

# PostgreSQL 10

Praxisbuch für Administratoren und Entwickler

Bearbeitet von  
Von: Lutz Fröhlich

1. Auflage 2018. Buch. 489 S. Gebunden  
ISBN 978 3 446 45395 1  
Format (B x L): 18.1 x 24.4 cm  
Gewicht: 1044 g

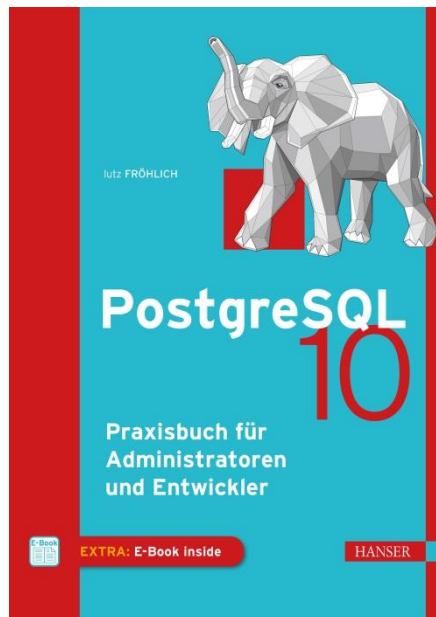
[Weitere Fachgebiete > EDV, Informatik > Programmiersprachen: Methoden > Datenbankprogrammierung](#)

schnell und portofrei erhältlich bei

  
DIE FACHBUCHHANDLUNG

Die Online-Fachbuchhandlung [beck-shop.de](http://beck-shop.de) ist spezialisiert auf Fachbücher, insbesondere Recht, Steuern und Wirtschaft. Im Sortiment finden Sie alle Medien (Bücher, Zeitschriften, CDs, eBooks, etc.) aller Verlage. Ergänzt wird das Programm durch Services wie Neuerscheinungsdienst oder Zusammenstellungen von Büchern zu Sonderpreisen. Der Shop führt mehr als 8 Millionen Produkte.

# HANSER



## Leseprobe

zu

## PostgreSQL 10

von Lutz Fröhlich

ISBN (Buch): 978-3-446-45395-1

ISBN (E-Book): 978-3-446-45641-9

Weitere Informationen und Bestellungen unter

<https://www.hanser-fachbuch.de/>

sowie im Buchhandel

© Carl Hanser Verlag, München

# Inhalt

<b>1</b>	<b>Einführung und Geschichte</b> .....	<b>1</b>
1.1	Die Geschichte von PostgreSQL .....	2
1.2	Verwendete Version .....	3
1.3	Konventionen .....	3
1.4	Software und Skripte .....	3
<b>2</b>	<b>Installation aus Paketen und Quellcode</b> .....	<b>5</b>
2.1	Paketinstallation .....	5
2.1.1	Paketinstallation unter Linux .....	5
2.1.2	Paketinstallation unter Windows .....	6
2.2	Installation aus dem Quellcode .....	8
2.2.1	Installation aus dem Quellcode unter Linux .....	8
2.2.2	Installation aus dem Quellcode unter Windows .....	9
2.3	Erste Schritte .....	12
<b>3</b>	<b>Upgrade auf Version 10</b> .....	<b>17</b>
3.1	Upgrade mit pg_dumpall .....	17
3.2	Upgrade mit pg_upgrade .....	19
3.3	Migration nach Native Partitioning .....	21
3.4	Regressionstests .....	23
<b>4</b>	<b>Die Architektur von PostgreSQL</b> .....	<b>25</b>
4.1	Überblick .....	25
4.2	Memory und Prozesse .....	26
4.2.1	Hintergrundprozesse .....	27
4.2.2	Der Shared Memory .....	29
4.3	VACUUM .....	37
4.4	Cluster, Datenbanken und Tabellen .....	40

<b>5</b>	<b>Server und Datenbanken administrieren</b>	<b>45</b>
5.1	Parameter-Einstellungen	45
5.1.1	Einstellungen im Betriebssystem	45
5.1.2	Cluster-Einstellungen	47
5.1.3	Gebietsschema und Zeichensatz	57
5.2	Datenbanken verwalten	60
5.3	Konkurrenz	63
5.4	Die WAL-Archivierung einschalten	66
5.5	Wartungsaufgaben	68
5.5.1	VACUUM	68
5.5.2	ANALYZE	71
5.6	Nützliche Skripte und Hinweise	71
5.6.1	Eine Passwortdatei verwenden	72
5.6.2	Welche Parameter sind Nicht-Standard?	72
5.6.3	Eine Session killen	73
5.6.4	Eine Tabelle nach Excel kopieren	73
5.6.5	Die Datei .psqlrc	74
5.6.6	Einen WAL-Switch manuell auslösen	75
5.6.7	Die PostgreSQL-Server-Logdatei in eine Tabelle laden	75
5.6.8	Automatisches Rotieren von Logdateien	76
5.6.9	Nicht verwendete Indexe identifizieren	76
5.6.10	Microsoft Excel als Datenbank-Client	77
5.6.11	Den Inhalt der Kontrolldatei ausgeben	79
5.6.12	Platzverbrauch von Tabellen	80
5.6.13	Die Anzahl von Verbindungen begrenzen	80
5.6.14	Tabellen und Indexe in einen anderen Tablespace legen	81
5.6.15	Temporäre Dateien verwalten	82
5.6.16	Lang laufende SQL-Anweisungen	82
5.7	Beispielschemata	83
<b>6</b>	<b>Neue Features</b>	<b>85</b>
6.1	Neue Features in Version 10	85
6.1.1	Native Table Partitioning	86
6.1.2	Paralleles SQL	88
6.1.3	Logische Replikation	88
6.1.4	Änderungen der Architektur	90
6.1.5	SQL-Anweisungen	92
6.1.6	Monitoring	98
6.1.7	Werkzeuge	99
6.1.8	Konfigurationsparameter	102
6.2	Neue Features in den Versionen 9.2 bis 9.6	102
6.2.1	Backend	102
6.2.2	Replikation	103
6.2.3	Performance	104

<b>7</b>	<b>Sicherung und Wiederherstellung</b>	<b>105</b>
7.1	Online-Sicherung mit Point-in-time-Recovery	106
7.2	Offline-Sicherung auf Dateisystemebene	111
7.3	SQL Dump	111
<b>8</b>	<b>Sicherheit und Überwachung</b>	<b>117</b>
8.1	Sicherheit	118
8.1.1	Rollen und Privilegien	118
8.1.2	Authentifizierung und Zugangskontrolle	125
8.1.3	Rechteverwaltung	127
8.1.4	Sichere Verbindungen	132
8.1.5	Out-of-the-box-Sicherheit	136
8.1.6	Hacker-Attacken abwehren	137
8.2	Überwachung	142
8.2.1	Auditing	142
8.2.2	Monitoring	144
<b>9</b>	<b>Replikation zwischen Clustern</b>	<b>151</b>
9.1	Physische Replikation	152
9.1.1	Vorbereitung und Planung	152
9.1.2	Konfiguration und Aktivierung	153
9.1.3	Kaskadenförmige Replikation	157
9.1.4	Hot Standby	158
9.1.5	Synchrone Replikation	159
9.1.6	Die Replikation überwachen	161
9.1.7	Failover und Switchover	163
9.2	Logische Replikation	168
9.3	Logical Decoding	174
9.3.1	Logical Decoding mit Java als Consumer	175
<b>10</b>	<b>Das Regelsystem</b>	<b>179</b>
10.1	Das Regelsystem für SELECT-Anweisungen	180
10.2	Das Regelsystem für DML-Anweisungen	181
10.3	Regeln und Views	185
<b>11</b>	<b>Performance Tuning</b>	<b>187</b>
11.1	Out-of-the-box-Tuning	187
11.1.1	Goldene Regeln für neue Server und Datenbanken	188
11.1.2	Das Utility „pgTune“	189
11.1.3	Optimierung von Memory-Parametern	190
11.2	Performance-Analyse	193
11.2.1	Analyse mit dem „Statistics Collector“	193
11.2.2	Der Background Writer	200
11.2.3	Analyse mit „pgstatspack“	201

<b>12</b>	<b>Optimierung von SQL-Anweisungen</b>	<b>205</b>
12.1	Ausführungsschritte	206
12.2	Der SQL-Optimizer	207
12.3	Statistiken und Histogramme	208
12.4	Zugriffsmethoden	211
12.5	Join-Methoden	212
12.6	SQL-Optimierung	215
12.6.1	Der EXPLAIN-Befehl	216
12.6.2	Ausführungspläne verstehen und optimieren	219
<b>13</b>	<b>Einsatz großer Datenbanken</b>	<b>229</b>
13.1	Partitionierung von Tabellen	230
13.1.1	Native Table Partitioning	230
13.2	Paralleles SQL	233
13.3	Materialized Views	238
13.4	BRIN-Indexe	240
<b>14</b>	<b>PostGIS</b>	<b>245</b>
14.1	PostGIS und PostgreSQL	245
14.2	PostGIS installieren	246
14.2.1	Paketorientierte Installation	246
14.2.2	Installation aus dem Quellcode	249
14.3	Erste Schritte mit PostGIS	250
14.4	PostGIS in der Praxis anwenden	255
<b>15</b>	<b>Applikationen für PostgreSQL entwickeln</b>	<b>261</b>
15.1	Applikationsdesign	261
15.2	Entwicklungswerkzeuge	265
15.3	PostgreSQL Extensions	266
<b>16</b>	<b>SQL-Erweiterungen</b>	<b>269</b>
16.1	Datentypen	269
16.2	Funktionen und Sprachen	270
16.2.1	SQL-Funktionen	271
16.2.2	Funktionen mit prozeduralen Programmiersprachen	275
16.2.3	C-Funktionen	279
16.3	Operatoren	284
16.4	Das Extension-Netzwerk	286
16.4.1	Extensions entwickeln	287
16.4.2	Extensions publizieren	290

<b>17</b>	<b>PL/pgSQL-Funktionen und Trigger</b> .....	<b>295</b>
17.1	PL/pgSQL-Funktionen .....	295
17.1.1	Abfragen und Resultsets .....	299
17.1.2	Cursor verwenden .....	301
17.1.3	DML-Anweisungen .....	303
17.1.4	Dynamische SQL-Anweisungen .....	305
17.1.5	Fehlerbehandlung .....	306
17.2	Trigger .....	307
<b>18</b>	<b>Embedded SQL (ECPG)</b> .....	<b>311</b>
<b>19</b>	<b>Java-Programmierung</b> .....	<b>321</b>
19.1	Eine Entwicklungsumgebung einrichten .....	321
19.2	Verarbeitung von Resultsets .....	324
19.3	DML-Anweisungen und Transaktionen .....	327
19.4	Bindevariablen verwenden .....	329
19.5	Java und Stored Functions .....	330
19.6	Large Objects .....	334
19.7	JDBC-Tracing .....	338
<b>20</b>	<b>Die C-Library libpq</b> .....	<b>341</b>
20.1	Die Entwicklungsumgebung einrichten .....	341
20.2	Programme mit „libpq“ erstellen .....	346
<b>21</b>	<b>PHP-Applikationen</b> .....	<b>359</b>
21.1	Installation und Konfiguration .....	360
21.2	Applikationen mit PHP entwickeln .....	362
21.3	Die PDO-API .....	370
<b>22</b>	<b>Client-Programmierung mit Perl-DBI</b> .....	<b>373</b>
22.1	SELECT-Anweisungen und Resultsets .....	376
22.2	DML-Anweisungen .....	380
22.3	Bindevariablen verwenden .....	381
22.4	Fehlerbehandlung und Tracing .....	383
22.5	Nützliche Skripte und Beispiele .....	386
22.5.1	Mehrere Server abfragen .....	386
22.5.2	Parallele Verbindungen .....	387
22.5.3	Large Objects verarbeiten .....	390
22.5.4	Asynchrone Abfragen .....	390
22.5.5	Datenbanken vergleichen .....	391
<b>23</b>	<b>Large Objects</b> .....	<b>395</b>

