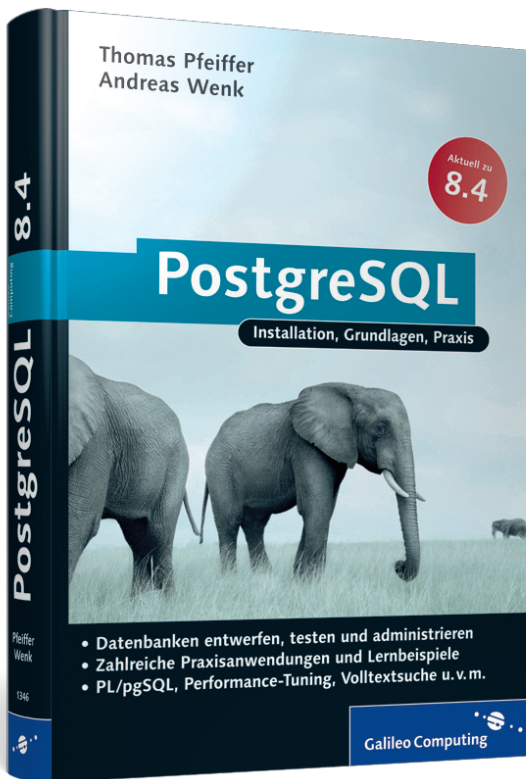


Thomas Pfeiffer, Andreas Wenk

PostgreSQL

Das Praxisbuch



Auf einen Blick

1	Vorwort	13
2	Werkzeuge	17
3	Praxis 1: Die Grundlagen	47
4	Praxis 2: Fortgeschrittene Funktionen	131
5	User Defined Functions	225
6	Praxis 3: Textsuche, Performance, Administration	299
7	Installation	405

Inhalt

Geleitwort von Peter Eisentraut	11
---------------------------------------	----

1 Einleitung	13
---------------------------	-----------

2 Werkzeuge	17
--------------------------	-----------

2.1	Das mitgelieferte Kommandozeilenprogramm psql	17
2.1.1	psql unter Windows	17
2.1.2	Einige wichtige psql internen Befehle näher betrachtet	22
2.1.3	psql mit SQL-Befehlen nutzen	24
2.1.4	SQL-Befehle aus einer externen Datei aufrufen	26
2.2	pgAdmin III – das Standard-PostgreSQL-Frontend	27
2.2.1	Verbindung zu einem Datenbank-Cluster herstellen	30
2.2.2	Eine Datenbank erstellen	33
2.3	Weitere Features von pgAdmin III	39
2.3.1	Der Grant Assistent	40
2.3.2	Werkzeuge	43

3 Praxis 1: Die Grundlagen	47
---	-----------

3.1	Herausforderung und Modell: Unsere kleine Firma	47
3.2	Theorie und Praxis: Was ist SQL?	55
3.2.1	SQL – Structured Query Language	55
3.2.2	Wie fing es an?	56
3.2.3	Der SQL-Sprachkern	57
3.3	Relationale Datenbanken und das Entity-Relationship-Modell	60
3.3.1	Relationale Datenbanken	60
3.3.2	Das Entity-Relationship-Modell (ER-Modell)	63
3.4	Die Umsetzung	65
3.4.1	Erstellen und Löschen einer Datenbank [CREATE DATABASE, DROP DATABASE]	65
3.4.2	Tabellen erstellen [CREATE TABLE, DROP TABLE]	68
3.4.3	Nichts ist von Bestand – Daten aktualisieren [UPDATE]	77

3.4.4	Weg damit – Daten löschen [DELETE]	79
3.4.5	Her mit den Daten! – Einfache Abfragen [SELECT]....	80
3.4.6	Bitte nicht alles – Nur bestimmte Daten abfragen [WHERE]	82
3.4.7	Das Muster macht's [LIKE]	85
3.4.8	Seitenweise [LIMIT und OFFSET]	86
3.4.9	Sortiert wär's besonders schön [ORDER BY]	87
3.5	Exkurs 1: Datenbankdesign und seine Folgen	89
3.5.1	Am Anfang war der Zettel und der Bleistift	89
3.5.2	Datenbankmodellierung	90
3.6	Schlüsselfrage: Keys & Constraints	91
3.7	Exkurs 2: Sinn und Zweck von Templates	99
3.8	Datentypen	100
3.8.1	Ganzzahlentypen	100
3.8.2	Zahlen beliebiger Präzision	101
3.8.3	Fließkommatypen	103
3.8.4	Selbstzählende Datentypen	105
3.8.5	Zeichenkettentypen	107
3.8.6	Typen für Datum und Zeit	108
3.8.7	Geometrische Typen	110
3.8.8	Arrays	113
3.8.9	Weitere Datentypen	118
3.9	Vergleiche und andere nützliche Dinge: Operatoren und Aggregatfunktionen	123
3.9.1	Logische Operatoren	123
3.9.2	Vergleichsoperatoren	124
3.9.3	Mathematische Operatoren	125
3.9.4	Aggregatfunktionen	126
3.10	Gedankenstütze: Kommentare in der Datenbank	128

4 Praxis 2: Fortgeschrittene Funktionen 131

4.1	Veränderung muss sein: Spalten hinzufügen, entfernen, umbenennen [ALTER TABLE]	133
4.2	Regelwerk: foreign keys & Constraints	136
4.3	Abfragen über mehrere Tabellen [JOIN]	143
4.4	Ordnung halten: Daten sortiert und gruppiert ausgeben [GROUP, ORDER, HAVING, DISTINCT]	151
4.5	Transaktionen: Ein paar Worte zum Thema Sicherheit	154
4.6	Kontrollstrukturen per SQL [CASE .. WHEN .. THEN]	161
4.7	Reguläre Ausdrücke: Noch mehr Muster	163

4.7.1	SIMILAR TO	164
4.7.2	Reguläre Ausdrücke	165
4.8	Wenn eine Abfrage nicht reicht – Subselects (Unterabfragen)	166
4.9	Common Table Expressions und Recursive Queries [WITH, WITH RECURSIVE]	168
4.10	Window Functions [OVER (PARTITION BY ...)]	171
4.10.1	Einfache Window Functions	172
4.10.2	Window Function mit Subselect	173
4.10.3	Kombination aus CTE und Window Function	173
4.11	Datenmengen [UNION, EXCEPT, INTERSECT]	175
4.12	Typecasting: Wenn der Typ nicht stimmt	178
4.13	In Serie: Sequenzen [NEXTVAL, CURVAL, SETVAL]	179
4.14	Selects auf Abwegen [CREATE TABLE AS]	181
4.15	Finden und gefunden werden: Indizes	182
4.15.1	Einfache Indizes	183
4.15.2	Mehrspaltige Indizes	183
4.15.3	Unique Constraints	184
4.15.4	Funktionsindizes	184
4.15.5	Partielle Indizes	185
4.16	Views: Sichten auf das System	186
4.16.1	Views	186
4.16.2	Schemata	191
4.17	Mehr Sicherheit: Das Rechte- und Rollensystem [GRANT, REVOKE, OWNER]	194
4.18	Wenn mal was anderes gemacht werden soll – Das Regelsystem [CREATE RULE]	199
4.19	Funktionen für alle Lebenslagen	204
4.19.1	Mathematische Funktionen	204
4.19.2	Datums- und Zeitfunktionen	207
4.19.3	Zeichenkettenfunktionen	210
4.19.4	Aggregatfunktionen	212
4.20	Die Form wahren: Ausgabeformatierung	215
4.21	Jede Menge Daten [COPY]	218

5 User Defined Functions 225

5.1	Stored Procedures versus User Defined Functions	226
5.2	Vorteile durch den Einsatz von User Defined Functions	226
5.3	Mit Bordmitteln – SQL	228
5.3.1	Kurzer Überblick	228

