Die SQL-Anweisung enthält keine Sub-SELECTs.
- Es kommt kein äußerer Verbund zum Einsatz.
- Bei inneren Verbunden müssen die Tabellen mit Fremdschlüssel-Bedingungen zusammenhängen und die View muss so konstruiert sein, um die Kette der referenzierten Tabellen rekonstruieren zu können. In diesem Zusammenhang müssen noch weitere Bedingungen erfüllt sein, die im Manual nachgelesen werden können.
- Spalten, die hinzugefügt oder geändert werden sollen, müssen direkt mit einer Spalte aus einer Basistabelle verbunden sein.
- Um Datensätze einfügen zu können, müssen alle obligatorischen (NOT NULL) Attribute der referenzierten Basistabellen in der View enthalten sein.

Falls die Option WITH CHECK OPTION beim Erstellen der View angegeben wird, wird das Hinzufügen und Ändern von Datensätzen nur dann gestattet, wenn der aktualisierte bzw. neue Datensatz auch mit der View gesehen (das heißt selektiert) werden kann.

Beispiel 7.14

```
CREATE VIEW LeserStatistik (Name, Ausleihen) AS
SELECT Name, COUNT(BuchNr) FROM Leser LEFT JOIN Ausleihe ON
(Leser.LeserNr = Ausleihe.LeserNr)
GROUP BY Ausleihe.LeserNr, Name ORDER BY Name;
```

Bemerkung: In SAP DB können in Views keine ORDER BY-Klauseln verwendet werden.

7.21 Rules

PostgreSQL wartet mit einem ausgereiften Transformationssystem für SQL-Anweisungen auf. Bevor eine Anweisung vom Executor ausgeführt wird, werden eventuell vorhandene Transformationsregeln angewendet und der QEP transformiert. Der Befehl zum Definieren von Transformationsregeln sieht wie folgt aus:

Definition 7.4 CREATE RULE-Syntax

```
CREATE RULE Name AS ON Ereignis TO Tabelle [WHERE Bedingung]
DO [INSTEAD] Aktion(en)
```

Ereignisse können SELECT, INSERT, UPDATE oder DELETE auf eine bestimmte Tabelle (oder View) sein. Die Regel kann weiters an eine Bedingung geknüpft werden und führt die angegebene(n) Aktion(en) entweder an Stelle des ursprünglichen Befehls (bei der Angabe von INSTEAD) oder zusätzlich zu der die Regel auslösenden SQL-Anweisung aus. Als Aktion kann entweder eine SQL-Anweisung oder NOTHING (um gar nichts zu tun) angegeben werden. Außer bei SELECT-Regeln können auch mehrere Aktionen angegeben werden, die sequenziell ausgeführt werden.

Die View aus Beispiel 7.14 wird in PostgreSQL intern in folgende SELECT-Transformationsregel umgewandelt:

```
CREATE RULE "_RETURN" AS ON SELECT TO LeserStatistik
DO INSTEAD SELECT Name, COUNT(BuchNr) AS Ausleihen FROM Leser
LEFT JOIN Ausleihe ON (Leser.LeserNr = Ausleihe.LeserNr)
GROUP BY Ausleihe.LeserNr, Name ORDER BY Name;
```

Mit Hilfe von Regeln ist es möglich, dort wo es Sinn macht, Views modifizierbar zu machen, indem INSERT-, UPDATE- und DELETE-Regeln definiert werden. Ähnlich wie bei Triggern können in diesen Fällen mit den Record-Variablen OLD und NEW auf die alten und neuen Datensätze zugegriffen werden. Im Gegensatz zu den zeilenorientierten Triggern, operieren Regeln aber auf der ganzen Anweisung.

7.22 Modifikation des Datenbankschemas

Im Laufe der Zeit kann es notwendig werden, Modifikationen im Datenbankschema vorzunehmen, wie zum Beispiel neue Attribute einer Tabelle hinzuzufügen. Bei einigen Modifikationen ist darauf zu achten, dass Referenzen existieren könnten, die entweder die beabsichtigte Änderung verhindern oder dann nicht mehr gültig wären. Modifikationen werden in SQL mit dem Befehl ALTER TABLE durchgeführt, der in allen drei OSDB ähnlich angewendet wird. MySQL erzeugt bei jeder Schemaänderung eine temporäre Tabelle, die dann die ursprüngliche Tabelle ersetzt.

In folgender Tabelle wollen wir die Möglichkeiten dieses Befehls gegenüberstellen:

Tabelle 7.11 Vergleich von ALTER TABLE

Modifikation	**SAP DB**	**PostgreSQL**	**MySQL**
Umbenennen von Tabellen	Ja [31]	Ja	Ja
Umbenennen von Spalten	Ja [31]	Ja	Ja
Hinzufügen von Spalten	Ja	Ja	Ja
Default-Wert beim Anlegen von neuen Spalten[32]	Nein	Nein	Ja
Reihenfolge der Spalten verändern	Nein	Nein	Ja
Anlegen eines Primärschlüssels	Ja	Ja	Ja
Anlegen von Sekundärschlüsseln	Nein	Ja	Ja
Hinzufügen von Fremdschlüssel-Bedingungen	Ja	Ja	Ja
Hinzufügen von CHECK-Bedingungen	Nein	Ja	Nein
Modifizieren des Spaltentyps[33]	Ja	Nein	Ja
Modifizieren der NOT NULL-Bedingung	Ja	Ja	Ja
Modifizieren von CHECK-Bedingungen	Nein	Nein	Nein
Modifizieren des DEFAULT-Wertes	Ja	Ja	Ja
Löschen von Spalten	Ja	Ja	Ja

[31] Umbenannt wird mit den Befehlen RENAME TABLE und RENAME COLUMN.

[32] SAP DB und PostgreSQL setzen den Wert einer neu hinzugefügten Spalte immer auf NULL.

[33] Dies funktioniert nur dann, wenn sich die bereits bestehenden Werte in den neuen Typ konvertieren lassen.

Automatisches Löschen von verknüpften Objekten (wie Views oder Constraints)	Nein	Ja	Nein
Löschen von Integritätsbedingungen	Ja	Ja	Nein
Löschen des Primärschlüssels	Ja	Ja	Ja
Manuelles Setzen von Spaltenstatistiken	Nein	Ja	Nein
Speicherart von Spalten bestimmen (zum Beispiel Spalte komprimiert speichern)	Nein	Ja	Nein
Physikalisches Umsortieren der Daten	Nein	Ja [34]	Ja
Aktualisieren von Sekundärindexen aus- bzw. einschalten	Nein	Nein	Ja
Ändern des Tabellentyps (in MySQL) und weiterer Tabellenoptionen	Nein	Nein	Ja

Beispiel 7.15 Einfügen einer Spalte Geburtsdatum in die Tabelle Leser

```
ALTER TABLE Leser ADD COLUMN Geburtsdat DATE;
```

7.23 Temporäre Tabellen

Alle drei OSDB unterstützen das Anlegen von temporären Tabellen. Temporäre Tabellen sind Tabellen, die nur in einer Sitzung bekannt sind und am Ende der Sitzung automatisch gelöscht werden. In PostgreSQL kann angegeben werden, dass die temporäre Tabelle bei jedem COMMIT automatisch geleert werden soll. MySQL hat eine gravierende Einschränkung in Bezug auf temporäre Tabellen. Diese können in einer FROM-Klausel nur einmal verwendet werden. Damit sind Joins mit sich selbst nicht möglich.