<b>24</b>	<b>PostgreSQL in die IT-Landschaft einbinden</b> .....	<b>401</b>
24.1	Features und Funktionen .....	401
24.2	Datensicherheit und Wiederherstellung .....	402
24.3	Desaster Recovery .....	403
24.4	Überwachung .....	404
24.5	Administrierbarkeit .....	404
24.6	Verfügbarkeit .....	405
24.7	Datensicherheit und Auditing .....	406
24.8	Performance und Skalierbarkeit .....	406
24.9	Schnittstellen und Kommunikation .....	407
24.10	Support .....	408
24.11	Fazit .....	408
<b>25</b>	<b>Migration von MySQL-Datenbanken</b> .....	<b>409</b>
25.1	Unterschiede zwischen MySQL und PostgreSQL .....	409
25.2	Eine Migration durchführen .....	411
<b>26</b>	<b>Von Oracle nach PostgreSQL migrieren</b> .....	<b>417</b>
26.1	Die Migration planen .....	417
26.2	Unterschiede zwischen Oracle und PostgreSQL .....	419
26.2.1	Unterschiede der Datentypen .....	419
26.2.2	Syntaktische und logische Unterschiede .....	420
26.2.3	Steigerung der Kompatibilität von PostgreSQL .....	423
26.3	Portierung von Oracle PL/SQL .....	424
26.4	Tools zur Unterstützung der Migration .....	427
26.5	Technisches Vorgehen .....	429
26.6	Ein Migrationsbeispiel .....	429
26.6.1	Manuelle Migration .....	430
26.6.2	Migration unter Verwendung von „Ora2Pg“ .....	436
26.6.3	Große Tabellen laden .....	440
<b>27</b>	<b>Replikation zwischen Oracle und PostgreSQL</b> .....	<b>443</b>
27.1	Datenbanklink zwischen Oracle und PostgreSQL .....	443
27.2	Replikation mit Oracle XStream .....	449
<b>28</b>	<b>PostgreSQL in der Cloud</b> .....	<b>463</b>
28.1	Private Cloud .....	464
28.2	Public Cloud .....	466
	<b>Index</b> .....	<b>469</b>



## Einsatz großer Datenbanken

Die durchschnittliche Größe von Datenbanken hat in den letzten Jahren deutlich zugenommen und der Trend für die Zukunft zeigt weiter nach oben. Datenbanksysteme sind der Schlüssel und die Basis aller Informationen. Es gibt kaum eine Applikation, die keine Datenbank verwendet. Die Möglichkeit der strukturierten und sicheren Ablage hat dazu geführt, dass kaum Daten außerhalb von Datenbanken gespeichert werden. Relationale Datenbanksysteme sind immer noch in der Überzahl, auch wenn sich die Art und Weise der Speicherung von Daten in der Zukunft ändern wird.

PostgreSQL ist in der Lage, große Datenbanken robust und performant zu verwalten. Für alle Systeme gilt, dass der Umgang mit sehr großen Datenbanken eigenen Gesetzen unterliegt und spezielle Features benötigt werden.

Die Planung für den Einsatz großer Datenbanken beginnt beim Design der Infrastruktur sowie der Applikationen. Häufig, aber nicht ausschließlich, begegnet man großen Datenbanken im Data Warehouse-Umfeld.

Das ständige Wachstum des Datenbestands sowie der Anzahl von konkurrierenden Benutzern stellt Hersteller, Designer und Administratoren immer wieder vor neue Herausforderungen. Dazu kommt der Umstand, dass die Hardwarekomponenten in den letzten Jahren kaum schneller geworden sind. Die Taktfrequenz von CPU-Kernen stagniert seit Jahren. Im Zuge der „Green IT“ ist man eher dazu übergegangen, die Single CPU Clockspeed zu reduzieren, um den Energieverbrauch und die Wärmeentwicklung zu senken. Bei den I/O-Systemen drängen immer mehr Solid State Disks auf den Markt, die einen besseren Durchsatz gegenüber Disk-Spindeln haben, aber auch einige Nachteile mit sich bringen.

Um große Datenmengen in vertretbare Zeit verarbeiten zu können, ist man dazu übergegangen, intelligente Softwarelösungen zu entwickeln, die Systeme größer zu machen und stark zu parallelisieren.

Die Firma Oracle hat die Oracle Database Machine (ODMExadata) auf den Markt gebracht. Diese enthält unter vielen anderen Lösungen Smart Scans und Storage-Indexe, die physikalische I/O-Operationen stark reduzieren und eine virtuelle Durchsatzrate von vielen Giga-byte pro Sekunde ermöglichen. In-Memory-Datenbanksysteme sind populär, wenngleich noch teuer, da in der Regel die gesamte Datenbank in den Memory passen muss.

Features für die Parallelisierung von Prozessen und Operationen sind eine notwendige Voraussetzung für den Einsatz großer Datenbanken. Mit dem Wachstum des Datenbestands ist die Erhöhung des Parallelitätsgrads eine gute Möglichkeit für die Verbesserung der Skalierbarkeit.

PostgreSQL war in dieser Hinsicht nicht untätig und hat eine ganze Reihe neuer Features eingeführt. Zu den wichtigsten gehören:

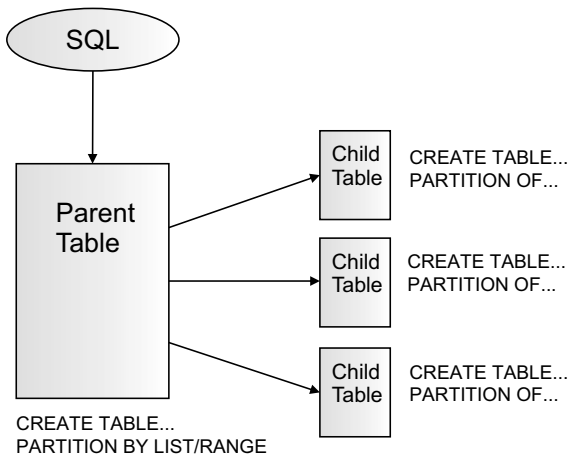
- Partitionierung: Einführung des Native Partitioning in PostgreSQL 10.
- Paralleles SQL: Ständige Erweiterung der Funktionalität für paralleles SQL in den Versionen 9 und 10.
- Materialized Views: Möglichkeit, mit aggregierten Daten zu arbeiten.
- BRIN-Indexe: Intelligente Lösung, um große Datenmengen zu scannen.

## ■ 13.1 Partitionierung von Tabellen

Als bisherige Methode zur Partitionierung von Tabellen wurde die Funktionalität „Table Inheritance“ verwendet. Dabei wurden mehrere Tochtertabellen angelegt und die Konsistenz mithilfe von Check Constraints und Triggern gewährleistet. Diese Technologie hat einige entscheidende Nachteile. So müssen zum Beispiel INSERT-Anweisungen über Trigger an die Tochtertabellen weitergegeben werden, was klare Performancenachteile mit sich bringt. Mit der Version 10 wurde das „Native Table Partitioning“ eingeführt.

### 13.1.1 Native Table Partitioning

Die Technologie von PostgreSQL 10 ist wesentlich performanter und verwendet eine effiziente Verteilungsmethode auf die Tochtertabellen. Es müssen keine Constraints und Trigger mehr gebildet werden. Das verbessert auch die Zuverlässigkeit und bietet zusätzliche Verwaltungsoptionen.



**Bild 13.1** Native Table Partitioning

Es stehen zwei Arten zur Verfügung: List Partitioning und Range Partitioning. Beim List Partitioning werden den einzelnen Partitionen bestimmte Werte zugeordnet. Es gibt die Einschränkung, dass die Partitionierung nur auf Basis einer Spalte erfolgen kann. Zunächst wird die Elterntabelle mit der PARTITION BY-Klausel angelegt. Die Tabelle ist noch nicht funktionsfähig, es können keine Sätze eingefügt werden. Im nächsten Schritt werden die Tochtertabellen angelegt. Darin werden letztendlich die Daten gespeichert. Die Elterntabelle dient der Kommunikation mit der SQL-Anweisung sowie dem Routing in die einzelnen Partitionen. In Listing 13.1 finden Sie ein Beispiel für eine partitionierte Tabelle mit List Partitioning.

**Listing 13.1** Eine Tabelle mit List Partitioning anlegen

```
postgres@[local]:5432[postgres]> CREATE TABLE employees (
> employee_id INT,
> department_id INT,
> first_name VARCHAR(30),
> last_name VARCHAR(30),
> hr_id INT,
> salary INT)
> PARTITION BY LIST(department_id);
CREATE TABLE
(postgres@[local]:5432)[postgres > CREATE TABLE employees_p10
> PARTITION OF employees
> FOR VALUES IN (10);
CREATE TABLE
. . .
```

In „psql“ kann die Definition der Elterntabelle einschließlich der zugehörigen Partitionen angezeigt werden (siehe Listing 13.2). Eine partitionierte Tabelle kann über die Spalte „relkind“ im View „pg\_class“ identifiziert werden.

**Listing 13.2** Definition einer partitionierten Tabelle anzeigen

```
(postgres@localhost:5432)[postgres]> \d+ employees
Table "public.employees"
  Column          | Type          | Collation | Nullable |
-----+-----+-----+-----+
 employee_id     | integer      |           |          |
 department_id   | integer      |           |          |
 first_name      | character varying(30) |           |          |
 last_name       | character varying(30) |           |          |
 hr_id           | integer      |           |          |
 salary          | integer      |           |          |
Partition key: LIST (department_id)
Partitions: employees_p10 FOR VALUES IN (10),
            employees_p20 FOR VALUES IN (20),
            employees_p30 FOR VALUES IN (30)
(postgres@localhost:5432)[postgres]> SELECT relname, relpages, relkind
> FROM pg_class WHERE relname like 'employees%';
 relname | relpages | relkind
-----+-----+-----+
 employees | 0 | p
 employees_p10 | 12346 | r
 employees_p20 | 12346 | r
 employees_p30 | 12346 | r
```



**HINWEIS:** Beim List Partitioning ist zu beachten, dass für alle potenziellen Werte des Partitionsschlüssels eine Partition existieren muss. Andernfalls kommt es zum Fehler „ERROR: no partition of relation „employees“ found for row“.