5.3.2	Der Aufbau einer User Defined Function	228
5.3.3	Eine User Defined Function ausführen	232
5.3.4	Eine User Defined Function umbenennen	233
5.3.5	Eine User Defined Function löschen	233
5.3.6	Alle eigenen User Defined Functions ansehen	234
5.3.7	Funktionen ohne Rückgabewert (RETURNS void)	236
5.3.8	Funktionen mit einfachen Datentypen als Rückgabewert (RETURNS integer, text, numeric ...)	238
5.3.9	Funktionen mit zusammengesetzten Datentypen	239
5.3.10	Funktionen, die ein Mengenergebnis zurück liefern (RETURNS SETOF)	241
5.4	Wenn's ein bisschen mehr sein soll: PL/pgSQL	248
5.4.1	Eigenschaften von Funktionen in PL/pgSQL	248
5.4.2	Installation von PL/pgSQL	249
5.4.3	Welche Eingabe- und Rückgabewerte sind möglich?	249
5.4.4	Der Aufbau einer User Defined Function in PL/pgSQL	250
5.4.5	Debug-Ausgaben und Exceptions	253
5.4.6	Rückgabe: RETURN, RETURN NEXT und RETURN QUERY	255
5.4.7	Variablen deklarieren und einen Alias für einen Parameter vergeben	256
5.4.8	Die unterschiedlichen Statements	263
5.4.9	Es geht rund: Kontrollstrukturen	267
5.4.10	Cursor	277
5.5	Auslösende Momente [TRIGGER]	287
5.6	Darwin in der Datenbank [INHERITS]	293

6 Praxis 3: Textsuche, Performance, Administration 299

6.1	Suchmaschine im Eigenbau: Volltextsuche	299
6.1.1	Prinzip der Volltextsuche	300
6.1.2	Die Funktionen to_tsvector() und to_tsquery() und die Datentypen tsvector und tsquery	302
6.1.3	Der GIN- und der GiST-Index	305
6.1.4	Aufbau einer Suche	309
6.1.5	Weitere Funktionen für die Volltextsuche	314
6.1.6	Operatoren für die Volltextsuche	319
6.1.7	Eine Suche starten	321
6.1.8	Dictionaries	327

6.1.9	Konfiguration	334
6.2	Performance-Tuning	337
6.2.1	Einführende Überlegungen	338
6.2.2	Der Weg einer Anfrage bis zum Ergebnis	341
6.2.3	EXPLAIN ANALYZE – einen Query Plan lesen	344
6.3	Administration	355
6.3.1	Benutzerverwaltung [CREATE ROLE]	355
6.3.2	Authentifizierung – die Datei pg_hba.conf	370
6.3.3	Exkurs: Multiversion Concurrency Control (MVCC).....	374
6.3.4	Wartung der Datenbank [VACUUM]	375
6.3.5	Sicher ist sicher: Backup und Recovery	379
6.3.6	Schlussbemerkungen	391
6.4	Tablespaces und Tabellenpartitionierung	392
6.4.1	Tablespaces	392
6.4.2	Tabellenpartitionierung	397
7	Installation	405
7.1	Installation auf Linux-Systemen	405
7.1.1	Die Quellen selbst übersetzen (kompilieren)	405
7.1.2	Installation mit dem Paketmanager	411
7.2	Installation unter Windows	413
7.2.1	Der Downloadbereich der Webseite	414
7.2.2	pgInstaller – One-Click-Installer	414
7.3	Die wichtigsten Konfigurationsdateien	420
7.3.1	postgresql.conf	421
7.3.2	Die Einstellungen in der Datei postgresql.conf	421
7.3.3	pgtune für Linux-Systeme	427
7.4	Schlussbemerkungen	429
7.5	Startschuss	429
	Index	431

Wie kommt man zur PostgreSQL? Wer hilft? Und: Was erwartet Sie in diesem Buch? Ein kurzer Abschnitt, der Ihnen das Weiterlesen schmackhaft machen soll.

1 Einleitung

BEGIN WORK;

Ziele

Fangen wir damit an, was dieses Buch nicht ist. Dieses Buch ist keine Referenz! Der Grund dafür ist relativ einfach: Referenzen zu schreiben ist langweilig, und Referenzen zu lesen ist noch viel langweiliger.

Unser Ziel ist es, dass Sie als Leser die Vorzüge der PostgreSQL kennen lernen – mit oder ohne SQL-Erfahrung. Wenn Sie bereits Erfahrungen mit Datenbanken gesammelt haben auch gut. Wir hoffen, Sie entdecken Dinge, die Sie bislang noch nicht kannten. Um Sie auf Ihrer Forschungsreise zu unterstützen, geben wir Ihnen mit diesem Buch einen Praxisleitfaden an die Hand.

Wir haben für Sie eine Sammlung anschaulicher Beispiele zusammengestellt. Auch wenn diese vielleicht nicht jedermanns Geschmack treffen, der eine will mehr, der andere weniger, so sollten sich die Beispiele doch recht einfach an Ihre eigenen Bedürfnisse adaptieren lassen. Damit kommen dann sowohl Einsteiger als auch fortgeschrittene Benutzer auf ihre Kosten.

Wege zur PostgreSQL

Vielleicht haben Sie ja einen ähnlichen *Leidensweg* wie die Autoren hinter sich. Grundsätzlich im Thema Webentwicklung angesiedelt, haben wir lange Zeit MySQL als Datenbank eingesetzt. Dies ergab sich (wie auch für viele andere Webentwickler) aufgrund der weiten Verbreitung von MySQL bei den Hosting-Anbietern fast automatisch. Allerdings, wie das nun mal so ist: Man wächst mit der Erfahrung. Irgendwann sind wir ein-

fach an den Punkt gekommen, an dem wir »*ein bisschen mehr Datenbank*« brauchten.

Nun sind wir grundsätzlich der Meinung, dass es sinnvoller ist, Geld in das Wissen der Mitarbeiter anstatt in teure Lizenzen zu investieren, das heißt, bei unseren Projekten bevorzugen wir, wenn möglich, Open-Source-Technologien und -Software. Damit fiel unsere Wahl recht schnell auf die PostgreSQL-Datenbank – eine Entscheidung, die wir bislang nicht bereut haben.

PostgreSQL ist cool!

Tatsächlich erhalten Sie mit der PostgreSQL ein mächtiges Datenbanksystem. Damit verbunden ist an so mancher Stelle natürlich eine höhere Komplexität, insbesondere bei der Konfiguration und bei der Erweiterung des Datenbanksystems. Auf der anderen Seite erhalten Sie aber auch eine höhere Flexibilität, oder salopp gesagt: PostgreSQL hat eine Menge Schrauben, an denen Sie drehen können.

Kein Grund zur Panik – in diesem Buch zeigen wir Ihnen, wie Sie sicher durch die vermeintlichen Untiefen des Datenbanksystems manövrieren. Und um gleich mal mit einem verbreiteten Vorurteil aufzuräumen: Ja, es dauert ein bisschen länger mit der PostgreSQL warm zu werden, aber es ist nicht zu kompliziert. Ganz im Gegenteil: Sie werden sehen, dass ein PostgreSQL-Cluster genauso schnell installiert ist wie andere Datenbanksysteme. Auch die Konfiguration stellt keine allzu großen Hürden dar.

Aber nicht vergessen: Dies ist keine Referenz. Das ist auch nicht notwendig, denn wer Referenzmaterial benötigt (und das werden Sie), wird leicht im Internet fündig. PostgreSQL hat eine extrem gute Dokumentation zu bieten (<http://www.postgresql.org>). Wir ermutigen Sie bereits jetzt, sich die Seite ausgiebig anzusehen und die entsprechende Dokumentation, passend zu der von Ihnen verwendeten Version (bevorzugt natürlich die 8.4) zu bookmarken.

Lizenz

Die PostgreSQL-Datenbank ist vollständig Open Source und steht unter der BSD-Lizenz (*Berkley Software Distribution*, <http://de.wikipedia.org/wiki/BSD-Lizenz>). Das bedeutet, dass Sie die Software kopieren, verändern oder vertreiben dürfen – so wie Sie es für nötig halten, solange Sie

einen Copyright-Hinweis in Ihrer Distribution belassen (<http://www.postgresql.org/about/licence>). Diese Lizenz gibt Ihnen die größtmögliche Freiheit und Flexibilität für Ihre Projekte, zwingt Sie aber auch, die Auflage und ihre Regeln zu respektieren.

Community

Hinter der PostgreSQL steht wie bei so vielen Open-Source-Projekten eine Community – eine große, hilfsbereite und viel gelobte Community – eine wirklich coole Community. Wir möchten Sie ermutigen, sich an dieser zu beteiligen. Der einfachste Einstiegspunkt ist das Mitlesen der unterschiedlichen Mailinglisten (<http://www.postgresql.org/community/lists/>). Stellen Sie Fragen. Beantworten Sie Fragen. Beteiligen Sie sich an der Weiterentwicklung, indem Sie Vorschläge besprechen, oder übermitteln Sie selbst einen *Patch* (englisch *Flicken*, also eine Nachbesserung des Programms). Es stehen Ihnen alle Wege offen. Nur so wächst die Software und damit die Verbreitung von PostgreSQL.