Beispiel 7.16 Anlegen einer temporären Tabelle basierend auf dem Ergebnis einer SQL-Abfrage

```
/* SAP DB */
CREATE TABLE TEMP.Autoren AS SELECT DISTINCT Autor FROM Buch

/* PostgreSQL */
CREATE TEMPORARY TABLE Autoren AS SELECT DISTINCT Autor FROM Buch;

/* MySQL */
CREATE TEMPORARY TABLE Autoren SELECT DISTINCT Autor FROM Buch;
```

[34] Mit dem Befehl CLUSTER kann eine Tabelle gemäß einem Index pyhsikalisch umsortiert werden.

7.24 Objekt-relationaler Ansatz

Das objektorientierte Paradigma hat sich in konventionellen Programmiersprachen bereits durchgesetzt. In die Datenbankwelt haben objektorientierte Konzepte zumindest teilweise schon Einzug gehalten. Vor allem PostgreSQL definiert sich als objekt-relationale Datenbank (ORDBMS). Dies machen folgende Eigenschaften aus, die es von einer reinen relationalen Datenbank unterscheidet:

- Vererbung von Tabellen (auch Mehrfachvererbung)
- Stored Procedures und Trigger
- Hoher Grad an Erweiterbarkeit: Benutzerdefinierte Datentypen, Operatoren, Funktionen, Aggregatfunktionen, Typkonvertierungen, prozedurale Sprachen, …
- Überladen von Funktionen
- Array-Datentyp

Besonders interessant ist das Vererben von Tabellen. Dies möchten wir anhand eines Beispiels zeigen. Wir definieren eine Tabelle Person, die personenbezogene Daten, wie Name, Geburtsdatum und Geschlecht enthält. Die bereits bekannte Tabelle Leser soll die Attribute von Person erben und zusätzlich noch ein Adressfeld aufweisen.

Beispiel 7.17 Vererbung in PostgreSQL

```
CREATE TABLE Person (
     id INTEGER PRIMARY KEY,  -- eindeutiger Primärschlüssel
     Vorname VARCHAR(50) NOT NULL,
     Nachname VARCHAR(50) NOT NULL,
     Geschlecht CHAR(1) NOT NULL DEFAULT '?'
     CHECK Geschlecht IN ('F', 'M', '?'),
     Geburtsdat DATE
);

CREATE TABLE Leser (
     Adresse VARCHAR(200)
) INHERITS(Person);
```

```
SELECT * FROM Person; -- zeigt alle Personen an (auch Leser)
SELECT * FROM ONLY Person; -- nur Tabelle Person anzeigen
```

7.25 Anfallende Wartungsarbeiten

Für den praktischen Einsatz von Datenbanksystemen ist es relevant, welche Wartungsarbeiten während des laufenden Betriebes durchzuführen sind. In PostgreSQL ist die bereits angesprochene Vaccum-Routine in regelmäßigen Abständen aufzurufen. Falls große Datenmengen gelöscht werden, ist in MySQL ein Aufruf von OPTIMIZE in Erwägung zu ziehen, um den belegten Speicherplatz zurückzugewinnen. SAP DB kennt diese Probleme zwar nicht, dafür muss eine fixe Größe beim Erstellen der Data Devspaces angegeben werden, was dazu führt, dass bei zu klein dimensionierten Werten ein häufiges Hinzufügen von Devspaces notwendig wird. Zu den Wartungsarbeiten im weiteren Sinn zählen auch das regelmäßige Sichern der Daten sowie die periodische Aktualisierung der Tabellenstatistik für Indexe. Diese Wartungsarbeiten wurden in Abschnitt 6.2.3 und 4.2 beschrieben.

7.26 Verfügbare Client-Tools

So wie unter UNIX üblich ist die primäre Schnittstelle für alle Administrationsaufgaben die Kommandozeile. Die drei OSDB bieten eine Reihe von Kommandozeilenbefehlen an, um die Datenbank zu verwalten, zu sichern und interaktive SQL-Sitzungen abzuhalten. Die CLI-Tools können auch von entfernten Rechnern aufgerufen werden und stellen dann über TCP/IP eine Verbindung zum Datenbankserver her.

Für alle drei OSDB gibt es auch grafische Administrationsprogramme und Lösungen, um über das Web auf die Datenbank zuzugreifen. SAP DB stellt für die Microsoft Windows Plattform die Applikationen Database Manager und SQL Studio bereit. Plattformübergreifend kann auch ein Webinterface installiert werden, das zum Beispiel auf den frei verfügbaren Webserver Apache basiert. Für PostgreSQL und MySQL gibt es eine ganze Menge von Open-Source-Lösungen, um die Administration und Entwicklung zu vereinfachen. Dazu zählen Programme wie pgAccess, pgAdminII und phpPgAdmin für PostgreSQL[35] und MySQL Control Center, MySQL GUI und phpMyAdmin für MySQL[36], um nur einige wenige bedeutende Softwarepakete zu nennen.

---

[35] Eine Liste der erhältlichen Tools findet sich zum Beispiel unter http://gborg.postgresql.org.

[36] MySQL Control Center und MySQL GUI werden von MySQL AB entwickelt und auf der offiziellen Homepage zum Download bereitgestellt.

7.27 Programmierschnittstellen

Alle drei OSDB unterstützen den von Microsoft entwickelten Industriestandard ODBC, der sowohl auf der Windowsplattform als auch unter UNIX zum Einsatz kommt. Wir wollen eine kurze Übersicht über die verfügbaren Programmierschnittstellen und die unterstützten Skriptsprachen geben:

- **ODBC (Open Database Connectivity)**
  Mit Hilfe eines Treibers und eines ODBC-Treiber-Managers ist ein transparenter Zugriff auf ein Datenbanksystem möglich, das heißt durch die standardisierte Schnittstelle können Programme, die auf ODBC basieren, auf jede ODBC-kompatible Datenbank zugreifen.

- **JDBC (Java Database Connectivity)**
  JDBC ist die standardisierte Datenbankschnittstelle von Java. Alle drei OSDB bieten sowohl JDBC 1.0 und 2.0 (ab JDK 1.2) für ihre Datenbank an.