Für das Range Partitioning können Bereiche, also Minimal- und Maximalwert für den Partitionsschlüssel, angegeben werden. Im Gegensatz zum List Partitioning können mehrere Spalten als Partitionsschlüssel angegeben werden. Das Anlegen ist analog zum List Partitioning. Zuerst muss die Elterntabelle und danach die partitionierten Tabellen angelegt werden (siehe Beispiel in Listing 13.3).

**Listing 13.3** Eine Tabelle mit Range Partitioning anlegen

```
(postgres@localhost:5432)[postgres]> CREATE TABLE sales (
> sales_id INT,
> customer_id INT,
> product_id INT,
> sales_date DATE,
> amount NUMERIC(12,2)
> PARTITION BY RANGE (sales_date);
CREATE TABLE
(postgres@localhost:5432)[postgres]> CREATE TABLE sales_2017_01
> PARTITION OF sales
> FOR VALUES FROM ('2017-01-01') TO ('2017-01-31');
CREATE TABLE
(postgres@localhost:5432)[postgres]> CREATE TABLE sales_2017_02
> PARTITION OF sales
> FOR VALUES FROM ('2017-02-01') TO ('2017-02-28');
CREATE TABLE
. . .
(postgres@localhost:5432)[postgres]> \d+ sales
Table "public.sales"
  Column      |      Type      | Collation | Nullable |
-----+-----+-----+-----+
 sales_id     | integer        |           |          |
 customer_id  | integer        |           |          |
 product_id   | integer        |           |          |
 sales_date   | date           |           |          |
 amount       | numeric(12,2) |           |          |
Partition key: RANGE (sales_date)
Partitions: sales_2017_01 FOR VALUES FROM ('2017-01-01') TO ('2017-01-31'),
            sales_2017_02 FOR VALUES FROM ('2017-02-01') TO ('2017-02-28')
```

Indexe können an den Tochtertabellen angelegt werden, jedoch nicht an der Elterntabelle. Es ist gestattet, aber nicht notwendig, einen Index für den Partitionsschlüssel anzulegen. Es gibt kein Werkzeug, um sicherzustellen, dass Indexe auf allen Tochtertabellen angelegt sind. Diese müssen individuell gewartet und kontrolliert werden.

Partitionen können mit einem DROP-Befehl gelöscht werden. Mit dieser Methode können zum Beispiel historische Daten sehr effizient und sicher entfernt werden:

```
postgres@localhost:5432)[postgres]> DROP TABLE sales_2017_01;
DROP TABLE
```

Sollen die Daten noch erhalten bleiben, jedoch nicht mehr in der Elterntabelle auftauchen, dann kann eine Partition mit dem DETACH-Befehl entfernt werden (siehe Listing 13.4).

**Listing 13.4** Eine Partition von der Tabelle entfernen

```
(postgres@localhost:5432)[postgres]> SELECT count(*) FROM employees;
count
-----
3000000
(postgres@localhost:5432)[postgres]> ALTER TABLE employees DETACH PARTITION
employees_p30;
ALTER TABLE
(postgres@localhost:5432)[postgres]> SELECT count(*) FROM employees;
count
-----
2000000
(postgres@localhost:5432) postgres]> SELECT count(*) FROM employees_p30;
count
-----
1000000
```

Für das Native Table Partitioning gelten die folgenden Einschränkungen:

- Primärschlüssel werden auf partitionierten Tabellen nicht unterstützt. Das hat zur Folge, dass auch keine Fremdschlüssel auf anderen Tabellen, die auf die partitionierte Tabelle verweisen, angelegt werden können.
- Ein Update auf den Partitionsschlüssel, der das Umschichten des Satzes in eine andere Partition zur Folge hat, bricht ab mit der folgenden Fehlermeldung: „ERROR: new row for relation“ sales\_2017\_01 „violates partition constraint“.
- Trigger müssen auf den Tochartabellen und können nicht zentral auf der Elterntabelle definiert werden.



**HINWEIS:** Falls Sie bereits Inheritance Partitioning einsetzen, dann sollten Sie die Tabellen auf Native Partitioning umstellen. Einerseits wird Inheritance Partitioning nicht weiterentwickelt werden und andererseits ist es sinnvoll, die Performancevorteile von Native Partitioning zu nutzen. Hinweise zur Migration auf Native Partitioning finden Sie in Kapitel 3, „Upgrade auf Version 10“.

## ■ 13.2 Paralleles SQL

Die Entscheidung, ob eine SQL-Anweisung oder deren Operationsstufen parallel ausgeführt werden, trifft der Query-Planer (Optimizer). Die folgenden Parameter beeinflussen die Entscheidung, ob parallele Ausführungspläne generiert werden können:

- *max\_parallel\_workers\_per\_gather*: Legt die maximal Anzahl von Worker-Prozessen fest, die für einen einzelnen Ausführungsschritt eines Query-Plans gestartet werden können. Der Standardwert ist 2. Diese Prozesse werden auch vom Pool, der durch den Parameter „max\_worker\_processes“ begrenzt ist, abgezogen. Eine weitere Begrenzung stellt der übergeordnete Wert „max\_worker\_processes“ dar.

- *dynamic\_shared\_memory\_type*: Darf nicht auf den Wert „none“ gesetzt werden, damit paralleles SQL ausgeführt wird. Die parallelen Prozesse benötigen Dynamic Shared Memory, um Daten gegenseitig auszutauschen.

Der EXPLAIN-Befehl gibt mit dem Ausführungsplan die Anzahl von parallelen Worker-Prozessen, die der Query-Planner bestimmt hat, zurück. Im Beispiel in Listing 13.5 hat sich der Query-Planner für zwei Worker-Prozesse entschieden, obwohl die Hardware mehr Systemressourcen zur Verfügung hat. Die Begrenzung kommt vom Parameter „max\_parallel\_workers\_per\_gather“, der standardmäßig auf dem Wert 2 steht.

**Listing 13.5** Paralleles SQL auf einer großen Tabelle

```
(postgres@[local]:5432)[hanser]> EXPLAIN
SELECT count(*) FROM big
WHERE cont LIKE '%text%';

                                QUERY PLAN
-----
Finalize Aggregate (cost=591117.23..591117.24 rows=1 width=8)
-> Gather (cost=591117.01..591117.22 rows=2 width=8)
    Workers Planned: 2
-> Partial Aggregate (cost=590117.01..590117.02 rows=1 width=8)
    -> Parallel Seq Scan on big (cost=0.00..569283.68 rows=8333334 width=0)
        Filter: ((cont)::text ~ '%text%'::text)
```

Nach einer Erhöhung des Parameterwertes auf 8 wählt der Optimizer sechs parallele Worker-Prozesse (siehe Listing 13.6). Die Formulierung „Workers Planned“ im Ausführungsplan ist durchaus zutreffend. Es handelt sich um eine Einschätzung des Optimizers und Berücksichtigung der begrenzenden Parameter. Wenn während der Ausführung nicht genügend parallele Prozesse mehr zur Verfügung stehen, werden entsprechend weniger gestartet. Eine solche Begrenzung kann durch die Werte der Parameter „max\_worker\_processes“ oder „max\_parallel\_workers“ ausgelöst sein.

**Listing 13.6** Den Parallelitätsgrad einer SQL-Abfrage erhöhen

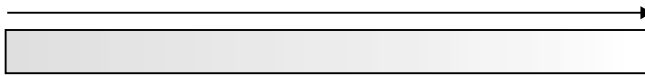
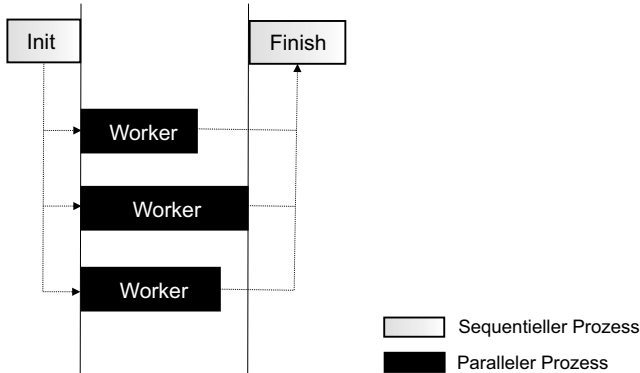
```
(postgres@[local]:5432)[hanser]> EXPLAIN
SELECT count(*) FROM big
WHERE cont LIKE '%text%';

                                QUERY PLAN
-----
Finalize Aggregate (cost=516117.63..516117.64 rows=1 width=8)
-> Gather (cost=516117.01..516117.62 rows=6 width=8)
    Workers Planned: 6
-> Partial Aggregate (cost=515117.01..515117.02 rows=1 width=8)
    -> Parallel Seq Scan on big (cost=0.00..506783.67 rows=3333334 width=0)
        Filter: ((cont)::text ~ '%text%'::text)
```

Im vorliegenden Beispiel wurden folgenden Antwortzeiten erzielt:

- Parallel mit 6 Worker-Prozessen: Time: 906.751 ms
- Nicht parallel: Time: 2957.980 ms (00:02.958)

Mit sechs Worker-Prozessen hat sich die Ausführungszeit in etwa gedrittelt. Paralleles SQL skaliert also nicht linear. Dies liegt in der Natur der parallelen Verarbeitung. Jeder parallelisierte Prozess unterteilt sich in einen Anteil von parallelen Teilen und Teilen, die nicht parallelisiert werden können und damit sequenziell laufen (siehe Bild 13.2).

**Sequentieller Prozess:****Parallelierter Prozess:****Bild 13.2** Das Prinzip parallelisierter Prozesse

Zu den sequenziellen Anteilen eines parallelisierten Prozesses gehören mindestens ein Initial- und ein Finish-Prozess. Im Fall eines parallelen Scans einer Tabelle muss die Arbeit an die parallelen Prozesse verteilt und die Ergebnisse müssen nach dem Scan wieder zusammengeführt werden.



**TIPP:** Die Parameter zur Begrenzung der Anzahl von parallelen Prozessen schützen das System vor Überlastung. Im Data Warehouse-Umfeld ist eine CPU-Auslastung von 100 Prozent prinzipiell kein Problem. Der Server muss jedoch vor einer Überlastung geschützt werden. Wird die Run Queue zu lang, gehen die Antwortzeiten für alle Prozesse in den Keller.

PostgreSQL ist in der Lage, die folgenden Aufgaben zu parallelisieren:

- Parallele Scans
- Parallele Joins
- Parallele Aggregation

Die folgenden Scan-Operationen können in PostgreSQL 10 parallel ausgeführt werden:

- *Parallel Sequential Scan:* Die Blöcke einer Tabelle werden auf die Worker-Prozesse verteilt.
- *Parallel Bitmap Heap Scan:* Der Master-Prozess durchsucht den Index und erstellt eine Bitmap für die Blöcke, die gelesen werden müssen. Diese werden auf die Worker-Prozesse aufgeteilt und parallel gescannt.
- *Parallel Index Scan:* Die Arbeit wird auf die Worker-Prozesse verteilt. Das Durchsuchen der Indexteile erfolgt dann parallel. Die Worker-Prozesse liefern die zugehörigen Tuples an den Master-Prozess.
- *Parallel Index-Only Scan:* Funktioniert wie ein Parallel Index Scan. Es werden keine Tuples gelesen, da sich alle Informationen im Index selbst befinden.