Außerdem möchten wir auf zwei deutsche Webseiten hinweisen: das deutsche PostgreSQL-Forum (<http://www.pg-forum.de/>) und die Informations- und Linksammlung unter <http://www.postgresql.de/>. Ein Vorbeisurfen lohnt sich alle mal obwohl die Websites nicht soooo aktiv sind (bis auf das Forum). Die absoluten PostgreSQL-Geeks treffen Sie allerdings im IRC Chat auf *freenode.net*: *#postgresql* (*irc://irc.freenode.net/postgresql*) und *#postgresql-de* (*irc://irc.freenode.net/postgresql-de*).

ROLLBACK?

Sie halten die erste Auflage unseres Buches in den Händen. Sie können uns glauben – wir haben alles auf Herz und Nieren getestet. Aber nobody is perfect. Wenn sich also Fehler eingeschlichen haben, bitten wir Sie uns diese mitzuteilen, damit wir diese in der zweiten Auflage (die ja bestimmt kommt ...) korrigieren können. Senden Sie uns doch einfach eine E-Mail an info@pg-praxisbuch.de. Oder noch einfacher haben Sie es, wenn Sie die Website zum Buch unter <http://www.pg-praxisbuch.de> besuchen. Dort können Sie mit uns in Kontakt treten, einen Kommentar hinterlassen oder uns mit Fragen löchern ... Die Beispiele aus dem Buch finden Sie übrigens dort oder auf der Website des Verlages: <http://www.galileocomputing.de/2008>.

Danke

Ohne ein paar nette Menschen, auf deren Unterstützung wir zählen können und konnten, wäre es schwer gewesen, das Buch zu schreiben. Da wären zum Beispiel Klara, Kiana, Ole, Meike und Verena, die auf uns verzichten mussten und uns mit gewährter Zeit unterstützt haben. Annette und Wolfgang Wenk danken wir für das Schreibcamp im März 2009 und viele hilfreiche Tipps zum Thema *Schreiben*. Außerdem danken wir Andreas Putzo und Arne Böttger, die quergelesen, viele sehr hilfreiche Kommentare gegeben und Fehler isoliert haben (klingt nicht so schlimm wie »entdeckt haben«). Stefan Reimers, seines Zeichens selbst Autor beim Galileo-Computing-Verlag, ist der Fachgutachter dieses Buchs und hat zu allen Kapiteln Kommentare, Fragen und Verbesserungsvorschläge geliefert – danke.

Tja – und dann wollen wir uns sehr herzlich bei dem Mann bedanken, der dieses Buch überhaupt erst möglich gemacht hat, weil er uns im letzten Jahr gefragt hat, ob wir Lust haben ein Buch über die PostgreSQL-Datenbank zu schreiben: unser Lektor Jan Watermann.

Nicht vergessen dürfen wir die Core-Entwickler und Contributor der PostgreSQL-Datenbank. Zur ersten Kategorie gehört unter anderem Peter Eisentraut. An ihn geht ein dickes Dankeschön für das sehr gelungene Vorwort. Ohne diese Menschen und deren unermüdlichem Einsatz gäbe es die PostgreSQL nicht, und wir müssten auf eine Menge Spaß und die *most advanced open source database* verzichten.

So, bei uns in Hamburg sagt man, wenn's losgehen soll: »Nu ma ran an' Speck!« Viel Erfolg und Spaß wünschen Ihnen

Hamburg,
Thomas Pfeiffer und Andreas Wenk

COMMIT;

Wir werden hier den ersten produktiven Kontakt mit der Datenbank aufnehmen, und obwohl das Kapitel »Praxis 1« heißt, bleibt es uns nicht erspart, doch noch einen kurzen Ausflug zur theoretischen Seite zu machen.

3 Praxis 1: Die Grundlagen

In diesem Kapitel wollen wir uns zunächst mit den folgenden grundlegenden Fragen befassen: Wie kommen die Daten in die Datenbank, und wie kommen sie wieder heraus? Bevor wir nun unsere Datenbank erstellen und jede Menge SQL-Statements produzieren, werden wir uns zunächst ein praktisches Beispiel ausdenken, an dem wir das Gelernte festzurren.

3.1 Herausforderung und Modell: Unsere kleine Firma

Gehen wir von folgender Problematik aus: Wir haben ein Unternehmen mit einer gewissen Anzahl von Mitarbeitern. Dieses Unternehmen bietet Produkte und/oder Dienstleistungen zum Verkauf an. Die Herausforderung, der wir uns nun gegenüber sehen, ist die Erstellung einer Datenbank, in der Informationen über Mitarbeiter, Produkte, Kunden, Bestellungen und daraus resultierende Rechnungen hinterlegt werden können.

Dieses kleine Modell wird uns dann als Grundlage für die vorgestellten Funktionen der Datenbank genügen, erhebt aber natürlich keinen Anspruch auf Vollständigkeit. Wir werden der Übersicht halber viele Tabellendefinitionen etwas kleiner halten, also zum Beispiel bei den Kunden auf zusätzliche Felder für die Lieferanschrift oder Bankverbindung verzichten, da es sich bei solchen Felder eher um Wiederholungen eines bereits vorgestellten Typs handelt. Es ist Ihnen natürlich freigestellt, den Tabellen weitere Felder nach Belieben (sofort oder später) hinzuzufügen.

Außerdem wird der Kern unserer Datenbank aus einer vergleichsweise übersichtlichen Zahl von Tabellen bestehen. Erweiterungen sind auch

hier denkbar, etwa für die Bereiche »Lagerverwaltung«, »Mahnwesen« oder vielleicht sogar »Customer-Relationship-Management«. Diese Themen würden sich allerdings besser in einem Buch über Applikationsentwicklung machen, das wir hier jedoch nicht schreiben.

Nun zurück zur Aufgabe: Die ersten Daten die wir in unserer Datenbank hinterlegen möchten, sind die unserer Mitarbeiter. Wir sehen hier Felder für die gängigen Informationen wie »Vorname«, »Nachname«, »Telefon« und so weiter vor. Eine Struktur für die Tabelle `mitarbeiter` könnte also etwa wie folgt aussehen:

Information	Typ des Werts
ID	Mitarbeiter-Nr. / eindeutiger Schlüssel
Anrede	nur die Werte »herr« oder »frau«
Vorname	Zeichenkette, max. 100 Zeichen
Nachname	Zeichenkette, max. 100 Zeichen
E-Mail	Zeichenkette, max. 150 Zeichen
Position	Zeichenkette, max. 150 Zeichen
Telefon	Zeichenkette, max. 100 Zeichen
Mobiltelefon	Zeichenkette, max. 100 Zeichen
Straße	Zeichenkette, max. 100 Zeichen
PLZ	Zeichenkette, max. 10 Zeichen
Ort	Zeichenkette, max. 100 Zeichen
Bemerkungen	beliebig lange Zeichenkette für einen Freitext
Gehalt	Zahl mit zwei Nachkommastellen
Geburtsdatum	Datum

Tabelle 3.1 Struktur für die Tabelle »mitarbeiter«

Was wir wollen ist folgendes: Die Tabelle `mitarbeiter` soll wie übrigens die meisten unserer Tabellen eine eigene *eindeutige ID* bekommen. Grundsätzlich könnte man hier eine vielleicht bereits vorhandene Personalnummer verwenden, wir möchten jedoch erreichen, dass das Datenbanksystem für jeden neu eingefügten Datensatz eine eigene ID vergibt und werden deswegen später den Datentyp `serial` verwenden, der genau dies gewährleistet. Wir werden im Folgenden `serial` zwar als Datentyp

bezeichnen, doch es sei schon im Vorfeld angemerkt, dass es sich hierbei nur um einen Alias für die Verwendung einer *Sequenz* handelt. Wer es jetzt schon genauer wissen möchte, springt einfach schnell mal zu Abschnitt 3.8.4, »Selbstzählende Datentypen«.

Unterschied natürlicher und technischer Primärschlüssel

Beachten Sie, dass eine bereits vorhandene Personalnummer einen *natürlichen Primärschlüssel* darstellt. Das ist also ein Schlüssel, der eindeutig ist – bei einer Personalnummer in einem Unternehmen sollte das sichergestellt sein, da die Personalabteilung (hoffentlich) fortlaufende Nummern nutzt.

Im Gegensatz dazu ist der als *serial* generierte Primärschlüssel ein *technischer Schlüssel*. Diesen Schlüssel lassen wir von der Datenbank erstellen (er ist selbstverständlich eindeutig und fortlaufend), und er ist völlig unabhängig von irgendwelchen Eigenschaften des Unternehmens im *real life*.