- **API – Bibliothek für die Programmiersprache C/C++**
  Der ODBC-Treiber, Embedded SQL und oft auch die Skriptsprachenunterstützung basiert auf einer proprietären C Bibliothek, die etliche Funktionen enthält, um sich mit der Datenbank zu verbinden, SQL-Anweisungen auszuführen und das Ergebnis zeilenweise zu verarbeiten. Damit ist diese Variante, auf die Datenbank zuzugreifen, die direkteste und damit auch die schnellste. PostgreSQL und MySQL haben ihre API übersichtlich im Manual [2,3] dokumentiert, während sich SAP DB in ihrer Dokumentation dazu nicht äußert und lieber ODBC als erste Wahl ansieht.

- **Embedded SQL – Precompiler für C/C++**
  Der Standard Embedded SQL stellt eine Vereinfachung dar, in C/C++-Programmen SQL-Anweisungen auszuführen. Mit speziellen Preprozessor-Befehlen wird SQL-Code in die höhere Programmiersprache eingebettet. Mit einem speziellen Precompiler für das jeweilige DBMS wird der SQL-Code durch API-Aufrufe ersetzt. Einen solchen Precompiler gibt es für SAP DB und PostgreSQL.

- **Skriptsprachen: Perl, Python, PHP, Tcl**
  Neben den klassischen Programmiersprachen C, C++ und Java sind die DBMS-Hersteller auch bemüht, populäre Skriptsprachen wie Perl, Python oder PHP zu unterstützen. Die ersten drei Skriptsprachen können auf alle drei OSDB zugreifen, während Tcl nur von PostgreSQL und MySQL explizit unterstützt wird.

7.28 SQL/92-Konformität

Der SQL/92-Standard definiert drei Sprachebenen der Konformität: Entry, Intermediate und Full SQL, während im neueren SQL/99[37] eher von Features gesprochen wird, die teilweise zu Paketen zusammengefasst wurden. Eine genaue Untersuchung der Konformität würde den Rahmen dieser Arbeit sprengen, da eine solche Analyse schwierig und wahrscheinlich auch nicht allzu nützlich wäre. Streng genommen unterstützen die drei OSDB nur SQL/92 Entry Level. PostgreSQL ist bemüht, auch Teile des SQL/99-Standards zu unterstützen und gibt im Manual zu jedem Befehl an, inwieweit der Befehl dem Standard folgt oder nicht. Wir verweisen weiters auf den Anhang C in [2], wo eine Liste der unterstützten und nicht unterstützten Funktionen von SQL/99 angegeben ist. Die größten Unterschiede liegen, einmal von nicht vorhandener Funktionalität abgesehen, in den prozeduralen Sprachen von benutzerdefinierten Funktionen und in den eigenen Erweiterungen der Datenbanksysteme. In der Regel wird also eine moderate Adaption des SQL-Codes von DBMS zu DBMS notwendig sein.

[37] Die offizielle Bezeichnung des Standards lautet ISO/IEC 9075:1999.

Kapitel 8

Zusammenfassung

In dieser Arbeit legten wir das Hauptaugenmerk auf den Vergleich und die Erläuterung der Konzepte der drei OSDB und erstellten im vorherigen Kapitel einen Kriterienkatalog für die Auswahl eines geeigneten Datenbanksystems. Abschließend wollen wir die betrachteten sowie noch einige weitere Aspekte, wie zum Beispiel die technischen Grenzwerte, in einer tabellarischen Form gegenüberstellen. In diesem Kapitel werfen wir weiters einen Blick auf Oracle, den Marktführer unter den kommerziellen Datenbanksystemen. Wir zeigen, was die freien OSDB von Oracle unterscheidet und zu guter Letzt versuchen wir, einen kleinen Ausblick auf die angekündigten Neuerungen der zukünftigen Versionen zu geben.

8.1 Tabellarischer Vergleich

In der folgenden Tabelle sollen die wichtigsten Merkmale kurz und bündig gegenübergestellt werden. Eine Beschreibung wurde entweder bereits in den vorangegangenen Kapiteln gegeben oder findet sich in den Benutzerhandbüchern [1,2,3].

Tabelle 8.1 Gegenüberstellung der wichtigsten Merkmale

	SAP DB	**PostgreSQL**	**MySQL**
Lizenz	GNU GPL	BSD-Lizenz	GNU GPL
Untersuchte Version	7.3.23	7.3.1	4.0.3-beta
(Offiziell) unterstützte Betriebssysteme	AIX, HP-UX, Linux, Solaris, Tru64, Windows	AIX, BSD/OS, FreeBSD, HP-UX, IRIX, Linux, MacOS X,	AIX, BSDI, FreeBSD, HP-UX, IRIX, Linux, MacOS X, NetBSD,

		NetBSD, OpenBSD, OpenServer, Solaris, Tru64, UnixWare, Windows (mit cygwin)	Novell Net-Ware, Open-BSD, OS/2, SunOS, Open-Server, OSF, QNX, Solaris, Tru64, Unix-War, Windows
DBMS-Typ	relational	objekt-relational	relational
ANSI-SQL-Standard	SQL/92	SQL/99 [38]	SQL/92
SQL-Dialektunterstützung	Oracle7, DB2	Nein	Nein
Speicherplatzallozierung	Devspaces	automatisch	automatisch
Transaktionsverarbeitung			
Synchronisationskontrolle	Sperrprotokoll	MVCC + Sperrprotokoll	MyISAM: Tabellensperren InnoDB: MVCC + Sperrprotokoll BDB: Sperrprotokoll
(Kleinste) Sperrgranularität	Zeile	Zeile	MyISAM: Tabelle InnoDB: Zeile BDB: Block
Isolationsgrade	0, 1, 15, 2, 3 (alle SQL/92-Isolationsgrade abgedeckt)	READ COMMITTED, SERIALIZ-ABLE	MyISAM: keine InnoDB: alle SQL/92-Isolationsgrade BDB: nur SERIALIZ-ABLE
Sub-Transaktionen	Ja	Nein	Nein

[38] PostgreSQL orientiert sich bewusst am SQL/99-Standard, unterstützt ihn aber nur teilweise.

Backup und Restore			
Konsistentes Hot-Backup	Ja	Ja	Nein[39]
Inkrementelles Backup	Ja	Nein	Nein
Bulk Data Interface	Ja (Replication Manager)	Ja (COPY-Befehl)	Ja (SELECT INTO- und LOAD DATA-Befehl)
Export des Tabellenschemas	Ja	Ja	Ja
Erweiterter Schemaexport (Views, Stored Procedures, Trigger, Benutzer, etc.)	Nein	Ja	Nein
Unterstützung externer Backup-Tools	ADSM/TSM, Backint for Oracle/SAP DB, NetWorker	Nein	Nein
Replikation			
Hot Standby	Nein	Nein	Master-Slave-Replikation
Load Balancing	Nein	Nein	Ja, nur lesend
Spezielle Verbundtypen in SQL			
OUTER JOIN	Ja	Ja	Ja
NATURAL JOIN	Nein	Ja	Ja
Verschachtelte JOINs	Nein	Ja	Nein[40]

[39] Für InnoDB wird ein kommerzielles Hot-Backup-Tool angeboten.