Parallele Joins sind seit der Version 10 für alle drei Methoden – Nested Loops, Hash Joins und Merge Joins – möglich. Alle Join-Methoden benutzen einen inneren Loop, die sogenannte Driving Table. Der innere Loop kann auch das Ergebnis eines vorangegangenen Joins sein. Jeder Worker-Prozess durchläuft dabei den ganzen inneren Loop.

Eine parallele Aggregation erfolgt in zwei Etappen. Jeder der parallelen Worker-Prozesse führt die Aggregation für seinen Bereich durch. Im zweiten Schritt werden die Ergebnisse an den Master-Prozess übergeben. Der Master-Prozess bringt die Ergebnisse zusammen.

Aktuell gibt es folgende Einschränkungen für die Ausführung von parallelem SQL:

- Die SQL-Anweisung schreibt Daten oder sperrt Tabellen oder Datensätze.
- Die SQL-Anweisung benutzt Funktionen, die als „PARALLEL UNSAFE“ markiert sind.
- Das SQL läuft innerhalb einer Anweisung, die bereits parallel ausgeführt wird.

Auch wenn der Query Optimizer einen parallelen Plan vorgegeben hat, kann es passieren, dass die Ausführung nicht parallel erfolgt. Diese Situation kann eintreten, wenn zum Beispiel die maximale Anzahl von Worker-Prozessen erreicht ist.

Parallel Query funktioniert seit Version 10 auch für Prepare- und Execute Anweisungen, Bitmap Joins und Merge Joins. Weiterhin können Index Scans auch parallelisiert werden. Die Entscheidung, ob SQL-Abfragen parallel ausgeführt werden, trifft der Query Optimizer. In Listing 13.7 finden Sie ein Beispiel für Prepare und Execute.



**HINWEIS:** Tabellen müssen nicht zwangsläufig partitioniert sein. Paralleles SQL funktioniert auch mit nicht partitionierten Tabellen.

#### Listing 13.7 Paralleles SQL mit Prepare und Execute

```
(postgres@localhost:5432)[postgres]> PREPARE c1 AS
SELECT entry_date,count(*) FROM order_entry GROUP BY entry_date;
PREPARE
(postgres@localhost:5432)[postgres]> EXPLAIN EXECUTE c1;
QUERY PLAN
-----
Finalize GroupAggregate (cost=76779.30..76784.30 rows=200 width=12)
Group Key: order_entry_201201.entry_date
-> Sort (cost=76779.30..76780.30 rows=400 width=12)
Sort Key: order_entry_201201.entry_date
-> Gather (cost=76720.01..76762.01 rows=400 width=12)
Workers Planned: 2
-> Partial HashAggregate (cost=75720.01..75722.01 rows=200 width=12)
Group Key: order_entry_201601.entry_date
-> Append (cost=0.00..63220.00 rows=2500002 width=4)
-> Parallel Seq Scan on order_entry_201201 (cost=0.00..10536.67 rows=416667 width=4)
-> Parallel Seq Scan on order_entry_201202 (cost=0.00..10536.67 rows=416667 width=4)
-> Parallel Seq Scan on order_entry_201203 (cost=0.00..10536.67 rows=416667 width=4)
-> Parallel Seq Scan on order_entry_201204 (cost=0.00..10536.67 rows=416667 width=4)
-> Parallel Seq Scan on order_entry_201205 (cost=0.00..10536.67 rows=416667 width=4)
-> Parallel Seq Scan on order_entry_201206 (cost=0.00..10536.67 rows=416667 width=4)
```



Das Beispiel in Listing 13.8 zeigt einen parallelen Index-Only Scan für eine nicht partitionierte Tabelle. Im Data Warehouse-Umfeld werden häufig Index-Only Scans benötigt. PostgreSQL 10 unterstützt nun auch da parallele Abfragen.

**Listing 13.8** Paralleler Index-Only Scan

```
(postgres@localhost:5432)[postgres]> \d+ sales
      Table "public.sales"
  Column          |          Type          |
-----+-----+-----
 sales_id         |      bigint            |
 customer_id      |      integer           |
 product_id       |      integer           |
 sales_date       | timestamp with time zone |
 amount           |      numeric           |
Indexes:
    "pk_sales" PRIMARY KEY, btree (sales_
    "idx_sales" btree (product_id)
(postgres@localhost:5432)[postgres]> EXPLAIN
> SELECT count(*) FROM sales WHERE sales_id > 10 AND sales_id < 400000;
      QUERY PLAN
-----
Finalize Aggregate (cost=17945.78..17945.79 rows=1 width=8)
-> Gather (cost=17945.57..17945.78 rows=2 width=8)
    Workers Planned: 4
    -> Partial Aggregate (cost=16945.57..16945.58 rows=1 width=8)
        -> Parallel Index Only Scan using pk_sales on sales
            (cost=0.44..16419.32 rows=210497 width=0)
                Index Cond: ((sales_id > 10) AND (sales_id < 400000))
```

Um die Arbeit zwischen den Worker-Prozessen parallelisieren zu können, muss die Driving Table für die parallele Weiterverarbeitung aufteilbar sein (siehe Listing 13.9). In diesem Beispiel wird ein serieller Bitmap Index Scan ausgeführt. Dabei wird eine Datenstruktur im Shared Memory mit allen Blöcken aufgebaut, die gescannt werden müssen. Die Worker-Prozesse können dann den Heap Scan parallel ausführen.

**Listing 13.9** Paralleler Bitmap Heap Scan

```
(postgres@localhost:5432)[postgres]> EXPLAIN
> SELECT count(*) FROM sales
> WHERE product_id < 100
> GROUP BY product_id;
      QUERY PLAN
-----
Finalize GroupAggregate (cost=251653.96..251666.46 rows=500 width=12)
 Group Key: product_id
-> Sort (cost=251653.96..251656.46 rows=1000 width=12)
    Sort Key: product_id
-> Gather (cost=251499.14..251604.14 rows=1000 width=12)
    Workers Planned: 2
    -> Partial HashAggregate (cost=250499.14..250504.14 rows=500 width=12)
        Group Key: product_id
    -> Parallel Bitmap Heap Scan on sales (cost=74441.67..242213.86 rows=1657055
width=4)
        Recheck Cond: (product_id < 100)
        -> Bitmap Index Scan on idx_sales (cost=0.00..73447.43 rows=3976933 width=0)
            Index Cond: (product_id < 100)
```

## ■ 13.3 Materialized Views

Materialized Views sind eine wichtige Voraussetzung, um in großen Datenbanken die erwartete Performance erzielen zu können. Aggregierte Tabellen können nicht nur Ergebnisse in wesentlich kürzerer Zeit liefern, sondern schonen auch die Systemressourcen. In Data Warehouse-Datenbanken sind aggregierte Tabellen häufig zu finden, aber nicht darauf beschränkt.

Auch Materialized Views verwenden das Rule System wie normale Views, allerdings werden die Ergebnisdaten in einer Tabelle gespeichert. In Listing 13.10 wird eine große Tabelle mit 50 Millionen Sätzen angelegt, so wie sie in einem Data Warehouse vorkommt.

**Listing 13.10** Eine große Tabelle anlegen

```
(postgres@localhost:5432)[hanser]> CREATE TABLE sales (
> sales_id INT,
> customer_id INT,
> product_id INT,
> sales_date DATE,
> amount NUMERIC(12,2));
(postgres@localhost:5432)[hanser]> INSERT INTO sales
> SELECT n, MOD(n,100) + 1, MOD(n,10) + 1,
> TIMESTAMP '2016-01-01 00:00:00' + RANDOM() * (now() - TIMESTAMP '2016-01-01
00:00:00'),
> RANDOM()::NUMERIC * 1000
> FROM generate_series(1,50000000) x(n);
INSERT 0 50000000
```

Um die Abfragen auf die Verkaufszahlen pro Produkt zu beschleunigen, wird in Listing 13.11 ein Materialized View erstellt.

**Listing 13.11** Ein Materialized View erstellen

```
(postgres@localhost:5432)[hanser]> CREATE MATERIALIZED VIEW sales_sum
> AS SELECT product_id, SUM(amount) FROM sales
> GROUP BY product_id;
SELECT 10
(postgres@localhost:5432)[hanser]> SELECT * FROM sales_sum ORDER BY 1;
 product_id |      sum
-----+-----
          1 | 2498948596.70
          2 | 2500118594.65
          3 | 2499927067.36
          4 | 2499631272.60
          5 | 2500590545.55
          6 | 2499671040.93
          7 | 2501213310.02
          8 | 2500211103.53
          9 | 2500597263.13
         10 | 2500607800.36
(10 rows)
```

In Listing 13.12 finden Sie einen Laufzeitvergleich zwischen der Abfrage auf das Materialized View und der Originaltabelle mit 50 Millionen Sätzen. Während das Scannen der 50 Millionen Sätze 5 Minuten dauert, liefert das Materialized View das Ergebnis in weniger als einer Sekunde. Die Messung ist nach dem Start des Servers erfolgt. Wenn sich die Daten im Cache befinden, kann die Laufzeit variieren.

**Listing 13.12** Lauzeitvergleich Materialized View und Tabelle

```
(postgres@localhost:5432)[hanser]> SELECT product_id, SUM(amount)
> FROM sales
> GROUP BY product_id ORDER BY 1;
product_id |      sum
-----+-----
          1 | 2498948596.70
. . .
Time: 5026,761 ms (00:05,027)
(postgres@localhost:5432)[hanser]> SELECT * FROM sales_sum
> ORDER BY 1;
product_id |      sum
-----+-----
          1 | 2498948596.70
. . .
Time: 154,807 ms
```

Leider gibt es keinen automatischen Refresh-Mechanismus für Materialized Views. Der Refresh-Befehl (Listing 13.13) muss manuell abgesetzt oder in einen Job eingebunden werden. Dabei wird der Inhalt komplett ersetzt. Ein inkrementeller Refresh (Fast Refresh) ist nicht möglich. Mit der Option „CONCURRENTLY“ können SQL-Anweisungen auf das View zugreifen, während der Refresh läuft. Der Refresh läuft länger, allerdings muss die Session mit der Abfrage nicht auf das Beenden der Refreshs warten. Voraussetzung für einen konkurrierenden Refresh ist, dass das Materialized View einen eindeutigen Index besitzt.

**Listing 13.13** Ein Materialized View aktualisieren

```
(postgres@localhost:5432)[hanser]> REFRESH MATERIALIZED VIEW sales_sum;
REFRESH MATERIALIZED VIEW
(postgres@localhost:5432)[hanser]> CREATE UNIQUE INDEX'
> i_sales_sum ON sales_sum(product_id);
CREATE INDEX
(postgres@localhost:5432)[hanser]> \d sales_sum
      Materialized view "public.sales_sum"
  Column | Type | Collation | Nullable | Default
-----+-----+-----+-----+-----
product_id | integer |          |          |
sum       | numeric |          |          |
Indexes:
    "i_sales_sum" UNIQUE, btree (product id)
(postgres@localhost:5432)[hanser]> REFRESH MATERIALIZED VIEW
> CONCURRENTLY sales_sum;
REFRESH MATERIALIZED VIEW
```

## ■ 13.4 BRIN-Indexe

B-Tree-Indexe sind effizient, solange sie eine gewisse Größe nicht überschreiten. Sie wachsen nahezu linear mit der Tabelle und werden für sehr große Tabellen langsam und ineffektiv. Große B-Tree-Indexe müssen häufig von der Disk nachgeladen werden und Index Scans laufen aufgrund der Tiefe der Verzweigungen vergleichsweise lang.