Im Feld für die Anrede möchten wir nur die Werte »herr« oder »frau« abspeichern. Das mag etwas merkwürdig anmuten – man könnte hier auch nur »0« oder »1« für eine männlich/weiblich-Unterscheidung oder aber einen Freitext für größtmögliche Flexibilität zulassen. Wir möchten anhand dieses Feldes jedoch den ersten Kontakt mit dem Regelwerk (Constraints) der Datenbank aufnehmen.

Die Felder *Vorname* bis *Ort* sind dann wieder weitgehend selbsterklärend. Hier sollen nur beliebige Zeichenketten mit einer Maximallänge gespeichert werden. Übrigens legen wir hier auch Postleitzahlen und Telefonnummern als Zeichenketten ab. Darüber kann man zwar geteilter Meinung sein, aber wir entscheiden uns hier dafür, um zum Beispiel Probleme mit der Speicherung führender Nullen zu vermeiden (etwa bei der Postleitzahl 01099).

Die letzten drei Felder haben dann noch einmal andere Datentypen: Im Feld für die Bemerkungen möchten wir einen beliebig langen Text erfassen, brauchen also eine maximale Obergrenze hier nicht.

Das Gehalt möchten wir als Dezimalzahl mit zwei Nachkommastellen speichern. Als Letztes wollen wir dann noch das Geburtsdatum des Mitarbeiters hinterlegen können und werden auch hier den dafür bestimmten Datentyp benutzen.

Wir gehen mal davon aus, dass es sich bei dem Unternehmen nicht um eine »Zweimannklitsche« handelt, sondern dass wir eine größere Anzahl von Mitarbeitern verwalten. Üblicherweise sind diese Mitarbeiter gewis-

sen Abteilungen zugeordnet. Wir erfassen hier zunächst ganz einfach die Abteilungen des Unternehmens:

Information	Typ des Werts
ID	eindeutiger Schlüssel der Abteilung
Abteilungs-Nr.	Ganzzahl (Integer)
Abteilung	Zeichenkette, max. 50 Zeichen

Tabelle 3.2 Struktur für die Tabelle »abteilungen«

Damit haben wir die Struktur für einen zweiten »Informations-Container«. Noch wissen die beiden Objekte `abteilungen` und `mitarbeiter` nichts voneinander. Das werden wir aber in Kürze ändern, wenn wir in der Tabelle `mitarbeiter` jedem Datensatz (also Mitarbeiter) sagen, zu welcher Abteilung er gehört. Praktisch bedeutet das, dass wir der Struktur für die Tabelle `mitarbeiter` noch ein weiteres Feld hinzufügen, in dem wir einen Verweis auf einen Eintrag in der Tabelle `abteilungen` speichern können.

Damit sieht die gewünschte Struktur zwischen den beiden zu erstellenden Tabellen wie folgt aus:

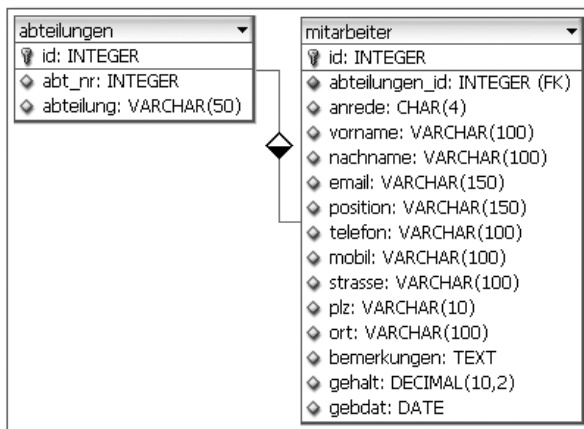


Abbildung 3.1 Mitarbeiter und Abteilungen im fertigen Modell

Wir haben beide Tabellen also mit jeweils einem sogenannten *Primärschlüssel* (*primary key*) versehen.

Eine Grundanforderung an ein relationales Datenbanksystem ist, dass jeder Datensatz einer Tabelle (Relation) über einen Schlüssel, der sich nicht ändert, eindeutig identifizierbar sein muss.

Ein Primärschlüssel ist eine *eindeutige Identifikation* einer Zeile in einer Tabelle. Ein Wert einer als Primärschlüssel definierten Spalte darf in der Tabelle nur einmal vorkommen, (sonst hat sich's was mit Eindeutigkeit) und es darf nur einen Primärschlüssel pro Tabelle geben.

Technisch gesehen ist der Primärschlüssel eine Verknüpfung von `UNIQUE Constraint` und `NOT NULL Constraint`, aber wir wollen der Geschichte nicht vorgreifen, später erfahren Sie mehr dazu.

In Abbildung 3.1 sehen Sie ebenfalls die bereits angekündigte Beziehung zwischen den beiden Tabellen. Es handelt sich hierbei um eine *1:n-Beziehung*, das heißt, eine Abteilung kann beliebig viele Mitarbeiter enthalten.

Dafür haben wir der Tabelle `mitarbeiter` die Spalte `abteilungen_id` gegönnt. Diese ist vom gleichen Typ wie die Spalte `id` der Tabelle `abteilungen` (Integer). Die Angabe `FK` nach dem Spaltentyp besagt, dass es sich hierbei um einen sogenannten *foreign key* (*Fremdschlüssel*) handelt. Das bedeutet, dass Werte dieser Spalte aus der Schlüsselspalte einer anderen Tabelle stammen.

Jetzt zu den Dingen, die wir verkaufen möchten. Das können entweder Produkte und/oder Dienstleistungen sein. Allen gemeinsam dürfte sein, dass sie eine Bezeichnung und einen Preis haben. Auf das Wesentlichste beschränkt könnte die Struktur unserer Produkttabelle etwa so aussehen:

Information	Typ des Werts
ID	eindeutiger Schlüssel des Produkts
Artikel-Nr.	Zeichenkette, max. 100 Zeichen (zum Beispiel EAN-Code)
Bezeichnung	Zeichenkette, max. 200 Zeichen
Beschreibung	beliebig lange Zeichenkette für Produktbeschreibung
Preis	Zahl mit zwei Nachkommastellen
Steuersatz	Zahl mit drei Nachkommastellen (zum Beispiel 0.190)

Tabelle 3.3 Struktur für die Tabelle »produkte«

In dieser Tabelle haben wir uns dazu entschlossen, die Artikelnummer als Zeichenkette abzulegen und nicht wie in der Tabelle `abteilungen` die Abteilungsnummer als Integer-Wert. Das ist letztlich Geschmacksache, erlaubt in der Produkttabelle aber auch die Speicherung alphanumerischer Zeichenketten. Der Preis des jeweiligen Produkts soll als Nettobetrag mit zwei Nachkommastellen gespeichert werden, der Steuersatz hingegen als Zahl mit drei Nachkommastellen (man denke an Frankreich mit zur Zeit 19,6 % Mehrwertsteuer).

Gönnen wir den Kunden unseres Unternehmens auch noch eine eigene Tabelle. Hier werden zunächst nur einfache Informationen wie Name und Anschrift gespeichert. Wir weichen an dieser Stelle nur von unserer Definition des Felds `anrede` ab, so wie wir es für die Tabelle `mitarbeiter` beschrieben haben, da wir bei unseren Kunden mit der Anrede gegebenenfalls auch so etwas wie »Herr Prof. Dr.« abspeichern möchten.

Information	Typ des Werts
ID	eindeutiger Schlüssel, Kundennummer
Anrede	Zeichenkette, max. 50 Zeichen
Vorname	Zeichenkette, max. 100 Zeichen
Nachname	Zeichenkette, max. 100 Zeichen
Straße	Zeichenkette, max. 100 Zeichen
PLZ	Zeichenkette, max. 10 Zeichen
Ort	Zeichenkette, max. 100 Zeichen
Telefon	Zeichenkette, max. 100 Zeichen
E-Mail	Zeichenkette, max. 200 Zeichen

Tabelle 3.4 Struktur für die Tabelle »kunden«

Kunden können Bestellungen aufgeben, deshalb werden wir diese in einer dafür bestimmten Tabelle speichern. Für den ersten Entwurf genügt es, wenn wir uns zu einer Bestellung die Kundennummer, das Bestell-, Liefer- und Rechnungsdatum merken. Daraus ergibt sich folgende Möglichkeit einer Struktur:

Information	Typ des Werts
ID	eindeutiger Schlüssel, Bestellnummer
Kunden-Nr.	Verweis auf eine ID in der Tabelle <code>kunden</code>
Bestelldatum	Zeitstempel
Lieferdatum	Zeitstempel
Rechnungsdatum	Zeitstempel

Tabelle 3.5 Struktur für die Tabelle »bestellungen«

Im Feld für die Kundennummer müssen wir später sicherstellen, dass hier nur Werte gespeichert werden können, die auch als ID in der Tabelle `kunden` vorhanden sind. Außerdem müssen wir noch festlegen, was mit einem Eintrag in der Tabelle `bestellungen` passieren soll, wenn ein dazu korrespondierender Eintrag in der Tabelle `kunden` gelöscht wird. Hier werden Sie eine weitere Begegnung mit dem Regelwerk der Datenbank haben.