[40] In MySQL sind JOIN-Ketten der Form *Tabelle1* JOIN *Tabelle2* JOIN *Tabelle3*, … möglich.

Abfrageoptimierung			
Indextypen	B*-Bäume	B-Bäume, R-Bäume, Hashtabellen, GiST	B-Bäume, Hashtabellen, Volltextindex
Funktionsbasierte Indexe	Nein	Ja	Nein
Partielle Indexe	Nein	Ja	Nein
Verbundstrategien	Sort-Join	Nested-Loop-Join, Hash-Join, Merge-Sort-Join	Nested-Loop-Join, One-Sweep-Multi-Join
Performance			MyISAM / InnoDB
Benchmark-Laufzeit (Sek.)	690.56	1434.85	427.19 / 653.61
Durchsatz: Mixed IR, Mixed OLTP (Tup/Sek)	6.02 27.43	5.64 61.70	12.01 / 12.66 11.75 / 14.75
Datensicherheit			
Benutzergruppen Rollen (Kleinste) Granularität	Ja Ja Spalte	Ja Nein Tabelle	Nein Nein Spalte
Zugriffs- und Ressourcenbeschränkungen	Ja	Nein	Ja
Authentifizierungsmethoden	Passwort, XUSER	Passwort, Kerberos, IDENT, PAM, Hostebene	Passwort, Hostebene
Verbindungsverschlüsselung	Nein	SSL	SSL

Datentypen			
SQL/92-Datentypen	Ja, außer Bitketten und Intervalle	Ja	Ja, außer Bitketten und Intervalle
Präzision bei Zeitangaben	Mikrosekunde	Mikrosekunde	Sekunde
Zeitzonen	Nein	Ja	Nein
BLOB	Ja	Ja	Ja
BOOLEAN	Ja	Ja	Nein
Geometrische Datentypen	Nein	Ja	Nein
Aufzählungstyp	Nein	Nein	Ja
Mengentyp	Nein	Nein	Ja
Unicode-Unterstützung	Ja	Ja	Ja
Benutzerdefinierte Datentypen	Nein	Ja	Nein
Funktionen und Operatoren			
Bit-Operationen	Nein	Ja	Ja
Zufallszahlen	Nein	Ja	Ja
Formatierte Zahlenausgabe	Nein	Ja	Ja
Geometrische Funktionen	Nein	Ja	Nein
Zeichensatzkonvertierung	Ja	Ja	Nein
DECODE/ENCODE	Nein	Ja	Nein
Stringtoken extrahieren	Nein	Ja	Ja
Phonetischer Vergleich	Ja	Nein	Ja
Suche mit regulären Ausdrücken	Nein	Ja	Ja
Volltextsuche	Nein	Nein	Ja
Passwortverschlüsselung und Fingerprints	Nein	Nein	Ja
Aggregatfunktion: Varianz	Ja	Ja	Nein

Benutzerdefinierte Funktionen und Operatoren	Nein	Ja	Nein
Sequenzen	Ja	Ja	Nein, aber AUTO_INCREMENT
Constraints			
Fremdschlüssel-Integrität	Ja	Ja	nur bei InnoDB
Löschregel	Ja	Ja	nur bei InnoDB
Änderungsregel	Nein	Ja	Nein
CHECK-Bedingungen	Ja	Ja	Nein
Domain-Definitionen	Ja	Ja	Nein
Verzögertes Überprüfen der referenziellen Integrität	Nein	Ja	Nein
Sub-SELECTs			
Im SELECT-Kopf	Nein	Ja	Nein
In der FROM-Klausel	Ja	Ja	Nein
In Bedingungsausdrücken	Ja	Ja	Nein
Stored Procedures			
PL/SQL-ähnliche Sprache	Ja	Ja	Nein
Benutzerdefinierte Sprachen	Nein	Ja	Nein
Update/Delete mit Cursor	Ja	Nein	Nein
Trigger	Ja	Ja	Nein
Rekursive SQL-Anweisungen	Ja	Nein	Nein
Views	Ja (änderbar)	Ja, nur lesend	Nein
SQL-Transformationssystem	Nein	Ja (Rules)	Nein

Modifikation des Datenbankschemas			
Hinzufügen, Umbenennen und Löschen von Spalten	Ja	Ja	Ja
Modifizieren von Spaltentypen	Ja	Nein	Ja
Hinzufügen von CHECK-Bedingungen	Nein	Ja	Nein
Anlegen und Löschen des Primärschlüssels	Ja	Ja	Ja
Anlegen von Sekundärschlüsseln	Nein	Ja	Ja
Temporäre Tabellen	Ja	Ja	Ja
Namensräume innerhalb einer Datenbank	Ja (durch den Besitzer bestimmt)	Ja (mittels CREATE SCHEMA)	Nein
Objekt-relationaler Ansatz			
Vererbung von Tabellen	Nein	Ja	Nein
Überladen von Funktionen	Nein	Ja	Nein
Array-Datentyp	Nein	Ja	Nein
Benutzerdefinierte Datentypen, Operatoren, Aggregatfunktionen, Sprachen	Nein	Ja	Nein
Regelmäßige Wartungsarbeiten	UPDATE STATISTICS	VACUUM	OPTIMIZE und ANALYZE TABLE

Programmierschnittstellen	ODBC, JDBC, Embedded SQL, Perl, Python, PHP	ODBC, JDBC, API, Embedded SQL, Perl, Python, PHP, Tcl	ODBC, JDBC, API, Embedded SQL, Perl, Python, PHP, Tcl
Interne Grenzwerte (Limits)			
Datenbankgröße *	32 TB	unbeschränkt	unbeschränkt
Tabellengröße *	< 32 TB	16 TB	8 EB (=2^{63})
SQL-Anweisungslänge *	64 KB	16 MB	~ 1 MB
Länge eines Bezeichners	32 Zeichen	64 Zeichen	64 Zeichen
Tabellen je Datenbank *	unbeschränkt	unbeschränkt	unbeschränkt
Spalten je Tabelle	1024	1600	3398
Satzlänge *	8088 Byte	1.6 TB	65534 Byte
Spalten je Index	16	16	16
Indexlänge	1024 Byte	> 8 KB	500 Byte
Datensätze je Tabelle *	unbeschränkt	unbeschränkt	unbeschränkt
BLOB-Größe *	2 GB	< Satzlänge	2 GB
Anzahl an Fremdschlüsseln je Tabelle	16	> 16	-
Spaltenanzahl in GROUP BY bzw. ORDER BY	128	> 64	> 64
Maximale Anzahl von Tabellen in einem Verbund	64	> 64	31
Länge von Stored Procedures / Triggern	255 SQL-Anweisungen	unbegrenzt	-
Anzahl von Triggern je Tabelle	3	unbegrenzt	-

* Die mit Stern gekennzeichneten Werte sind vom Betriebssystem und/oder von den verfügbaren Ressourcen abhängig.