Seit der Version 9.5 gibt es Block Range-Indexe (BRIN). In einem Block werden nicht einzelne Einträge von Spaltenwerten gespeichert, sondern es werden Datenblöcke indiziert. Für jeden Block werden ein Minimal- und ein Maximalwert in Form von Bitmaps gespeichert (siehe Bild 13.3).

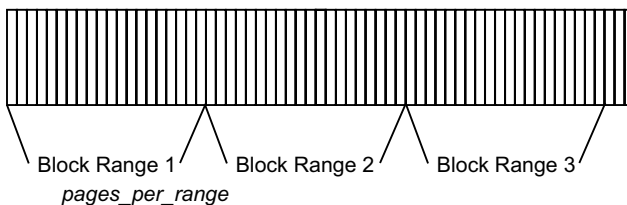
Ein BRIN-Index kann einen signifikanten Performancevorteil gegenüber einem B-Tree-Index erzielen. Der Aufbau des Index macht klar, wann er seine Stärken entfalten kann. Die besten Effekte werden erzielt, wenn die Werte der indizierten Spalte weitgehend in Sortierreihenfolge in den Blöcken gespeichert sind. Ein klassisches Beispiel ist das Auftragseingangsdatum in einer Auftrags-tabelle. Neue Aufträge werden in zeitlicher Reihenfolge aufgenommen und für gespeicherte Sätze ändert sich der Wert nicht.

Das Attribut „pages\_per\_range“ legt fest, wie viele Datenblöcke in einer Block Range zusammengefasst werden. Die Festlegung erfolgt mit der Erstellung des Indexes. Je größer der Wert, desto kleiner wird der Index. Allerdings müssen möglicherweise zu viele Blöcke gescannt werden, wenn der Wert zu groß gewählt wird. Der Standardwert ist 128.

Der entscheidende Vorteil eines BRIN liegt in der Größe. Gerade für sehr große Tabellen ist der Unterschied gewaltig. Eines der Probleme von BRIN-Indexen ist, dass sich bei stark ändernden Tabellen die Qualität und die Effektivität reduzieren. Deshalb kommen BRIN-Indexe insbesondere im Data Warehouse-Umfeld zum Einsatz, wo Daten in einer bestimmten Reihenfolge geladen werden und Tabellen wenigen Veränderungen unterliegen.

BRIN-Indexe werden normal nicht verändert. Seit Version 10 gibt es eine automatische Pflege. Wird der Index mit dem Attribut „autosummarize=true“ angelegt, erfolgt eine Aktualisierung bei jedem manuellen und automatischen VACUUM-Lauf.

### Datenblöcke (Pages)



Block	Minimum	Maximum
1	2018-01-02	2018-01-06
2	2018-01-04	2018-01-12
3	2018-01-10	2018-01-22

**Bild 13.3** Block Range-Index (BRIN)

Für das folgende Beispiel wird in Listing 13.14 eine Tabelle mit mehr als 30 Millionen Sätzen angelegt.

**Listing 13.14** Eine große Tabelle anlegen

```
(postgres@localhost:5432)[hanser]> CREATE TABLE temperature(
> location_id    INTEGER,
> t_time        TIMESTAMP,
> t_celsius     INTEGER);CREATE TABLE temperature(
CREATE TABLE
(postgres@localhost:5432)[hanser]> INSERT INTO temperature
> VALUES (1,generate_series('2017-01-01'::timestamp,'2017-12-31'::timestamp,'1
second'),
> round(random()*100)::int);
INSERT 0 31449601
```

Auswertungen erfolgen nach der Spalte „t\_time“, in der sich Daten vom Typ „TIMESTAMP“ befinden. Zunächst wird ein normaler B-Tree-Index auf die Spalte gelegt (siehe Listing 13.15).

**Listing 13.15** Einen B-Tree-Index erstellen

```
(postgres@localhost:5432)[hanser]> CREATE INDEX i_temperature_btree
> ON temperature (t_time);
(postgres@localhost:5432)[hanser]> \d temperature
```

```
Table "public.temperature"
  Column      |          Type          | Collation | Nullable | Default
-----+-----+-----+-----+-----
 location_id | integer                |           |          |
  t_time     | timestamp without time zone |           |          |
  t_celsius  | integer                |           |          |
Indexes:
  "i_temperature_btree" btree (t_time)
```

Der Ausführungsplan für eine Abfrage zeigt einen parallelen Index Scan auf den B-Tree-Index (siehe Listing 13.16). Die tatsächliche Ausführungszeit beträgt etwas mehr als 600 Millisekunden.

**Listing 13.16** Paralleler Index Scan mit B-Tree-Index

```
(postgres@localhost:5432)[hanser]> EXPLAIN ANALYZE
> SELECT AVG(t_celsius)> FROM temperature
> WHERE t_time > '2017-02-28' AND t_time < '2017-04-01';
QUERY PLAN
-----
Finalize Aggregate (cost=95517.26..95517.27 rows=1 width=32) (actual
time=609.800..609.800 rows=1 loops=1)
-> Gather (cost=95517.05..95517.26 rows=2 width=32) (actual time=309.727..609.793
rows=3 loops=1)
    Workers Planned: 2
    Workers Launched: 2
-> Partial Aggregate (cost=94517.05..94517.06 rows=1 width=32) (actual
time=285.748..285.748 rows=1 loops=3)
-> Parallel Index Scan using i_temperature_btree on temperature
(cost=0.56..91492.08 rows=1209986 width=4) (actual time=0.376..339.046 rows=92160
loops=3)
    Index Cond: ((t_time > '2017-02-28 00:00:00'::timestamp without time zone) AND
```

```
(t_time < '2017-04-01 00:00:00'::timestamp without time zone))
Planning time: 0.504 ms
Execution time: 858.128 ms
(postgres@localhost:5432)[hanser]> SELECT AVG(t_celsius)
> FROM temperature
> WHERE t_time > '2017-02-28' AND t_time < '2017-04-01';
      avg
-----
 49.9919256336536580
Time: 633,886 ms
```

Für den zweiten Test wird ein BRIN-Index mit einer Block Range von „128“ angelegt (siehe Listing 13.17).

**Listing 13.17** Einen Block Range-Index anlegen

```
(postgres@localhost:5432)[hanser]> CREATE INDEX i_temperature
> ON temperature USING BRIN (t_time)
> WITH (pages_per_range = 128, autosummarize=true);
CREATE INDEX
(postgres@localhost:5432)[hanser]> ANALYZE temperature;
ANALYZE
(postgres@localhost:5432)[hanser]> \d temperature
Table "public.temperature"
  Column |          Type          | Collation | Nullable | Default
-----+-----+-----+-----+-----
 location_id | integer                |           |          |
 t_time      | timestamp without time zone |           |          |
 t_celsius   | integer                |           |          |
Indexes:
    "i_temperature" brin (t_time) WITH (pages_per_range='128', autosummarize='true')
```

Der Ausführungsplan in Listing 13.18 führt einen „Bitmap Index Scan“ durch, ebenfalls mit einem Parallelitätsgrad von 2. Die Kosten, verglichen mit dem Index Scan des B-Tree-Indexes in Listing 13.16, sind signifikant geringer. Die Ausführungszeit beträgt in etwa 268 Millisekunden.

**Listing 13.18** Ausführungsplan mit Block Range-Index

```
(postgres@localhost:5432)[hanser]> EXPLAIN ANALYZE
> SELECT AVG(t_celsius)
> FROM temperature
> WHERE t_time > '2017-02-28' AND t_time < '2017-04-01';
      QUERY PLAN
-----
Finalize Aggregate (cost=223316.45..223316.46 rows=1 width=32) (actual
time=245.086..245.086 rows=1 loops=1)
  -> Gather (cost=223316.23..223316.44 rows=2 width=32) (actual
time=245.059..245.079 rows=3 loops=1)
    Workers Planned: 2
    Workers Launched: 2
      -> Partial Aggregate (cost=222316.23..222316.24 rows=1 width=32) (actual
time=218.354..218.354 rows=1 loops=3)
        -> Parallel Bitmap Heap Scan on temperature (cost=763.63..219291.26
rows=1209986 width=4) (actual time=1.413..172.239 rows=921600 loops=3)
          Recheck Cond: ((t_time > '2017-02-28 00:00:00'::timestamp without time zone)
AND (t_time < '2017-04-01 00:00:00'::timestamp without time zone))
```

```

Rows Removed by Index Recheck: 2816
Heap Blocks: lossy=6897
-> Bitmap Index Scan on i_temperature (cost=0.00..37.64 rows=2913861
width=0) (actual time=1.737..1.737 rows=176640 loops=1)
    Index Cond: ((t_time > '2017-02-28 00:00:00'::timestamp without time zone)
AND (t_time < '2017-04-01 00:00:00'::timestamp without time zone))
    Planning time: 0.183 ms
    Execution time: 279.457 ms
(postgres@localhost:5432)[hanser]> SELECT AVG(t_celsius)
> FROM temperature
> WHERE t_time > '2017-02-28' AND t_time < '2017-04-01';
    avg
-----
 49.9919256336536580
Time: 268,739 ms

```

Die Ausführungszeit der SQL-Anweisung wurde etwas mehr als halbiert. Ein besserer Wert ist in diesem Beispiel nicht erzielbar, da die Zeit für die Aggregation einen Großteil der Gesamtausführungszeit ausmacht. Die Laufzeiten für die Index Scans unterscheiden sich deutlich:

- B-Tree-Index Scan: actual time=0.376..239.046
- BRIN-Index Scan: actual time=1.737..1.737

Das Beispiel ist natürlich ein Idealfall für den BRIN-Index. Die Daten wurden frisch geladen und zwar in Sortierung des Timestamps. Die Verteilung ist damit fast optimal. Beindruckend ist auch der Unterschied in der Größe der Indexe.