Für das Bestell-, Liefer- und Rechnungsdatum, das wir uns in dieser Tabelle merken möchten, wollen wir einen sogenannten Zeitstempel (engl. timestamp) verwenden, das heißt, wir können mit einer vollständigen Zeitangabe wie etwa »2009-12-24 14:00:00« arbeiten.

Nun reicht es natürlich nicht aus, sich einfach nur zu merken, dass ein bestimmter Kunde eine Bestellung aufgegeben hat, sondern wir wollen natürlich auch wissen, was dieser Kunde im Einzelnen bestellt hat.

Da sich eine Bestellung aus mehreren Positionen zusammensetzen kann, wollen wir hier eine weitere und zunächst letzte Tabelle anlegen. Sie soll die Bestellungen mit den Produkten verknüpfen. In dieser Tabelle benötigen wir eigentlich erst mal nur drei Informationen: die ID der Bestellung, die ID des bestellten Produkts und die Menge, die von diesem Produkt bestellt wurde.

Information	Typ des Werts
Bestell-Nr.	Verweis auf eine ID in der Tabelle <code>bestellungen</code>
Produkt-Nr.	Verweis auf eine ID in der Tabelle <code>produkte</code>
Menge	Ganzzahl, Vorgabewert 1

Tabelle 3.6 Struktur für die Tabelle »bestellungen_produkte«

Es fällt auf, dass diese Tabelle keine ID-Spalte wie die vorherigen Tabellen hat. Wir gehen mal davon aus, dass kein Produkt in einer Bestellung doppelt auftaucht, sondern bei Mehrfachbestellungen eines Produkts einfach die Menge erhöht wird. Deshalb ist die Nummer der Bestellung in Kombination mit der ID des Produkts eine Möglichkeit, den Datensatz eindeutig zu identifizieren. Wir werden hier später einen verketteten Primärschlüssel einsetzen.

Damit haben wir für unsere Beispieldatenbank zunächst sechs Tabellen zur Verfügung, um zumindest das Basissystem zu implementieren. Später werden wir dieses Modell dann noch an die erweiterten Anforderungen anpassen.

Im grafischen Modell sieht die zusätzliche Struktur der vier neuen Tabellen dann so aus:

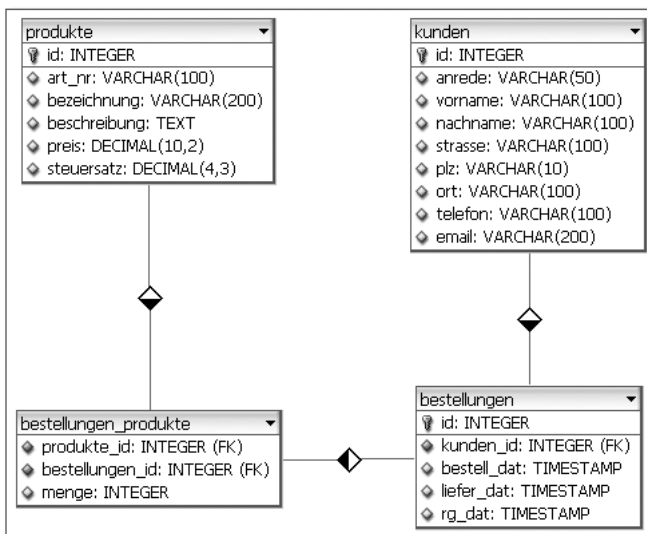


Abbildung 3.2 . Produkte, Kunden und Bestellungen im fertigen Modell

Auch hier zeigen sich wieder die Beziehungen zwischen den einzelnen Tabellen. So kann ein Kunde beliebig viele Bestellungen tätigen (1:n-Beziehung).

Das kennen wir schon aus der Beziehung zwischen den Tabellen *mitarbeiter* und *abteilungen*. Aber wir finden hier auch etwas Neues: eine sogenannte *n:m-Beziehung*: Das bedeutet, eine Bestellung kann nicht nur ein, sondern beliebig viele Produkte enthalten, ebenso wie ein Produkt

nicht nur in einer, sondern in ebenfalls beliebig vielen Bestellungen vorkommen kann.

Den »Kitt« zwischen den Tabellen `bestellungen` und `produkte` liefert uns die Tabelle `bestellungen_produkte`.

Den soeben angekündigten verketteten Primärschlüssel können Sie jetzt auch gut erkennen. Nehmen wir einmal an, wir hätten (unter anderem) Produkte mit den IDs 10, 20 und 30. Des Weiteren haben wir Bestellungen mit den IDs 1, 2, und 3 vorliegen.

In Bestellung 1 wurden die Produkte mit den IDs 10 und 20 jeweils einmal bestellt. In Bestellung 2 wurden die Produkte mit den IDs 20 und 30 jeweils zweimal bestellt. Schließlich wurden in Bestellung 3 einmal das Produkt mit der ID 10, zweimal das Produkt mit der ID 20 und dreimal das Produkt mit der ID 30 bestellt. Dann sieht der Inhalt der Tabelle `bestellungen_produkte` wie folgt aus:

produkte_id	bestellungen_id	menge
10	1	1
20	1	1
20	2	2
30	2	2
10	3	1
20	3	2
30	3	3

Tabelle 3.7 Daten der Tabelle »bestellungen_produkte«

Wie Sie sehen ist die Kombination von `produkte_id` und `bestellungen_id` eindeutig und genügt uns somit in der Kombination als primärer Schlüssel.

3.2 Theorie und Praxis: Was ist SQL?

3.2.1 SQL – Structured Query Language

Wir haben ja schon im vorigen Kapitel mit der PostgreSQL gesprochen. Wie war das gleich? `INSERT`, `SELECT`, `CREATE DATABASE` ...

Sobald wir als Mensch mit der Maschine »reden« wollen, muss eine Sprache her, die uns das ermöglicht. So gibt es in der Softwareentwicklung Sprachen der ersten (Assembler), zweiten (Cobol) und mittlerweile dritten Generation (Java, PHP, Ruby), die nichts anderes tun, als es dem Anwender so einfach wie möglich zu machen, mit der Maschine zu kommunizieren. *SQL (Structured Query Language)* ermöglicht das Gleiche, wobei SQL an sich keine Programmiersprache im klassischen Sinn ist – wie der Name ja auch sagt.

Wir müssen also die Syntax erlernen und letztlich verstanden haben. Danach sollte es uns möglich sein, Operationen auszuführen, wie etwa eine Datenbank zu erstellen oder Tabellen zu erstellen, Daten in die Datenbank einzufügen und Daten wieder aus der Datenbank auszulesen.

Wer das Inhaltsverzeichnis bereits überflogen hat, wird festgestellt haben, dass wir in Abschnitt 3.4, »Die Umsetzung«, eine Einführung in die SQL-Syntax geben. Im weiteren Verlauf des Buchs werden wir Ihnen dann reichlich Gelegenheit geben, Ihre Kenntnisse weiter zu vertiefen.

3.2.2 Wie fing es an?

Im Zeitalter von Wikis und der »Wikisierung« all unseren Wissens ist es kein Problem, reichlich Informationen über SQL zu erhalten. Für uns soll hier ein kleiner Rückblick auf die Entstehungsgeschichte ausreichend sein.

Eine Firma namens IBM hat SQL für ihre Forschungen an relationalen Datenbanksystemen bereits Mitte der 70er-Jahre entwickelt. Daran maßgeblich beteiligt war der Computerwissenschaftler Edgar F. Codd. Er gilt als der Erfinder des relationalen Datenbanksystems. Bei seiner Forschung hat er für die Abfrage der Datenbank den Vorläufer von SQL namens *SEQUEL (Structured English Query Language)* genutzt. Entwickelt wurde die Sprache von zwei IBM-Mitarbeitern namens Donald D. Chamberlin und Raymond F. Boyce.