8.2 Vergleich mit Oracle

Unter den kommerziellen Datenbanksystemen zählen Oracle und IBM DB2 zu den am meisten verbreitetsten. Eine aktuelle Marktsudie von IDC [21] sieht Oracle im Jahr 2002 mit einem Marktanteil von 39.4% nach wie vor an der Spitze. IBM DB2 wird mit 33.6% und Microsoft SQL Server an dritter Stelle mit 11.1% bewertet. Grund genug, einen kurzen Überblick über die Features von Oracle9i zu geben, die für den Einsatz von Oracle sprechen:[41]

- Synchronisationskontrolle: MVCC mit Sperrprotokoll auf Zeilenebene
- Isolationsgrade: SERIALIZABLE und READ COMMITTED
- Backup: Online und inkrementell
- SQL*Loader: BDI-Tool von Oracle
- Replikation: Hot Standby, Multimaster-Replikation mit Zwei-Phasen-Commit
- Query-Optimizer: Manuell mit Hints steuerbar
- Query-Plan-Stability: Optimierter QEP kann für Wiederverwendung gesichert werden.
- Hohe Skalierbarkeit mit Parallel-Query, Parallel-DML und Parallel-Index-Build/Scan
- Dynamisches Clustern von zusammengehörenden Tabellen: Dabei werden die Datensätze der Tabellen verzahnt abgespeichert, sodass die interne Speicher-organisation einem Verbund der Tabellen entspricht.
- Indextypen: B-Baum, Hash, Reversed Key und Bitmap
- Funktionsbasierte Indexe
- Auditing: Protokollierung der Zugriffe
- Rechteverwaltung mit Rollen auf Spaltenebene
- Veränderbare Views
- Instead-of-Trigger (für Views): Werden verwendet, um Views modifzierbar zu machen, falls sie diese Eigenschaft nicht ohnehin schon haben.
- Materialized View: Die View wird in einer physikalisch vorhandenen Tabelle abgelegt.
- Benutzerdefinierte Datentypen
- Objektdatentyp mit Methoden
- Nested Tables: Tabellen in Tabellen
- Multimediaunterstützung: interMedia
- Stored Procedures in PL/SQL oder Java (integrierte virtuelle Maschine)
- Embedded SQL für Java, C/C++, COBOL, Pascal, FORTRAN und Ada
- Spezieller Datenbankserver für Webapplikationen: Oracle WebDB

[41] Siehe Oracle9i Database New Features Release 2 (9.2) Part Number A96531-01.

- XML-Unterstützung: Eigener Datentyp für XML mit entsprechenden Funktionen; Abfrageergebnisse können in XML-Dokumente gespeichert werden.
- OLAP: CUBE- und ROLLUP-Operatoren für eine effiziente Datenanalyse
- Automatisches Partitionieren von großen Tabellen
- Data-Warehouse: Oracle stellt Gateways zu Verfügung, um auf heterogene DBMS transparent zugreifen zu können.

Oracle ist – wie man anhand der umfangreichen Featureliste sehen kann – den OSDB um einiges voraus. Gerade bei sehr großen Datenbanken und dort, wo hohe Verfügbarkeit verlangt wird, wird Oracle auf Grund seiner Replikations- und Verteilungsmöglichkeiten gewiss den Vorzug erhalten. Trotzdem hat Oracle auch Nachteile gegenüber den OSDB. Zum Beispiel gibt es nur sehr wenig eingebaute Datentypen. Bitketten, einen reinen Zeittyp und Intervalle gibt es nicht. Weiters können weder die Daten noch das Schema in SQL-Form exportiert werden.

8.3 Ausblick

Wie wir gesehen haben, gibt es sowohl viele Gemeinsamkeiten als auch gravierende Unterschiede zwischen den untersuchten OSDB. So wie bei jedem Softwarevergleich handelt es sich auch hierbei um eine Momentaufnahme der derzeit aktuellen Versionen. Auf den To-Do-Listen befinden sich ambitionierte Ziele, auf die wir gespannt warten dürfen.
SAP DB kündigt an, eine Hot-Standby-Lösung zu integrieren. In der neuen Version 7.4 wurde unter anderem das Log- und Backup-Management verbessert und das System Devspace (Abbildung von den logischen Seitenadressen in die physischen) wegrationalisiert. Außerdem kann jetzt die Datenbankgröße nach dem Erstellen verkleinert werden. Die Liste der dringendsten Weiterentwicklungen von PostgreSQL beinhaltet Replikationsunterstützung, bessere Backup-Möglichkeiten und eine Portierung nach Windows. In MySQL dürfen wir ab der Version 4.1 Sub-SELECTs und referenzielle Integrität für alle Tabellentypen erwarten. Langfristig soll MySQL um Views, Constraints, Stored Procedures und Trigger erweitert werden.

Bei der Betrachtung der Open-Source-Datenbanksysteme haben wir bewusst darauf verzichtet, eine davon als die Beste zu klassifizieren. Eine solche Wertung ist schon allein deshalb nicht möglich, da sich bestimmte Merkmale nicht in diese Denkweise pressen lassen. Stattdessen sollte mit dieser Arbeit die Grundlage geschaffen worden sein, anhand von objektiven Kriterien eine Entscheidungshilfe für die im Kontext der geplanten Anwendung geeignetste Open-Source-Datenbank zu geben. Abschließend lässt sich feststellen, dass die Untersuchung der hier vorgestellten Datenbanksysteme auch in nächster Zukunft nicht abgeschlossen sein wird. Die weitere Entwicklung bleibt abzuwarten.

Anhang A Programmierschnittstellen

Der Anhang A soll einen Überblick über die gängigsten Programmierschnittstellen zu den OSDB geben. Für jede Programmiersprache wird ein kleines Beispiel angegeben, das den Verbindungsaufbau und das Auslesen der Tabelle Buch illustriert. An den zentralen Punkten ist der Quellcode mit Kommentaren versehen.

A.1 ODBC

Das folgende Beispiel setzt eine bereits konfigurierte ODBC-Datenquelle mit dem Namen *biblio* voraus. Unter UNIX wird dies mit der Datei *.odbc.ini* im Homeverzeichnis des aktuellen Benutzers oder mit einer globalen Konfigurationsdatei *odbc.ini* in /var/spool/sql/ini bewerkstelligt. Für die Verwaltung der ODBC-Treiber kann optional ein Treibermanager verwendet werden, oder es wird die entsprechende Treiberbibliothek direkt in die Applikation eingebunden. Für letzteren Fall sieht eine solche Datei für SAP DB wie folgt aus:

Beispiel 8.1 Konfigurationsdatei odbc.ini

```
[biblio]
ServerDB=BIBLIO
ServerNode=localhost
```

Der Programmcode bezieht sich auf alle ODBC-kompatiblen Datenbanksysteme.