**Listing 13.19** Größenvergleich B-Tree- und BRIN-Index

schema_name	index_name	index_ratio	index_size	table_size
public	i_temperature_btree	0.43	674 MB	1565 MB
public	i_temperature	0	72 kB	1565 MB

# Index

## Symbole

24x7-Betrieb 405  
\$PGDATA 47, 50  
\$PGDATA-Verzeichnis 20  
%ROWTYPE 299  
%TYPE 299

## A

Abbruchbedingung 298  
Active State Perl 10, 374  
AddGeometryColumn 256  
Ad-hoc-Abfragen 158  
Administrierbarkeit 404  
Aggregation, parallele 236  
allow\_system\_table\_mods 57  
ALTER EXTENSION 288  
ALTER ROLE 142  
ALTER TABLESPACE 208  
ANALYZE 208, 224, 404f.  
ANALYZE-Befehl 71, 210  
ANSI-Syntax 421  
Antwortzeit 351  
Apache 264  
Applikationsdesign 261, 295  
Apply-Prozess 159, 172  
Architektur 25  
archive\_cleanup\_command 156  
archive\_command 53, 66  
archive\_mode 52  
Archive-Modus 36  
archive\_timeout 53, 67  
Archivierungskommando 67  
Array 326  
Audit-Daten 143  
Audit-Satz 144

Auditing 103, 309, 406  
Ausführungsplan 182, 205, 216, 224, 226f.,  
234  
– Effektivität des 211  
Ausführungspläne 207  
Ausführungsteil 296  
Auslastungsgrad der Disks 29  
authentication\_timeout 49  
Authentifizierung 49, 53, 406  
– von Clients 50  
Authentifizierungsmethode 50f.  
Authentifizierungsprozess 125, 139  
Authentifizierungszertifikate 141  
autocommit 66, 316, 328, 337, 375  
autovacuum 37, 69, 208  
autovacuum launcher 70  
autovacuum launcher process 37  
autovacuum\_max\_workers 37, 70  
autovacuum\_naptime 37, 70

## B

Backend-Prozess 28, 188, 206  
Background Writer 200  
Backup des Clusters 18  
Backup-Datei 19  
Backup-Modus 107  
Backup-Retention 467  
Base Backup 106  
BEGIN TRANSACTION 304  
Begrenzer für Konstanten 296  
Beispielschema 83  
Betrieb, professioneller 401  
Betriebssystem-Einstellungen 45  
Betriebstauglichkeit 401  
Betriebsumgebung 443



- Bibliotheken, dynamisch ladbare 279
  - Binärdaten 395
  - Binär-Kompatibilität 20
  - Binär-Modus 113
  - Binary-Installation 18
  - Binary Packages 5
  - Bind Peeking 227
  - Bindevariable 227, 329, 381
  - Binding by Reference 382
  - Bitmap Index Scan 211, 226, 242
  - Bitmap-Operationen 225
  - Blöcke, populäre 31
  - Blocknummer 42
  - Block Range 242
  - Block Range Index 104, 240
  - Bootstrap XID 38
  - BRIN 240
  - BRIN-Index 230, 242
  - Brute-Force-Attacke 137f.
  - BSI-Richtlinien 117
  - B-Tree-Indexe 211
  - Bucket-Nummer 31
  - Buffer Cache 213
  - Buffer Descriptor 30
  - Buffer Header 33
  - Buffer Pool 30
  - Bug Fixes 17
  - Build-Prozess 8
  - Build Table 212
  - Built-in-Funktion 330
  - Bulk Load 114
  - Bundle-Installation 246
  - Business-Funktionalität 182, 295, 309, 421
  - Byte-Array 335
- C**
- Callable Statement 332
  - Callback-Ereignisse 317
  - Callback-Verfahren 353
  - Cascading Standby 157
  - C-Compiler 314
  - C-Funktion 270, 279, 284
  - Checkpoint 26, 36, 107, 165, 192, 200
  - Checkpoint-Prozess 36
  - Checkpoint-Rate 189
  - checkpoint\_timeout 36
  - Checkpoint-Verhalten 201
  - client\_encoding 58
  - client\_min\_messages 56
  - Client/Server-Prinzip 295
  - Clock Sweep-Algorithmus 29, 33
  - Cloud 463
  - Cloud-Dienst 467
  - Cloud Service Provider 465
  - Code-Blöcke, anonyme 277
  - COMMIT 64, 304, 380
  - COMMIT-Anweisung 36
  - Compiler-Option 281
  - Computer, entfernte 12
  - configure-Befehl 18
  - Contribution Module 286
  - COPY-Befehl 73, 94, 114, 389, 435
  - Cost Based Optimizer 207
  - CPAN-Modul 374
  - C, Programmiersprache 311
  - CPU Clockspeed 229
  - CPU-Kosten 199, 207
  - Crash
    - des PostgreSQL-Servers 108
    - des Primärservers 165
  - Crash Recovery 67
  - created 61
  - CREATE DATABASE 60, 113
  - CREATE EXTENSION 277, 290
  - CREATE FUNCTION 285, 307
  - CREATE INDEX 91
  - CREATE OPERATOR 286
  - CREATE PUBLICATION 172
  - CREATE ROLE 94, 119
  - CREATE RULE 180
  - CREATE SEQUENCE 94
  - CREATE TABLE 62, 86, 271, 419
  - CREATE TABLESPACE 62, 81
  - CREATE TRIGGER 307
  - CREATE USER 94, 119
  - CREATE VIEW 180
  - Cross Column Statistics 92
  - CSV-Datei 389, 435, 440
  - CSV-Format 414
  - ctid 423
  - CTID-Schlüssel 212
  - CTID-Spalte 212
  - C- und C++-Programme 264
  - Cursor 301
    - expliziter 302
    - impliziter 302
  - Cursor-Variable 426
  - custom\_variable\_classes 57

**D**

Database as a Service 463  
 Database Independent Interface 373  
 Data Warehouse 353  
 Data Warehouse-Datenbanken 238  
 Data Warehouse-Systeme 88  
 Data Warehouse-Umfeld 229  
 Dateien, temporäre 82  
 Dateisystem-Cache 188  
 Datenbanken, große 229  
 Datenbank-Auditing 142  
 Datenbank-Cloud 463  
 Datenbankdateien, Speichern der 9  
 Datenbankkatalog 19, 206, 276  
 Datenbanklink 443, 448  
 Datenbankoperationen 270  
 Datenbankserver starten 9  
 Datenbanksicherheit 117  
 Datenbanksysteme, verteilte 89  
 Datenbankunabhängigkeit 321  
 Datenbank-Vendor 408  
 Datenbestand, Wachstum des 229  
 Datenblöcke 26  
 Datentypen 432  
 – geografische 250, 253  
 – geometrische 245  
 Datenverluste 153  
 Datenvolumen, wachsendes 262  
 Datumsformat 57  
 DBD 436  
 DBD-Modul 373  
 DBI-CSV-Modul 389  
 DBI-Modul 376  
 DBI-Schnittstelle 276  
 DBI-Trace 384  
 db\_user\_namespace 50  
 DDL-Anweisungen 142, 412  
 DDL-Kommandos 170  
 DDL-Operation 447  
 Deadlock 65  
 Dead Row-Versionen 54  
 Debug-Informationen 57  
 Debugger 358  
 default\_tablespace 53  
 Default-Tablespace 62  
 Definer-Rechte 141  
 Defragmentierung 37, 406  
 Deployment 263  
 Disaster-Recovery 89, 403  
 Disaster-Recovery-Lösung 151

Disaster-Recovery-Test 167  
 DETACH-Befehl 232  
 Diebstahl des verschlüsselten Passworts 138  
 Distribution 18  
 DLL-Datei 279, 281  
 DML-Anweisung 180, 275, 303, 380  
 DML-Operationen 127  
 Driver Manager 322  
 Driving Table 213, 215, 236 f.  
 DROP DATABASE 61, 112  
 DROP TABLESPACE 63  
 Dump-Datei 412  
 Dynamic Shared Libraries 279

**E**

ECPG 311  
 ECPG-Programm 315  
 Eigentümer  
 – der Software 8  
 – eines Objekts 127  
 Einsatz, betrieblicher 401  
 Einzelsatz-Verarbeitung 262  
 Embedded SQL 264, 311  
 Energieverbrauch 229  
 EnterpriseDB 6  
 Entwicklungsframework 321  
 Entwicklungsphase 262  
 Entwicklungsumgebung 265, 312, 341  
 Ergebnismenge 215, 221  
 Erweiterbarkeit 264, 402  
 – von PostgreSQL 269  
 Erweiterungen 266  
 Event-Trigger 263  
 Exception 383  
 Exception-Block 306  
 EXECUTE-Recht 128  
 EXPLAIN-Befehl 216, 234  
 EXPLAIN SUMMARY 98  
 Extension-Paket 289  
 Extensions 248, 287, 467  
 – publizieren 290

**F**

Failover 105, 163  
 – automatisches 164  
 Features, neue 1, 85, 102, 230  
 Fehleranalyse 384  
 Fehlerbehandlung 304, 306, 317, 384  
 Fehler-Level 306

Fehler-Stack 262  
 FETCH-Befehl 302  
 Foreign Key Constraints 432  
 fork-Kommando 28  
 FOR LOOP 296  
 FOR-Schleife 302  
 Freelist 33  
 Free Space Map 41  
 Fremdschlüssel 233  
 Frequency Histogram 210  
 Frozen XID 38  
 fsync 52  
 Full Table Scan 147, 199, 223f., 227

## G

Gateway-Datenbank 450  
 Gebietschema 57  
 GENERATED AS IDENTITY 93  
 Generic WAL Facility 103  
 Genetic Query Optimizer 207  
 Geographic Information System 245  
 GEOS Geometry Library 249  
 get\_raw\_page 43  
 GIN-Indexe 104  
 GIS-Applikationen 402  
 GiST-Index 254  
 GNU-Compiler 314  
 Golden Gate 443  
 Green IT 229

## H

Hacker-Attacken abwehren 137  
 Hard-Parsing 206  
 Hardwarekomponenten 229  
 Hardwareressourcen 47  
 hashcat 139  
 Hash-Funktion 212  
 Hash Join 212, 221  
 Hash-Kollision 30  
 Hash-Tabelle 30, 212, 221  
 Hashwert 30, 213, 379  
 Hauptprozess 27  
 heap\_page\_items 43  
 Height-Balanced Histogram 210  
 Heterogeneous Gateway 443, 448  
 Hintergrundprozesse 27  
 HIPAA 117  
 Hit Ratio 146, 195  
 Hochverfügbarkeit 89, 405

Hochverfügbarkeitskonzept 161  
 Hostvariable 315  
 hot\_standby 54  
 Hot Standby 54, 151, 158, 403  
 HTML-Code 362  
 HTML-Format 364  
 huge\_pages 47, 49  
 Huge Pages 46, 188

## I

IDE 281, 323  
 Incidents 142  
 Indexe, nicht verwendete 76  
 Index-Only Scan 211  
 – paralleler 237  
 Index Scan 200, 208, 211, 236  
 Infrastructure as a Service 463  
 initdb 40, 48, 57, 100  
 Installation  
 – der Binaries 8  
 – aus dem Quellcode 8, 249  
 Installationsaufwand 466  
 Installationsverzeichnis 8, 24  
 I/O-Aktivitäten 193  
 I/O-Kosten 207  
 I/O-Operationen 26, 29, 466  
 – physikalische 188  
 I/O-Performance, Optimierung der 61  
 iostat 193  
 I/O-Statistiken 199  
 I/O-Subsystem 29, 199, 229  
 IP-Adresse  
 – des Clients 13  
 – virtuelle 165  
 Isolation Level 63

## J

Java 264, 321  
 Java-Applikation 321  
 Java-Programm 332  
 Java SE Development Kit 324  
 JDBC 264, 407  
 JDBC-Funktion 322  
 JDBC-Schnittstelle 321  
 JDBC-Tracing 339  
 JDBC-Treiber 175  
 JDBC-Verbindung 452  
 Join-Methode 212  
 Join-Operation 215