Fast gleichzeitig hat auch Oracle (damals Relational Software) ein *DBMS (Datenbank-Managementsystem)* mit SQL als Sprache entwickelt und als Produkt vertrieben.

1986 war es dann so weit, und es wurde der erste Standard durch das *ANSI (American National Standards Institute)* verabschiedet: SQL-86 (auch SQL 1) war geboren. 1989 wurde der Standard dann aktualisiert und hat – wie nicht schwer zu erraten – den Namen SQL-89 verpasst bekommen.

In den Jahren 1992 und 1999 gab es wiederum Aktualisierungen des Standards. PostgreSQL unterstützt diese beiden Standards: SQL-92 (auch SQL 2) und SQL-99 (auch SQL 3).

Mittlerweile gibt es auch die Standards SQL-2003 und SQL-2006. Seit 1987 hat die ISO (*Internationale Organisation für Normung*) die gleichnamigen Standards aufgenommen.

Im Folgenden lesen Sie einen kleinen Überblick, welche Datenbank welche SQL-Standards unterstützt:

Datenbank	Version	SQL Standard
PostgreSQL	8.4	fast vollständig SQL-2003
MySQL	5.x	teilweise SQL-99, teilweise SQL-2003, teilweise proprietär
Oracle	10g	SQL-2003, teilweise proprietär
Ingres	2006 Release 3	SQL-92 / SQL-99
Firebird	2.1.1	SQL-92 / SQL-99, teilweise SQL-2003
IBM DB2	9.1	weitestgehend SQL-2003, teilweise proprietär
Sybase	11	teilweise SQL-2003
Informix	11.50	teilweise SQL-2003

Tabelle 3.8 Welche DB unterstützt welche Standards?

3.2.3 Der SQL-Sprachkern

SQL ist in drei Bereiche unterteilt, die jeweils unterschiedliche Datenbankoperationen abdecken. Diese Bereiche werden im Folgenden kurz vorgestellt.

Data Definition Language – DDL

Zu diesem Bereich gehören die Befehle `CREATE`, `ALTER` und `DROP`. Daraus ergibt sich fast von selbst die Funktion der DDL: Objekte erstellen, verändern oder löschen – das alles geschieht mit der DDL. Dazu gehört auch

das Erstellen der Datenbank selbst oder das Anlegen von Tabellen oder Views. Genauso gehören Operationen zum Verändern oder Löschen dieser Elemente dazu.

Data Manipulation Language – DML

Wie der Name schon sagt, geht es in diesem Bereich um das Verändern von Datenbankinhalten. Dabei kommen die Befehle `SELECT`, `INSERT`, `UPDATE` und `DELETE` zur Anwendung. Im Gegensatz zum endgültigen Befehl `DROP` sollten diese Befehle immer in eine Transaktion gekapselt werden. Was eine Transaktion ist und wie diese angewandt wird, erfahren Sie in Abschnitt 4.5, »Transaktionen: Ein paar Worte zum Thema Sicherheit«. Vorab wollen wir ein kurzes Beispiel hierzu geben, da das Thema sehr wichtig ist – wenn Ihnen Ihre Daten lieb sind.

Eine Transaktion wird eingesetzt, um die Konsistenz der Datenbank aufrechtzu erhalten. Nehmen wir die bereits vorgestellten Tabellen `kunden` und `bestellungen`. Jeder Kunde kann viele Bestellungen tätigen. Dies ist eine, wie oben schon erwähnte, *1:n-Verbindung*. Konsistent sind unsere Tabellen nur, wenn es zu jeder Bestellung in der Tabelle `bestellungen` mindestens einen Kunden in der Tabelle `kunden` gibt. Um die Tabellen mit Daten zu füllen, verwenden Sie die zwei folgenden `INSERT`-Statements:

```
INSERT INTO bestellungen (id,kunden_id) VALUES (1,24);
INSERT INTO kunden (id,vorname,nachname) VALUES
(24,'Andy','Wenk');
```

Was passiert nun, wenn das erste Statement aus irgendeinem Grund schief geht? Richtig, wir haben Leichen in der Tabelle `bestellungen`, weil es zur Bestellung mit der ID 1 keinen Kunden in der Tabelle `kunden` gibt.

Aus diesen (in größeren Applikationen fatalen) Gründen werden solche `INSERT`-Statements in eine Transaktion gekapselt. Nur wenn beide `INSERT`-Befehle erfolgreich durchgeführt werden konnten, werden Daten in die Datenbank geschrieben. In SQL sieht das dann so aus:

```
BEGIN TRANSACTION;
INSERT INTO bestellungen (id,kunden_id) VALUES (1,24);
INSERT INTO kunden (id,vorname,nachname) VALUES
(24,'Andy','Wenk');
COMMIT TRANSACTION;
```

Schreibweisen

Die Befehle `BEGIN TRANSACTION` und `COMMIT TRANSACTION` sind die ausführliche Schreibweise. `BEGIN` kann entweder von der Anweisung `WORK` oder `TRANSACTION` gefolgt werden. Sie können aber auch einfach nur `BEGIN` schreiben, da PostgreSQL selbstständig eine Transaktion durchführt, sobald mehr als ein SQL-Statement ausgeführt werden soll.

Wir sagen der PostgreSQL mit dem einleitenden `BEGIN` dass wir einen Transaktionsblock starten wollen. Mit `COMMIT` weisen wir die Datenbank an, den Transaktionsblock zu beenden. Alles, was zwischen `BEGIN` und `COMMIT` steht, wird also in einer Transaktion durchgeführt. Die Transaktion ist nur erfolgreich, wenn beide Statements erfolgreich ausgeführt werden konnten. Schlägt ein Statement fehl, führt die PostgreSQL einen `ROLLBACK` durch. Dadurch werden alle Änderungen an der Datenbank rückgängig gemacht, und unsere Datenbank bleibt in einem konsistenten Zustand. Sehr gut!

Data Control Language (DCL)

Jede PostgreSQL-Datenbank hat Benutzer (zumindest einen). Mit den Befehlen `GRANT` und `DENY` ist es uns möglich, Regeln festzulegen, welcher Benutzer Zugriff auf bestimmte Ressourcen der Datenbank (zum Beispiel auf eine Tabelle oder ein View) hat, oder welche Operationen der Benutzer ausführen darf. Damit ist uns etwa möglich, dem Benutzer *pfeiffer* nur die Befehle `SELECT`, `INSERT` und `UPDATE` in der Datenbank *db_von_andy* zu erlauben. Der Benutzer *wenk* hat dagegen auch das Recht, die Operation `DELETE` auszuführen und kann somit einfach alle Daten, die der Benutzer *pfeiffer* eingefügt hat, wieder löschen. (oh oh – das gibt Ärger ...).

Diese drei Bereiche umfassen alle möglichen Operationen, die wir in unserer Datenbank vornehmen können. Das hier Beschriebene ist natürlich nur der theoretische Ansatz. Den tieferen Einblick erhalten Sie im Abschnitt 3.4, »Die Umsetzung«. Bevor es damit losgeht, stellen wir Ihnen im nächsten Abschnitt das Modell unserer Beispielanwendung vor. In dem Zusammenhang erfahren Sie auch, was Entity-Relationship bedeutet und was eine relationale beziehungsweise objekt-relationale Datenbank ist.

Index

= 257
1/128 373
1:n-Beziehung 51, 90
1:n-Verbindung 58

A

Abfragen
 über mehrere Tabellen 148
Abhängigkeiten 408
absteigende Reihenfolge 88
ACID 159
ADD CONSTRAINT 96
ADD MAPPING FOR 337
Administration 355
age 207
Aggregatausdruck 153
Aggregatfunktionen 123, 126, 153, 212
Alias 85, 256, 258
ALIAS FOR 258
all 371
ALTER 133
 ADD COLUMN 134
 ADD CONSTRAINT 135
 DROP COLUMN 134
 DROP CONSTRAINT 135
 RENAME COLUMN 134
 SET DEFAULT 134
ALTER MAPPING FOR 337
ALTER ROLE 360
ALTER TABLE 96
ALTER TEXT SEARCH CONFIGURATION
 337
ANALYZE 182, 344, 347, 378
Anker 166
Aptitude 408, 411
Arrays 113
ASC 87, 152
aufsteigende Reihenfolge 87
Ausgabeformatierung 215
Authentifizierung 370
autoconf 406
automake 406
autovacuum 182
AVG 127, 212, 214