Beispiel 8.2 ODBC-Beispielprogramm

```
#include <stdio.h>
#include <sql.h>
#include <sqlext.h>
#include <sqltypes.h>

#define LEN 100 // Zeichenkettenlänge

/* Routine für Fehlerbehandlung */
void BailOut(SQLSMALLINT handletype, SQLHANDLE handle)
{
```

```
unsigned char sqlstate[6];
unsigned char msg[256];

if(handle!=SQL_NULL_HANDLE) {
  SQLGetDiagRec(handletype,handle,1,sqlstate,NULL,
     msg,sizeof(msg),NULL);
  fprintf(stderr,"\nSQL Error %s: %s\n",sqlstate,msg);
  }
else printf("\nError!\n");
exit(1);
}

int main() // Hauptprogramm
{
HENV henv;              // environment handle
HDBC hdbc;              // connection handle
HSTMT hstmt;            // statement handle

/* Deklaration der Datenvariablen */
SQLINTEGER BuchNr;
SQLCHAR Autor[LEN], Titel[LEN], Verlag[LEN];
SQLSMALLINT Jahr;

/* Deklaration der Indikatorvariablen */
SDWORD indBuchNr, indAutor, indTitel, indVerlag, indJahr;

/* Verbindung zur Datenbank herstellen */
if(SQLAllocHandle(SQL_HANDLE_ENV,SQL_NULL_HANDLE,&henv)
     !=SQL_SUCCESS) BailOut(0,SQL_NULL_HANDLE);
```

```
if(SQLAllocHandle(SQL_HANDLE_DBC,henv,&hdbc)!=SQL_SUCCESS)
  BailOut(SQL_HANDLE_ENV,henv);

if(SQLConnect(hdbc,"biblio",SQL_NTS,"test",SQL_NTS,"test",
 SQL_NTS)!=SQL_SUCCESS) BailOut(SQL_HANDLE_DBC,hdbc);

/* SQL-Anweisung ausführen */
if(SQLAllocHandle(SQL_HANDLE_STMT,hdbc,&hstmt)!=SQL_SUCCESS)
  BailOut(SQL_HANDLE_DBC,hdbc);
if(SQLExecDirect(hstmt,
"SELECT BuchNr,Autor,Titel,Verlag,Erscheinungsjahr FROM Buch",
SQL_NTS)!=SQL_SUCCESS) BailOut(SQL_HANDLE_STMT,hstmt);

/* Ergebnisspalten mit Variablen binden */
if(SQLBindCol(hstmt,1,SQL_C_ULONG,&BuchNr,0,&indBuchNr)
     !=SQL_SUCCESS) BailOut(SQL_HANDLE_STMT,hstmt);
if(SQLBindCol(hstmt,2,SQL_C_CHAR,Autor,LEN,&indAutor)
     !=SQL_SUCCESS) BailOut(SQL_HANDLE_STMT,hstmt);
if(SQLBindCol(hstmt,3,SQL_C_CHAR,Titel,LEN,&indTitel)
     !=SQL_SUCCESS) BailOut(SQL_HANDLE_STMT,hstmt);
if(SQLBindCol(hstmt,4,SQL_C_CHAR,Verlag,LEN,&indVerlag)
     !=SQL_SUCCESS) BailOut(SQL_HANDLE_STMT,hstmt);
if(SQLBindCol(hstmt,5,SQL_C_USHORT,&Jahr,0,&indJahr)
     !=SQL_SUCCESS) BailOut(SQL_HANDLE_STMT,hstmt);

/* Ergebnis zeilenweise ausgeben */
while(SQLFetch(hstmt)==SQL_SUCCESS)
  printf("%d, %s, %s, %s, %d\n",
     BuchNr,Autor,Titel,Verlag,Jahr);
```

```
/* Verbindung trennen */
if(SQLDisconnect(hdbc)!=SQL_SUCCESS)
     BailOut(SQL_HANDLE_DBC,hdbc);
SQLFreeHandle(SQL_HANDLE_DBC,hdbc);
SQLFreeHandle(SQL_HANDLE_ENV,henv);
exit(0);
}
```

A.2 JDBC

Zuerst wird der JDBC-Treiber in das JDK-Verzeichnis jre/lib/ext kopiert. Danach muss der Klassenname des Treibers und die DBMS-abhängige Verbindungs-URL angepasst werden. Der Rest ist für alle OSDB ident.

Beispiel 8.3 JDBC-Programmbeispiel

```
import java.sql.*;

public class Biblio {
    public static void main (String [] args)
    {
        String url="jdbc:sapdb://192.168.125.1/BIBLIO";
        String user="test";
        String password="test";
        Connection connection;

        try {
        /* JDBC-Treiber laden und Verbindung aufbauen */
        Class.forName ("com.sap.dbtech.jdbc.DriverSapDB");
        connection =
          DriverManager.getConnection(url, user, password);
```

```
        /* SQL-Anweisung ausführen */
        Statement stmt = connection.createStatement ();
        ResultSet resultSet =
          stmt.executeQuery ("SELECT * FROM Buch");

        /* Ergebnis ausgeben */
        while(!resultSet.isLast()) {
          resultSet.next();
          System.out.println(
            resultSet.getString("BuchNr")+", "+
            resultSet.getString("Autor")+", "+
            resultSet.getString("Titel")+", "+
            resultSet.getString("Verlag")+", "+
            resultSet.getString("Erscheinungsjahr"));
        }

        /* Verbindung trennen */
        connection.close();

        } catch(Exception e) {
          System.err.println(e);
          }
    }
}
```

A.3 API

Da SAP DB keine offizielle Dokumentation seiner C-API zur Verfügung stellt, beschränken wir uns auf PostgreSQL und MySQL.

Beispiel 8.4 PostgreSQL-Beispielprogramm

```
#include <stdio.h>
#include <libpq-fe.h>

void BailOut(PGconn *conn)
{
fprintf(stderr, "\nSQL Error: %s\n", PQerrorMessage(conn));
PQfinish(conn);
exit(1);
}

int main()
{
PGconn *conn; // Verbindungsvariable
PGresult *res; // Ergebnisvariable

int i,j,nFields;

/* Verbindung herstellen */
conn=PQconnectdb("host=localhost dbname=biblio user=test");
if(PQstatus(conn) == CONNECTION_BAD) {
  fprintf(stderr,"\nConnection failed!\n");
  exit(1);
}
```

```
/* Transaktion beginnen, Cursor deklarieren und öffnen */
res=PQexec(conn, "BEGIN");
if(!res || PQresultStatus(res) != PGRES_COMMAND_OK)
     BailOut(conn);
PQclear(res);

res=PQexec(conn, "DECLARE cur_buch CURSOR FOR
     SELECT BuchNr,Autor,Titel,Verlag,Erscheinungsjahr
     FROM Buch");
if(!res || PQresultStatus(res) != PGRES_COMMAND_OK)
     BailOut(conn);

PQclear(res);
res=PQexec(conn, "FETCH ALL IN cur_buch");
if(!res || PQresultStatus(res) != PGRES_TUPLES_OK)
     BailOut(conn);

/* Ergebnis ausgeben */
nFields = PQnfields(res);
for(i = 0; i < PQntuples(res); i++)
    {
        for (j = 0; j < nFields; j++) {
            printf("%s", PQgetvalue(res, i, j));
            if(j<nFields-1) printf(", ");
        }
        printf("\n");
    }
PQclear(res);
```

```
/* Verbindung beenden */
PQfinish(conn);
exit(0);
}
```