**K**

Klonen 466  
Kommando, dynamisches 305  
Kommandozeile 324  
Kompatibilität 423  
Kompatibilitätsgrad 443  
Kompilierung des Quellcodes 10  
Konfigurationsdateien 12  
Konflikte 174  
Konsistenz der Transaktionen 169  
Kontrolldatei 79, 289  
Koordinatensystem, kartesisches 250  
Korrektheit, syntaktische 311  
Kostenrechnung 207

**L**

Laden von Daten 434  
Large Objects 90, 102, 170, 334, 371, 390, 395  
Large Object-API 337  
Large Object Facility 395  
Lastverteilung 195  
Laufzeitstabilität 311  
LD\_LIBRARY\_PATH 249, 342  
LDAP-Server 126  
Left Outer Join 215  
Lesekonsistenz 33  
libpg 264, 369  
libpq 91, 341, 348, 396  
libpq-Library 311  
List Partitioning 86, 231  
listen\_address 49  
LOB 371  
- verwaistes 398  
LOB-Eintrag 397  
Locale 57  
lo\_import 354  
log\_connection 138  
log\_destination 55, 75  
log\_directory 56  
log\_filename 56  
log\_min\_messages 56  
log\_rotation\_age 56  
log\_rotation\_size 56  
Log Sequence Number 34, 41  
log\_statement 57, 142  
Logdateien 19, 21, 56, 143  
logging\_collector 55  
Logical Change Records 450

Logical Decoding 89, 103, 151, 174, 406  
Logical Replication Protocol 169  
Logical Replication Slot 89, 169  
Logical Sequence Number 161  
Logische Replikation 2, 85  
LOOP-Anweisung 298

**M**

Magic Block 279  
maintenance\_work\_mem 29, 49, 189  
Major Release-Upgrade 17  
Major Release-Wechsel 17, 85  
Major Version 279  
Major Version-Upgrade 19  
Make Utility 8  
Master-Detail-Beziehung 170  
Master-Prozess 236  
Master-Seite 171  
Master-Server 53 f.  
Master-Tabelle 86  
Materialized View 102, 230, 238  
max\_connections 29, 49, 80, 188  
max\_logical\_replication\_workers 55  
max\_replication\_slots 54, 170  
max\_standby\_archive\_delay 54, 159  
max\_standby\_streaming\_delay 54, 159  
max\_sync\_workers\_per\_subscription 55  
max\_wal\_senders 54, 170  
max\_wal\_size 36  
md5 51  
MD5 126, 139  
Memory-Kapazitäten 188  
Memory-Strukturen 25, 29  
Memory-Verbrauch 188  
Memory-Verwaltung 46  
Merge Join 221  
Metadaten 271, 324, 377  
Microsoft Excel 77  
Migration 1, 21, 409, 418, 440, 443  
- automatische 427  
- großer Tabellen 440  
- logische 18  
Migrationsaufwand 423  
Migrationspfad 22  
Minor Release-Wechsel 17  
Minor Version-Upgrade 19  
Mischbetrieb 443, 448  
Monitoring 144, 404  
Multi-Master-Replikation 402  
Multi-Master-Systeme 89

Multiversion Concurrency Control 37, 63  
 MVCC 37  
 MVCC-Modell 37  
 MySQL 264, 409  
 MySQL-Datenbank 410  
 MySQL Dump 411

## N

Nagios 149, 404  
 Native Partitioning 21  
 – Migration nach 21  
 Native Partitioning-Syntax 22  
 Native Partition Table 85  
 Native Table Partitioning 2, 230  
 Nested Loop Join 212f.  
 NetBeans 266, 321, 343, 361  
 Network Spoofing 140  
 Netzwerk-Interface 13  
 Netzwerk-Sniffer 139  
 Neuschreiben der Abfrage 179  
 Non-Functional Test 262  
 Normale XID 38  
 Null-Wert 421  
 Nummerierung der Versionen 17

## O

Objekt-ID 41, 354  
 Objekt-Identifizier 191  
 Observer 165  
 Observer-Prozess 163  
 ODBC-Schnittstelle 407, 429, 443  
 Offline-Attacke 138  
 Offline-Sicherung 105  
 – Vorteile 111  
 OLTP-Applikationen 205  
 Online-Backup 154, 405  
 – des Primärservers 155  
 Online-Sicherung 105, 402  
 OpenJUMP 251, 260  
 Open Source-Datenbanken 187  
 Open Source-Software 1  
 OpenSSL 132, 407  
 Operator, benutzerdefinierter 284  
 Optimierung der Schreibvorgänge 26  
 Optimierungsmöglichkeiten 187  
 Optimizer 179, 220  
 Optimizer-Statistiken 86, 212  
 Ora2Pg 415, 427, 436, 438  
 Oracle 417, 419, 421

Oracle-Applikationen 422  
 Oracle-Datenbank 427  
 Oracle-Spaltentypen 419  
 Oracle-Syntax 426  
 orafce 424  
 OSGeo Foundation 245  
 Outer Join 215  
 Outer Table 213  
 Out-of-place-Upgrade 18, 21  
 Out-of-the-box-Funktionalität 404  
 Out-of-the-box-Sicherheit 137  
 Out-of-the-box-Tuning 188  
 Output-Plugin 175

## P

PaaS 463f.  
 Page Header 41f.  
 pageinspect 42  
 Paketinstallation 5, 248  
 – Linux 5  
 – Windows 6  
 Parallel Bitmap Heap Scan 235  
 Parallel Bitmaps Scans 88  
 Parallel B-Tree-Index Scans 88  
 Parallel Index-Only Scan 235  
 Parallel Index Scan 235  
 Parallel Merge Joins 88  
 Parallel Query 2, 85, 104, 236  
 Parallel Sequential Scan 235  
 Parallele Aggregation 235  
 Parallele Joins 235f.  
 Parallele Scans 235  
 Paralleles SQL 88, 230  
 Parallelisierung von Prozessen 229  
 Parallelitätsgrad, Erhöhung 229  
 Parameter, dynamischer 48  
 Parsing 206, 376  
 PARTITION BY 86  
 PARTITION BY-Klausel 231  
 PARTITION OF 86  
 Partitionen 23  
 Partitionierung 230  
 Partitioning-Technologie 21  
 Partitionsschlüssel 232f.  
 passwordcheck 136  
 password\_encryption 50, 137  
 Passwort  
 – schwaches 122  
 – unverschlüsseltes 141  
 Passwortdatei 72, 156

- Passwortkomplexität 136
- Passwortregeln 406
- PDF-Datei 371
- PDO-API 370
- Perfmon 193
- Performance 187
  - Steigerung der 188
- Performance-Analyse 194, 201
- Performancefaktor 309
- Performancegewinn 26
- Performanceprobleme 145, 193
- Performance-Tuning 187, 190
- Performancevorteil 221
- Performance-Werkzeuge 223
- Perl 270, 276, 373
- Perl DBI 264
- Perl-Interpreter 10
- Perl Package Manager 374, 437
- pgAdmin 13
- pgAdmin 4 131, 148, 218, 265, 404
- pg\_authid 51, 121
- pg\_availability\_extensions 287
- pg\_basebackup 99, 106
- pg\_buffercache 190
- pg\_bulkload 440
- pg\_cancel\_backend() 73
- pg\_catalog 53
- pg\_class 130
- pg\_control 36
- pg\_controldata 34, 79
- pg\_ctl 99
- pg\_current\_logfile 96
- pg\_dump 99, 111
- pg\_dumpall 17, 115
- pg\_freespacemap 44
- pg\_hba.conf 406
- pg\_hba.conf 13, 18, 50, 125, 127, 134, 154, 172
- pg\_ident.conf 51
- pg\_largeobject 335, 395
- pg\_largeobject\_metadata 395
- pg\_largobject 353
- pg\_locks 147, 198
- pg\_monitor 92
- pg\_read\_all\_settings 92
- pg\_read\_all\_stats 92
- pg\_regress 23
- pg\_restore 115
- pg\_roles 121
- pg\_settings 47
- pg\_shadow 121
- pg\_start\_backup 107
- pg\_stat\_activity 73, 145, 196, 198
- pg\_stat\_database 146
- pg\_stat\_ssl 134
- pg\_stat\_statements 57, 201
- pg\_stat\_subscription 172
- pg\_switch\_wal 36, 75
- pg\_tables 271
- pg\_terminate\_backend(). 73
- pg\_trace 369
- pg\_upgrade 17, 19
- pg\_user 121
- pg\_walfile\_name 109
- pgoutput 169
- PGPASSFILE 72
- PGresult 349
- pgsnmp 404
- pgstatspack 193, 201
- pgtune 189
- PGXN-Netzwerk 290, 292
- PGXN-Utilities 290
- PHP 264, 359, 362
- PHP-Code 364
- PHP-Entwicklungssystem 360
- Pipe 19
- PITR-Sicherung, Vorteile 110
- Platform as a Service 463
- PL/Perl 276
- PL/pgSQL 263, 270, 275, 295, 425, 433
- PL/pgSQL-Block 296, 303
- PL/pgSQL-Code 142
- PL/pgSQL-Konsole 404
- PL/pgSQL-Variablen 299
- PL/SQL 417, 424
- PL/SQL-Code 418
- Plug and Play 287
- Plug-ins 264
- Point-in-time-Recovery 66, 105, 467
- port 49
- post\_auth\_delay 57
- PostGIS 245, 467
- PostGIS-Datenbank 246
- postgres.conf 13, 47
- PostgreSQL-Cluster 25
- PostgreSQL Extension Network 266, 286
- PostgreSQL-JDBC-Treiber 91, 322, 427
- PostgreSQL-ODBC-Treiber 78
- PostgreSQL-Optimizer 207
- PostgreSQL-Quellcode 10

PostgreSQL-Query-Planer 71  
 PostgreSQL-Server 25  
   – gefälschter 140  
 Postmaster-Prozess 27f.  
 PQerrorMessage 342  
 PQexec 346, 353  
 PQexecPrepared 356  
 PQprepare 346, 356  
 PQresultStatus 346  
 PQtrace 356  
 Präprozessor 311, 313  
 Prepare-Anweisung 377  
 Prepare- und Execute Anweisungen 236  
 Prepared Statement 227, 316  
 Primärschlüssel 87f., 169, 233  
 Primärserver 55  
 Private Cloud 463f.  
 Private Key 132f.  
 Privilegien von Objekten 118  
 Probe-Phase 212  
 Probe Table 212  
 Process Explorer 193  
 Programmierinterface, natives 341  
 Programmiersprache, prozedurale 295  
 Programmiersprache C 311  
 Proj4 Reprojection Library 249  
 Projektion 253  
 Provisioning 464  
   – automatisches 465  
 Proximity Analysis 258  
 Prozesse  
   – parallele 235  
   – wartende 30  
 Prozess-ID 196  
 Prozessabläufe 25  
 psql 12, 64, 74, 302  
 psqlrc 74  
 Public Cloud-Lösung 463  
 Public-Netzwerk 164  
 Publisher 171  
 Publisher- und Subscriber-Prinzip 90  
 Publisher-Datenbank 169