B

Backup & Recovery 379
Backup-Skript 388
Bacula 381
Bedingungen 82
BEGIN 156, 250
Belastungstests 340
Benutzer 195
Benutzergruppen 195
Benutzerkonten 194
Beziehungen 61
bigint 100
bigserial 105
binäre Daten 120
Bingo 277
Bitmap Index Scan 345
Blob 375
BOOLEAN 122
Box 111
Boyce, Raymond F. 56
B-Tree 307
BYTEA 120

C

CASE 267
CASE .. WHEN .. THEN 161
case sensitive 71, 86
CASE THEN 268
CAST 179
cert 372
Chamberlin, Donald D. 56
CHAR 71
char 107
character 107
character varying 107
CHECK-Constraint 70, 400
Checkpoint 425
Chen-Diagramm 63
CIDR-ADDRESS 373
Circle 111
CLI 17
Client-Server-Kommunikation 341
coalesce() 312
Codd, Edgar F. 56
COMMENT 128

COMMIT 156
 composite 256
 configure 406
 CONST 257
 Constraint
 CHECK 95
 NOT NULL 93, 97
 UNIQUE 93, 98
 constraint_exclusion 403
 Constraints 91, 136
 Unique Constraints 184
 COPY 218
 COPY FROM 219
 COPY TO 221
 CouchDB 374
 COUNT 126, 212
 CREATE DATABASE 65
 CREATE INDEX 94, 306, 307, 352
 CREATE LANGUAGE 249
 CREATE OR REPLACE FUNCTION 250
 CREATE ROLE 194, 355, 358
 CREATE ROLE (Gruppenrollen) 361
 CREATE RULE 403
 CREATE TABLE 68
 CREATE TABLE AS 181
 CREATE TEXT SEARCH DICTIONARY 328
 createdb 67, 385
 createlang 249
 creatuser 363
 Cron-Daemon 387
 Crow's-Foot-Diagramm 64
 current_date 207
 current_time 207
 current_timestamp 207
 currval 155
 Cursor 277
 ABSOLUTE 279
 BACKWARD 280
 CLOSE 281, 282
 Cursor als Referenz 282
 Cursor deklarieren 278
 Cursor-Referenz 282, 285
 FETCH 279, 280, 284
 FIRST 279
 FORWARD 280
 gebundene/ungebundene 278
 LAST 279
 MOVE 279, 280
 NEXT 279, 284
 OPEN 278, 280, 283
 OPEN CURSOR 281
 OPEN cursor FOR 279
 PRIOR 279

 RELATIVE 279
 RETURNS refcursor 283
 CURVAL 179

D

Data Control Language 59
 Data Definition Language 57
 Data Manipulation Language 58
 date 108
 date_trunc 208
 Datenbankdesign 338
 Datenbankmodell 89
 Datenbank-Werkzeuge 17
 Datenimport 218
 Datenintegrität 159
 Datentypen 100
 Datum 108
 Debug-Ausgaben 253
 DECLARE 250, 252, 256
 default dictionary 320
 default_text_search_config 336
 Defaultwerte 72
 DELETE 79
 DELIMITER 220
 DESC 88, 152
 Dictionarys 327
 DIFFERENCE 62
 DISTINCT 151
 DISTINCT ON 154
 DIVIDE 62
 DO 202
 DO INSTEAD 200
 double precision 103
 DROP DATABASE 65
 DROP INDEX 94
 DROP MAPPING 337
 DROP TABLE 68
 dropdb 385
 dropuser 363, 364
 Duplikate entfernen 154

E

Eigentümer 196
 Einstellungsparameter 421
 ELSEIF 268
 END 250
 END LOOP 271
 Entität 60, 63, 90
 Entity-Relationship-Modell 60, 63
 escapen 166
 EXCEPT 175

Exception 253
 EXECUTE 263, 265
 Executor 343
 EXPLAIN 344, 347
 explicit JOIN clauses 353
 extract 208

F

Fallunterscheidung 161
 Fließkommatypen 103
 FOR IN 270, 272, 274
 foreign key 51, 136
 FOUND 263
 FragmentDelimiter 325
 Free Space Map 376, 378
 FREEZE 377
 Fremdschlüssel 51
 Fremdschlüssel-Beziehung 137
 frozenXID 378
 FSM 377, 378
 Funktionen 204
 Datum und Zeit 207
 Mathematische 204
 Zeichenketten 210
 Funktionsindex 185

G

Ganzzahlentypen 100
 Genetic Query Optimizer 343
 Geometrische Typen 110
 GEQO 343
 german_stem 335
 GIN 305
 Generalized Inverted Index 306
 GiST 305, 307
 Generalized Search Tree Index 307
 GNU make 407
 GRANT 194, 365
 USAGE 198
 GRANT ALL ON TABLESPACE 395
 GROUP 151
 Groups 368
 Gruppenrolle 361
 gss 371
 GUI 17

H

Hardware 339
 Hash Join 346

HAVING 151
 Header-Dateien 408
 Header-Dateien (C) 408
 HighlightAll 325
 host 371
 hostnssl 371
 hostssl 371

I

IBM 56
 ident 371
 IF 267
 IF THEN ELSE 267
 INDEX
 CREATE INDEX 183
 DROP INDEX 183
 Funktionsindizes 184
 Mehrspaltige 183
 Partieller Index 185
 Index 93, 182
 Verwendung 182
 Index Scan 345
 Indizes 62
 INET 119
 INHERITS 397, 399
 Inklusionsverknüpfung 146
 InnoDB 374
 INSERT 74
 Installation 405
 Installation (Paketmanager) 411
 Installation Linux 405
 Installation Windows 413
 integer 100
 INTERSECT 62, 175
 INTERVAL 207
 interval 108
 IP-Adressen 373
 Ipv4 371
 Ipv6 371
 ISO-8601 110
 ispell 328
 Ispell Dictionary 332

J

Java 284
 JOIN 62, 143
 LEFT JOIN 145
 RIGHT JOIN 146
 join_collapse_limit 354
 Join-Typen 345

K

kartesisches Produkt 84
 Keys 91
 Kommentare 128
 Kommentarzeichen 422
 kompilieren 405
 Konfiguration 410, 412
 Konfigurationsdateien 420
 Kontrollstrukturen 161, 267
 Koordinaten 113
 krb5 371

L

label 252
 large object 375
 ldap 372
 length() 315
 length(tsvector) 315
 Lexem 301
 libxml 409
 LIKE 85
 LIMIT 86
 Linux-Distribution 412
 listen_addresses 423
 local 371
 Locale 417
 LOCK 374
 LOID 375
 LOOP 270, 271, 274
 Löschfortpflanzung 61, 139
 lower 210
 LPAD 212
 lpad 271
 Lseg 111

M

maintenance_work_mem 308
 make 407
 make install 407
 Makefile 407
 Manpage 364
 Maskierung 266
 MAX 127, 214
 max_connections 423
 MaxFragments 325
 MaxWords 325
 md5 371
 mehrdimensionale Datenstruktur 113

Merge Join 346
 MIN 127, 214

 MinWords 325
 Momentaufnahme der Datenbank 156
 Multiversion Concurrency Control 374
 Mustersuche 164
 Mustervorlage 215
 MVCC 156, 345, 374
 MySQL 374

N

n:m-Beziehung 54
 Nachkommastellen 79
 Namensraum 191
 Nested Loop Join 346
 Netzwerkadressen 119
 NEXTVAL 179
 nextval 155
 NOTICE 254, 255
 numeric 101

O

OFFSET 86
 OID 375
 ON DELETE 137
 CASCADE 138
 RESTRICT 141
 One-Click-Installer 414
 Operatoren 123
 Logische 123
 Mathematische 125
 Vergleichs- 124
 Operatorklassen 306
 Optimizer 342
 Optimizer Hints 342
 Oracle 56
 ORDER 151
 ORDER BY 87
 OUT 259, 260, 262
 OWNER 194