Beispiel 8.5 MySQL-Beispielprogramm

```
#include <stdio.h>
#include <mysql.h>

void BailOut(MYSQL *mysql)
{
fprintf(stderr,"\nSQL Error: %s\n",mysql_error(mysql));
mysql_close(mysql);
exit(1);
}

int main()
{
MYSQL mysql; // MySQL-Datenstruktur
MYSQL_RES *result; // Platzhalter für die Ergebnismenge
MYSQL_ROW row; // Platzhalter für eine Zeile

/* Verbindung herstellen */
mysql_init(&mysql);
if (!mysql_real_connect(&mysql,"localhost","test",NULL,
     "biblio",0,NULL,0)) BailOut(&mysql);
```

```
/* SQL-Abfrage ausführen */
if(!mysql_query(&mysql,
     "SELECT BuchNr,Autor,Titel,Verlag,Erscheinungsjahr
     FROM Buch")) BailOut(&mysql);
result=mysql_use_result(&mysql);
if(result==NULL) BailOut(&mysql);

/* Ergebnis ausgeben */
while ((row = mysql_fetch_row(result)))
{
  printf("%s, %s, %s, %s, %s\n",
    (row[0] ? row[0] : "NULL"), (row[1] ? row[1] : "NULL"),
    (row[2] ? row[2] : "NULL"), (row[3] ? row[3] : "NULL"),
    (row[4] ? row[4] : "NULL"));
}

/* Verbindung trennen */
mysql_close(&mysql);
}
```

A.4 Embedded SQL

Bei Embedded SQL handelt es sich auch um einen Standard. Das hier abgedruckte Beispiel gilt für SAP DB, lässt sich aber leicht für andere Datenbanksysteme modifizieren.

Beispiel 8.6 Embedded SQL-Beispielprogramm

```
#include <stdio.h>

void BailOut(sqlcatype *sqlca) {
fprintf(stderr,"\nSQL Error %d: %s\n",sqlca->sqlcode,
     sqlca->sqlerrmc);
```

```
EXEC SQL COMMIT WORK RELEASE;
exit(1);
}

int main()
{
/* Deklaration der gebundenen Variablen */
EXEC SQL BEGIN DECLARE SECTION;
  long BuchNr;
  char Autor[50], Titel[100], Verlag[100];
  int Jahr;
  int indAutor,indTitel,indVerlag,indJahr;
EXEC SQL END DECLARE SECTION;

/* Verbindung herstellen */
EXEC SQL SET SERVERDB 'BIBLIO' ON 'localhost';
EXEC SQL CONNECT 'TEST' IDENTIFIED BY 'TEST';
if(sqlca.sqlcode != 0) BailOut(&sqlca);

/* Cursor deklarieren und öffnen */
EXEC SQL DECLARE cur_buch CURSOR FOR
SELECT BuchNr,Autor,Titel,Verlag,Erscheinungsjahr FROM Buch;

EXEC SQL OPEN cur_buch;
if(sqlca.sqlcode != 0) BailOut(&sqlca);

/* Ergebnis ausgeben */
while(1) {
  EXEC SQL FETCH cur_buch INTO
```

```
     :BuchNr, :Autor :indAutor, :Titel :indTitel,
     :Verlag :indVerlag, :Jahr :indJahr;
  if(sqlca.sqlcode==100) break; // keine weiteren Zeilen
  if(sqlca.sqlcode != 0) BailOut(&sqlca);
  printf("%d, %s, %s, %s, %d\n",BuchNr,
     (indAutor ? "NULL" : Autor ),(indTitel ? "NULL" : Titel ),
     (indVerlag ? "NULL" : Verlag ),(indJahr ?  -1 : Jahr));
}

/* Verbindung trennen */
EXEC SQL CLOSE cur_buch;
EXEC SQL COMMIT WORK RELEASE;
exit(0);
}
```

A.5 PHP

Stellvertretend für die Websprache PHP betrachten wir die beliebte Kombination von PHP und MySQL. Für MySQL und PostgreSQL bietet PHP eigene Funktionen an, die den API-Funktionen ähnlich sind. Auf SAP DB wird über die in PHP integrierte ODBC-Schnittstelle zugegriffen.

Beispiel 8.7 PHP-Programmbeispiel

```
<?
// Verbindung herstellen
$hostName = "localhost";
$userName = "test";
$password = "";
$dbName = "biblio";

mysql_connect($hostName, $userName, $password)
or die("Verbindung fehlgeschlagen.");
```

```
mysql_select_db($dbName)
or die( "Verbindung zur Datenbank $dbName fehlgeschlagen.");

// SQL-Anweisung ausführen
$query = "SELECT * FROM Buch";
$result = mysql_query($query);

// Ergebnis in HTML-Tabelle ausgeben
print "<html><body><h1>Bibliothek</h1><table>";
$number = mysql_numrows($result);
for ($i=0; $i<$number; $i++) {
     $BuchNr = mysql_result($result,$i,"BuchNr");
     $Autor = mysql_result($result,$i,"Autor");
     $Titel = mysql_result($result,$i,"Titel");
     $Verlag = mysql_result($result,$i,"Verlag");
     $Jahr = mysql_result($result,$i,"Erscheinungsjahr");
     print "<tr><td>$BuchNr</td><td>$Autor</td><td>$Titel</td>
          <td>$Verlag</td><td>$Jahr</td></tr>";
}
print "</table></body></html>";

// Verbindung trennen
mysql_close();
?>
```

A.6 Perl

Perl verwendet für den Datenbankzugriff eine einheitliche Schnittstelle, die in der Klasse DBI (Database Interface) definiert ist. Im Hintergrund wird ein DBMS-spezifischer DBD (Database Driver) benötigt. Das folgende Beispiel verwendet DBD:Pg für PostgreSQL.