## Q

Quellcode-Distribution 264  
 Quellprogramme 8  
 Quellsystem 427  
 Query Optimizer 88, 205, 236  
 Query-Planer 179, 188, 233  
 Query Rewrite 179, 206

Query Tool 218, 265  
 Query Tree 179

## R

RAISE 306  
 RAISE-Befehl 308  
 Range Partitioning 22, 86, 231  
 Read Committed 63  
 Read-Only-Datenbanken 89  
 Read-Only-Modus 158  
 Rechtesystem 127  
 Recovery aus dem Archiv 163  
 Recovery-Algorithmus 36  
 Recovery-Modus 153, 158  
 Recovery-Zeit 403  
 Recovery-Zeitpunkt 109  
 recovery.conf 108, 156, 168  
 recovery.done 109  
 Ref-Cursor 302, 331  
 Referenzsystem 252  
 Refresh-Befehl 239  
 Regelsystem 179, 263  
 regexp\_match 97  
 Regressionstest 23, 440  
 Reiseportale 245  
 Release-Wechsel 264  
   – Major 1  
 Release-Zyklus 5  
 Replication API 178  
 Replication Stream 177  
 Replikation 54, 151, 450, 458  
   – aktivieren 154  
   – logische 88, 90, 168  
   – physikalische 89  
   – synchrone 54, 151, 160  
   – Überwachung der 161  
 Replikationsverbindungen 54f.  
 Reporting 158  
 Resource Manager 205  
 Ressourcenverbrauch 47, 194, 215, 262  
 Resultset 219, 222, 325, 332, 349, 382  
 RETURN 300  
 RETURNING 274  
 RETURNING-Klausel 271  
 RETURN NEXT 300  
 REVOKE-Befehl 131  
 Rewritten Query Tree 206, 227  
 Right Outer Join 215  
 Rollback 21, 64, 304, 380  
 Rollback-Operationen 37, 203

Rollendes Verfahren 167  
 Rollenkonzept 118  
 Rollentausch 151, 155, 163, 168  
 Root Cause-Analyse 262  
 Rotation von Logdateien 76  
 Roundtrip-Zeiten 329  
 Routenplaner 245  
 Row Header 41  
 Row Type 298  
 Rückgabewert 300  
 Rückwärtskompatibilität 17

## S

Satzsperrung 148  
 SAVEPOINT 64  
 Scan, sequenzieller 219  
 Scan-Operationen 235  
 Schema Browser 265  
 Schnittstellen 264, 407  
 Schreiboperationen 29  
 Schwellenwerte 149, 209  
 scram-sha-256 51  
 SCRAM-SHA-256 86, 126  
 search\_path 53  
 Segmentgröße 34  
 SELECT-Anweisung 64, 180  
 SELECT INTO 299  
 SELECT-Privilegien 120  
 Selektivität, hohe 220  
 Semantik-Prüfungen 206  
 Semaphoren 45  
 Sequential Scan 211  
 Sequenz 433  
 Sequenzdaten 170  
 Serializable 63, 66  
 Server, virtueller 45  
 Server-Log 56  
 Serverparameter 102  
 Server Programming Interface 264, 283  
 Serververbindung 13  
 Service-Level 464  
 Service Level Agreements 105  
 Sessions, blockierte 198  
 Session killen 73  
 SET ROLE 122  
 SET TRANSACTION 64  
 SETOF-Typ 331  
 Shared Buffer 26, 29, 190  
 Shared Buffer Cache 36  
 Shared Buffer Contention 31  
 Shared Buffer Pool 46  
 shared\_buffers 46, 49, 188  
 Shared Library 280  
 Shared Memory 26, 188  
 – Größe des 188  
 Shared Memory-Segmente 45  
 Shared-Modus 31  
 Shutdown-Prozess 167  
 Sicherheit 117  
 Sicherheitslücke 317  
 Sicherheitsrichtlinie 463  
 Single Block-Operationen 29  
 Skalierbarkeit 187, 262, 406  
 – der Applikation 261  
 Snapshot, initiales 171  
 SNMP-Schnittstelle 150, 404  
 Soft-Parsing 206  
 Solid State Disks 229  
 Sort Merge Join 212, 215  
 Sortierphase 215  
 Sortiervorgang 215  
 SOX 117  
 SOX Compliance Model 117  
 Spalte, geometrische 257  
 Spatial-Datenbank 248  
 Spatial Reference System 250  
 Spatial System 245  
 Sperren 198  
 Sperren auf Sätze 147  
 SPI 283  
 Sprachen  
 – prozedurale 263, 307  
 SQL, dynamisches 141, 305, 316  
 SQL-Abfragen 82  
 – asynchrone 390  
 SQL-Anweisungen  
 – lang laufende 82, 196  
 – optimieren 205  
 SQL Batches 329  
 SQL-Befehl 311  
 SQL Common Area 318  
 SQL Descriptor Area 315  
 SQL Developer 427ff.  
 SQL Dump 111, 402  
 – komprimierte 115  
 – Vorteile 115  
 SQL Engine 206, 269, 279, 311  
 SQL-Erweiterungen 402  
 SQLException 322  
 SQL-Funktionen 271, 273  
 – Erweiterung von 263



SQL Injection 141, 317  
 SQL-Optimierung 205, 215  
 SQL-Optimizer 205  
 SQL-Pläne 265  
 SQL-Syntax 421  
 SQLDA 315  
 SSH-Server 135  
 SSH-Tunnel 132, 135  
 ssl 50  
 sslinfo 134  
 SSL-Option 133  
 SSL-Verbindung 50, 126  
 SSL-Verschlüsselung 132  
 Standard-Blockgröße 33  
 Standardeinstellung 328  
 Standardparameter 187  
 Standby-Datenbanken 33, 105  
   – physikalische 152  
 standby-mode 156  
 Standby-Server 53f., 152  
   – Starten 153  
 Startup-Kosten 216  
 statement\_timeout 53  
 Statistics Collector 193, 197  
 Statistiken 21, 203, 208  
   – Sammeln 194  
 Statistiken auf Spaltenebene 209  
 Statistiken auf Tabellenebene 209  
 Statistiken des Betriebssystems 193  
 Statistiksammlung 209  
 Stored Functions 409  
 Streaming Replication 54, 151, 153, 162,  
   403, 405  
 Streaming Replication Protocol 103  
 Subnetz des Servers 126  
 Subqueries 219  
 Subscriber 53  
 Subscriber-Cluster 169  
 Summentabelle 309  
 Superuser 7, 52  
 Superuser-Privileg 119  
 Supportanforderungen 25  
 Switchover 163, 167  
 Switchover-Prozess 168  
 synchronous\_commit 52, 160  
 synchronous\_standby\_names 54, 160  
 Syntax-Prüfungen 206  
 sysctl.conf 46  
 Systemabsturz 165  
 Systemdatum 421  
 System-DSN 445

Systemkatalog 61, 264  
 Systemressourcen 45, 49, 196, 223  
 Systemtabellen 19, 57  
 Systemzustand, konsistenter 36

## T

Tabellen, aggregierte 238  
 Tabellen-Design 205  
 Tabellenpartitionen 61  
 Tabellen-Trigger 143  
 Table Scan 208  
 Tablespace 61  
 Taktfrequenz 229  
 target\_session\_attrs 91  
 TCP/IP-Port 49  
 TCP/IP-Sockets 126  
 temp\_buffers 29, 49, 82  
 temp\_file\_limit 49  
 temp\_tablespace 53, 82  
 Thin JDBC-Treiber 321  
 TID Scan 212  
 Trace-Datei 262, 358  
 Tracing 338, 356  
 track\_commit\_timestamp 54  
 Transaction Engine 409  
 Transaktion 328, 425  
 Transaktions-ID 38, 158  
 Transaktionskontrolle 271  
 Transaktionslog 29, 33, 457  
 Transaktionssteuerung 304  
 Transaktionsverhalten 316  
 Transaktionsvolumen 154, 189  
 Trigger 307  
 Trigger-Funktion 308  
 TRUNCATE-Befehl 90, 170  
 Tuples 41  
 Tuple Identifier 42  
 Typen, mehrdimensionale 298

## U

Überwachung 117, 142, 149  
 Überwachungsstruktur 404  
 Umgebungsanalyse 258  
 Umgebungsvariablen 12, 345  
 Umstellungsaufwand 418  
 UNDO-Strukturen 37  
 Upgrade 1, 17  
 Usage Count 31 f.

**V**

VACUUM 69, 200, 208, 240, 404 f.  
– automatisches 70  
vacuum\_defer\_cleanup\_age 54  
VACUUM FULL 37, 69  
VACUUM-Lauf 44  
VACUUM-Operationen 37  
VACUUM-Prozess 37, 39, 41  
VACUUM Worker-Prozess 37, 70  
Verbindungen  
– lokale 50  
– parallele 389  
Verbindungsanfrage 28, 125, 141  
Verbindungsfehler 342  
Verfügbarkeit 405  
– der Datenbanken 105  
Verschlüsselung 51, 135  
Version 10 1  
Versionierung 286  
– von Sätzen 38  
Versionsnummer 17  
Visibility Map 41  
Visual C++-Compiler 10  
Visual Studio 10  
vmstat 193

**W**

Wait Event 197  
WAL, Umbenennung nach 90  
WAL-Archiv 54  
WAL-Archivierung 66, 106, 402  
WAL-Block 35  
WAL Buffer 26, 33  
WAL-Dateiname 34  
WAL-Sätze, Auslesen der 177  
WAL-Segment 68, 106  
WAL-Switche 189  
WAL-Übertragung 89  
wal\_buffers 52  
wal\_keep\_segments 54  
wal\_level 52, 67  
wal\_receiver\_status\_interval 55  
wal\_receiver\_timeout 55  
wal\_retrieve\_retry\_interval 55  
wal\_sender\_timeout 54  
Wartungsarbeiten 167  
Wartungsaufgaben 68, 187, 405

Web-Entwicklung 264  
WGS 84 252  
WHERE-Bedingung 219  
WHERE CURRENT OF 304  
WHERE-Klausel 305  
Wiederherstellbarkeit 36, 52  
– des Clusters 37  
Wiederherstellung 105, 402  
– des Clusters 29  
Wiederherstellungsmethode 105  
Wiederherstellungszeitpunkt 109  
Windows-Betriebssystem 9  
Windows-Dienst 7, 11  
Windows-Eventlog 55  
Windows Process Explorer 27  
Windows SDK 10, 343  
Windows-Zeichensatz 60  
WITH GRANT OPTION 129  
WITH NOWAIT 65  
work\_mem 29, 49, 188, 213  
Worker-Prozesse 172, 233  
Write Ahead Log 29  
Writer-Prozess 26

**X**

XAMMP 360  
XID 38  
XID Wraparound 37 f.  
XML-Format 217  
xmltable 95  
XStream 443, 449  
XStream-Client 452  
XStream-Schnittstelle 458

**Z**

Zeichenkette 305  
Zeichensatz 58  
Zeichensatz-Konvertierung 59, 113  
Zeilenstatistiken 147  
Zertifikat 132, 135  
Zieldatenbank laden 22  
Zielumgebung 418  
Zugriff, konkurrierender 63  
Zugriffsbeschränkung 136  
Zugriffskontrolle 136  
Zugriffsmethoden 211, 264