P

pam 372
 Parallels 390
 Parameter 258
 partielle Indizes 309
 password 371

- PATH 413
- Path 111
- PERFORM 263
- Performance-Tuning 337
- pg_Programme 413
- pg_ctl 411
- pg_default 393
- pg_dump 381, 385
- pg_dumpall 382, 390
- pg_global 393
- pg_hba.conf 370
- pg_ident.conf 372
- pg_restore 386, 387
- pg_standby 392
- PG_VERSION 395
- pgAdmin III
 - Abfrage ausführen, Ergebnis speichern* 29
 - Abfrage zerlegen* 29
 - Ausgabefeld* 28
 - Backup und Wiederherstellen* 44
 - Berichte* 44
 - CREATE DATABASE* 32
 - Datenbank erstellen* 33, 34
 - Datenbankbenutzer* 33
 - Datentyp* 37
 - Grant Assistent* 39, 40
 - Gruppenrolle erstellen* 40
 - Historie* 29
 - Login-Rollen* 41
 - Meldungen* 29
 - Mitglied in* 41
 - Mitgliedschaft in Gruppenrolle* 41
 - Objektbrowser* 31, 34, 35, 37
 - Passwort speichern* 30
 - pgAdmin III* 27
 - Privilegien* 37, 40
 - Rechte für mehrere Tabellen gleichzeitig ändern* 41
 - Schema* 32
 - Server* 30
 - Server Status* 44
 - Skripts* 44
 - Spalten* 37
 - SQL-Editor* 28
 - SQL-Feld* 33, 37
 - Tabellen erstellen* 35
 - Verbindung zu Cluster* 30
 - Werkzeuge* 43
 - Werte in Tabellen einfügen* 37
- pgBouncer 392
- pgInstaller 414
- pgPool II 392
- pgtune 427
- PHP 284
- plainto_tsquery() 304
- Plan lesen 347
- Planer 342
- Planknoten-Typen
 - HashAggregate* 347
 - Limit* 347
 - Materialize* 347
 - Sort* 347
 - Unique* 347
- plpgsql 253
- Point 111
- Polygon 111
- port 423
- POSIX-Standard 166
- postgresql.conf 339, 355, 421
 - autovacuum* 426
 - checkpoint_completion_target* 425
 - checkpoint_segments* 425
 - client_min_messages* 426
 - effective_cache_size* 426
 - listen_addresses* 423
 - log_destination* 426
 - log_directory* 426
 - log_filename* 426
 - log_rotation_age* 426
 - logging_collector* 426
 - maintenance_work_mem* 424
 - max_connections* 423
 - max_fsm_pages* 424
 - max_fsm_relations* 424
 - port* 423
 - shared_buffers* 424
 - wal_buffers* 425
 - work_mem* 424
- Primärschlüssel 49
 - natürlicher* 49
 - technischer* 49
 - verketteten* 54
- PRODUCT 62
- PROJECT 62
- psql 17, 22, 23
 - [+]* 22
 - [S+]* 22
 - \i* 26
 - Befehlshistorie* 24
 - dt* 22
 - psql unter auf Unix- Systemen* 19
 - psql unter Windows* 17
 - Tab-ABC completion* 24

template1 25
verfügbare Hilfetemen 20
wichtige psql internen Befehle 22

Q

Quantoren 166
 query 80
 query @@ vector 322
 Query Plan 342
 Query Plan lesen 349
 Query Rewriter 342
 quote_ident() 266
 quote_literal() 266
 quote_nullable() 266

R

RAISE 253
 LOG 254
 NOTICE 254, 255
 RAISE 255
 WARNING 254
 RDBMS 60
 readline 408
 real 103
 RECORD 263
 refcursor 278, 285
 REFERENCES 137
 referenzielle Integrität 61
 regconfig 325
 Regeln für gute Performance 341
 Regelsystem 189
 Regulärer Ausdruck 163
 Reihenfolge
 absteigend 88
 aufsteigend 87
 reject 371
 Relation 51
 Relationale Datenbanken 60
 Relationale Operatoren 61
 relationales Datenbanksystem 51
 Relationen 60
 Release-Zyklus 412
 RENAME 361
 REPLACE 251
 RESET 360
 RETURN 253, 267
 RETURN NEXT 255, 275
 RETURN QUERY 255, 262
 RETURNING 264
 RETURNS 250
 RETURNS RECORD 261

RETURNS SETOF 273
 RETURNS TABLE 259, 261, 262
 REVOKE 194, 362, 369
 Rewrite-System 201
 ROLLBACK 156
 Rollen Attribute 357
 Rollensystem 194
 RPAD 212
 R-Tree 307
 RULE 199

S

samerole 371
 sameuser 371
 scalar 256
 Scan-Typen 344
 Schemata 191
 CREATE SCHEMA 191
 public 191
 SCP 381
 searched CASE 269
 searchpath 193
 Selbstzählende Datentypen 105
 selbstzählenden Datentypen 179
 SELECT 62, 80
 SELECT INTO 262, 263, 277
 SELF JOIN 147
 SEQUEL 56
 Sequential Scan 344
 Sequenz 69
 Sequenzen 179
 SERIAL 69
 serial 27, 105
 Server-Client-Kommunikation 343
 SETOF 273
 SETVAL 179
 setweight 310
 Shared Memory 423
 Shell-Skript 387
 ShortWord 325
 SHOW 351
 Sichten 186
 SIMLAR TO 164
 simple 328, 335
 simple CASE 269
 Simple Dictionary 330
 skalare Variable 263
 slony 392
 smallint 100
 snowball 328
 Snowball Dictionary 333
 Spaltenbezogene Rechte 196

- Spaltennamen 80
- SPLIT_PART 211
- SQL Standards 57
- sspi 371
- Stack Builder 417
- StartSel 325
- Statement
 - gruppierendes* 153
- Statement-Optimierung 338
- Statement-Typen 263
- Stop Words 330
- StopSel 325
- Storage Engine 374
- String-Verkettung 163
- Stringverkettung 178
- strip() 315
- strip(tsvector) 315
- Structured Query Language 55
- Subnetzmaske 373
- Subselects 166, 214
- SUBSTR 211
- substring 178
- SUM 127, 213
- Superuser 196
- synonym 328
- Synonym Dictionary 330
- Syntaxanalyzer 341
- Syntaxparser 341
- sysctl 423

T

- Tabellen verknüpfen 83
- Tabellenpartitionierung 392, 397
- Tabellenpartitionierung mit Rules 403
- Tabellenpartitionierung mit Trigger 398
- Table Of Contents (TOC) 386
- TABLESPACE 395
- Tablespaces 392
- Tar-Archiv 381
- tcl 409
- TCP/IP 370
- template0 99
- template1 99
- Templates 99
- TEXT 72
- text 107
- Thesaurus Dictionary 328, 331
- time 108
- timestamp 108
- to_char 215
- to_tsquery() 302, 304
- to_tsvector() 301, 302, 303

- TOC 386
- .toc 387
- TOC Entries 386
- Token 300
- token_type 337
- Token-Klassen 300
- Transaktion 58
- Transaktionen 154
 - parallele* 158
 - PHP-Klasse* 160
 - Sichtbarkeit* 158
- trust 371
- ts_debug() 316, 320
- ts_headline() 325
- ts_lexise() 318
- ts_rank() 323
- ts_rank_cd() 323
- ts_rewrite() 315
- ts_stat() 316
- tsquery 302
- tsvector 302
- tsvector_update_trigger() 314
- Tupel 61
- Typecasting 178

U

- Umwandlung von Arrays 117
- UNION 62, 175
 - UNION ALL* 176
- Unix-Domain-Sockets 370
- unscharfe Suche 163
- Unterabfragen 166
- UPDATE 77
- UPPER 211
- upper 210
- User Defined Functions 286
- useradd 410
- Users 368
- USING 266
 - DETAIL* 254
 - ERRCODE* 254
 - HINT* 254
 - MESSAGE* 254
 - USING* 254

V

- vacuum_cost_delay 377
- VACUUM 222, 340, 375
- VACUUM FULL 340, 376
- VACUUM VERBOSE 378
- vacuum_cost_limit 377

Index

vacuum_freeze_min_page 378
vacuumdb 375
vacuumlo 375
VARCHAR 71
varchar 107
Verarbeitungssicherheit 154
VERBOSE 377
Verknüpfung von Tabellen 138
verschachtelte Schleife 277
Views 186
 aktualisierbare 189
 CREATE VIEW 187
VirtualBox 390
VMware 390
vollständige Schreibweise 77
Volltextsuche 299, 352
 Aufbau einer Suche 309
 Gewichtung 310
 Konfiguration 334
 Oleg Bartunov 299
 Operatoren 319
 Ranking 323
 Suche starten 321
 Suchmaschine 299
 Teodor Sigaev 299
 Token 300
 Token-Klassen 300
 Trigger-Prozedur 313

Tsearch2 299
Vektorraummodell 301
Volltextsuche 299

W

Wahrheitswerte 122
WAL-Archivierung 391
WARNING 254
WHERE 82
WHILE 270, 271
Windows-Kommandozeile 420
Word Stemming 300

X

XID 374
XMAX 374
XMIN 374

Z

Zahlen beliebiger Präzision 101
Zeichenkettentypen 107
Zeit 108
zlib 408
zweidimensionale Objekte 111