Beispiel 8.8 Perl-Programmbeispiel

```
#!/usr/bin/perl

use DBI;
use strict;

# Verbindung herstellen
my $conn = DBI->connect("dbi:Pg:dbname=biblio","test","");

my ($BuchNr,$Autor,$Titel,$Verlag,$Jahr);
my (@row);

# SQL-Anweisung ausführen
my $query = $conn->prepare(
    "SELECT BuchNr,Autor,Titel,Verlag,Erscheinungsjahr " .
    "FROM Buch"
    );
$query->execute();

# Ergebnis anzeigen
while (@row = $query->fetchrow_array()) {
    ($BuchNr,$Autor,$Titel,$Verlag,$Jahr) = @row;
    print sprintf("%s, %s, %s, %s, %s\n",
          $BuchNr, $Autor, $Titel, $Verlag, $Jahr);
}
```

```
# Verbindung trennen
undef($query);
$conn->disconnect();
$conn = undef;
```

A.7 Python

Alle drei OSDB bieten Python DB API 2.0-konforme Treiber an. Als Beispiel zeigen wir den Zugriff auf PostgreSQL.

Beispiel 8.9 Python-Beispielprogramm

```
#!/usr/local/bin/python
import sys
from pyPgSQL import PgSQL

try:
  # Verbindung herstellen
  conn = PgSQL.connect(host="localhost",database="biblio",
     user="test")
  cursor = conn.cursor()

  # SQL-Anweisung ausführen
  stmt = "SELECT BuchNr, Autor, Titel, Verlag,
     Erscheinungsjahr FROM Buch"
  cursor.execute(stmt)
  resultSet = cursor.fetchall()

  # Ergebnis anzeigen
  for BuchNr,Autor,Titel,Verlag,Jahr in resultSet:
     print BuchNr, ", ", Autor, ", ", Titel, ", ", Verlag,
          ", ",Jahr
```

```
  # Verbindung trennen
  cursor.close()
  conn.close()
  sys.exit(0)

except PgSQL.Error, msg:
  print "\nSQL Error: %s\n" % msg
  sys.exit(1)
```

Anhang B Abkürzungen

ACID	Atomicity, Consistency, Isolation, Durability
ANSI	American National Standards Institute
API	Application Programming Interface
AS3AP	ANSI SQL Standard Scalable and Portable (Benchmark)
ASCII	American Standard Code for Information Interchange
BDB	BerkeleyDB (MySQL)
BDI	Bulk Data Interface
BLOB	Binary Large Object
BSD	Berkeley Software Distribution
CLI	Command Line Interface
CSV	Comma Separated Values
DB	Datenbank
DBA	Database Administrator (SAP DB)
DBD	Database Driver (Perl)
DBI	Database Interface (Perl)
DBM	Database Manager Operator (SAP DB)
DBMS	Database Management System
DRBD	Distributed Replicated Block Device
ERP	Enterprise Resource Planning
GiST	Generalized Search Tree
GNU	GNU is Not UNIX
GPL	General Public License
GUI	Graphical User Interface
IND	Inclusion Dependency
ISAM	Indexed Sequential Access Method (MySQL)
ISO	International Organization for Standardization
JDBC	Java Database Connectivity
JDK	Java Development Kit
MD5	Message-Digest 5 (Algorithmus)
MVCC	Multiversion Concurrency Control
NFS	Network File System
ODBC	Open Database Connectivity
OLAP	On-Line Analytic Processing
OLTP	On-Line Transaction Processing
ORDBMS	Object-Relational Database Management System

OSDB	Open-Source-Datenbanken
PAM	Pluggable Authentication Modules (Linux)
PHP	PHP Hypertext Preprocessor
PL	Procedural Language
QEP	Query Execution Plan
RA	Relationenalgebra
RAID	Redundant Array of Inexpensive Disks
RDK	Relationen-Domain-Kalkül
RTK	Relationen-Tupel-Kalkül
SH1	Secure Hash 1 (Algorithmus)
SQL	Structured Query Language
SSL	Secure Sockets Layer (Protokoll)
SYSDBA	Database System Administrator (SAP DB)
Tcl	Tool Command Language
TCP/IP	Transmission Control Protocol/Internet Protocol
URL	Uniform Resource Locator

Anhang C Literaturverzeichnis

[1] SAP DB Documentation 7.3, http://www.sapdb.org, SAP AG, 2002

[2] PostgreSQL 7.3.1 Documentation, http://www.postgresql.org, The PostgreSQL Global Development Group, 2002

[3] MySQL Reference Manual for version 4.0.3-beta, http://www.mysql.com, MySQL AB, 2002

[4] G. Vossen: Datenmodelle, Datenbanksprachen und Datenbankmanagementsysteme, 4. Auflage, Oldenburg Verlag München Wien, 2000

[5] C.J. Date: An Introduction To Database Systems, 6. Auflage, Addison-Wesley Publishing Company, 1995

[6] H. Garcia-Molina, J. D. Ullmann, J. Widom: Database System Implementation, Perentice Hall, 2000

[7] Y. Trudeau: Sapdb standby HA Howto, http://www.gestionbt.ca:8080/Sapdb-standby-HA-Howto.html, 2001

[8] J. Darren: PostgreSQL Database Replication Options (Slides) at O'Reilly Open Source Conference, 2002

[9] P. Reisner: DRDB, http://www.complang.tuwien.ac.at/reisner/drbd/, 2002

[10] B. Kemme, G. Alonso: Don't be lazy, be consistent: Postgres-R, A new way to implement Database Replication, in Proceedings of VLDB Conference, 2000

[11] GNU General Public License, http://www.gnu.org/licenses/gpl.html, Version 2, 1991

[12] C. J. Date, H. Darwen: SQL – Der Standard, SQL/92 mit den Erweiterungen CLI und PSM, Addison Wesley Longman GmbH, 1998

[13] P. Lehman, B. Yao: Efficient Locking for Concurrent Operations on B-Trees, ACM Transactions on Database Systems, Vol. 6, No. 4, 1981

[14] A. Guttman: R-trees: A dynamic index structure for spatial searching, in Proceedings of ACM SIGMOD Conference, 1984

[15] W. Litwin: Linear Hashing: A New Tool for File and Table Addressing, in Proceedings of VLDB Conference, 1980

[16] S. Simkovics: (Diplomarbeit) Enhancement of the ANSI SQL Implementation of PostgreSQL, Institut für Informationssysteme, Technische Universität Wien, 1998

[17] P. Franz: (Diplomarbeit) Vergleich von relationalen Datenbanken und Entwicklungswerkzeugen der vierten Generation, Sozial- und wirtschaftswissenschaftliche Fakultät, Universität Wien, 1992

[18] Z. Fong: (M.S. report) The design and implementation of the POSTGRES query optimizer, Computer Science Division, University of California, 1986

[19] T. Härder, E. Rahm: Datenbanksysteme, Konzepte und Techniken der Implementierung, 2. Auflage, Springer-Verlag Berlin Heidelberg New York, 2001

[20] J. Gray: The Benchmark Handbook for Database and Transaction Systems, 2. Auflage, Morgan Kaufmann, 1993

[21] International Data Coperation (IDC) Research: Oracle Hears Footsteps: IBM and Microsoft Gain on RDBMS Leader in 2002, Doc #29022, 2003