PostgreSQL mit nativer Partitionierung und Replikation

Schwerer Brocken

Susanne Schmidt

PostgreSQL springt auf Version 10 und empfiehlt sich für den Unternehmenseinsatz und Big-Data-Anwendungen.

Nach Version 9.6 bringt die nächste Hauptversion 10 der relationalen Open-Source-Datenbank PostgreSQL wichtige Neuerungen bei Unternehmensfunktionen, etliche kleine Verbesserungen und höheres Tempo. Zu den Highlights gehören native Partitionierung, logische Replikation und schlauere Statistiken für den Query Planner.

Das native Partitionieren von Tabellen arbeitet schneller als das bisherige Verfahren mit Triggern und ist zudem besser lesbar und wesentlich kürzer. Dieses Feature erzeugt eine Muttertabelle mit einem Basisschema, von dem Kindtabellen in per Schema festgelegten Stücken abgeleitet sind. Man verwendet Partitionen für Daten, die sich gut zusammenfassen und auslagern lassen, etwa monatsweise gruppierte Logs. Man erzeugt also eine Muttertabelle und deklariert eine Partition:

```
CREATE TABLE users (
  username text,
  login    timestamptz
) PARTITION BY range (login);
```

Versucht man nun, in die Muttertabelle „users“ Daten einzufügen, schlägt der *INSERT* fehl:

```
ERROR: no partition of relation "users" ⤷
                                found for row
```

Es müssen also Partitionen angelegt werden, weil die Muttertabelle nur lesbar ist. Später erreicht man lesend alle Datensätze in der Muttertabelle – die Partitionen enthalten nur die jeweiligen Bereiche (Ranges). Man legt beispielsweise Partitionen an, die jeweils einen Monat an Login-Daten erhalten sollen, wie Listing 1 zeigt. Anschließend fügt man wie üblich Daten ein. Datumswerte außerhalb der festgelegten Range schlagen fehl – man kann sie weder in die Muttertabelle (an den in den Partitionen festgelegten Ranges vorbei) noch in die Partitionen einfügen (siehe Listing 2).

Ein *SELECT*-Befehl funktioniert auf der Muttertabelle mit allen Daten und auf der Partition, die nur die Daten enthält, die in ihr Schema gehören. Das gilt auch für *DELETE*: So kann man etwa in einem Durchgang über die Muttertabelle kaskadierend alle Partitionen löschen.

Legt man nachträglich Partitionen an, muss man zunächst vorhandene Partitionen mit *detach* trennen, anschließend mit *attach* wieder anfügen und auf das Reorganisieren der Daten warten. In dieser Zeit kann die Muttertabelle keine neuen Daten empfangen. Deswegen empfiehlt es sich, alle voraussichtlich benötigten Partitionen zu Beginn anzulegen. Des Weiteren muss man im Vorfeld auf Details der Primary Keys achten, die nicht in der Muttertabelle gesetzt werden können, sondern nur in den Partitionen.

Tabellen abonnieren

PostgreSQL beherrscht das Replizieren von Datenbanken; dabei wird das Write Ahead Log (WAL) der Primary-Datenbank zum Secondary-Server geschickt, dort abgespielt und damit die Replizierung der Daten erzielt. Der Secondary-Server beantwortet nur Leseanfragen und nimmt keine Daten zum Schreiben entgegen. Versagt der Primary-Server, kann man im Betrieb auf den Secondary-Server umschwenken (Physical Replication). Im Unterschied dazu arbeitet die neue logische (logical) Replikation nach dem Publisher-Subscriber-Modell und erlaubt es, Änderungen einer Tabelle zu abonnieren. Man setzt dazu den Parameter *wal_level* auf *logical* und erzeugt anschließend eine *PUBLICATION*:

```
CREATE PUBLICATION publication_new_data ⤷
                    FOR TABLE users, addresses;
```

Damit ist die Publikation verfügbar und kann von Abonnenten benutzt werden. Derzeit kann man nur ganze Tabellen als Publikation zur Verfügung stellen. Das Erzeugen eines Abonnenten ist ebenfalls unkompliziert:

```
CREATE SUBSCRIPTION subscription_new_data
CONNECTION  'host=theprimary dbname= ⤷
                                 mydatabase ...'
PUBLICATION  publication_new_data;
```

Fügt man einer existierenden Publikation neue Tabellen hinzu, muss man mit *ALTER SUBSCRIPTION my_subscription REFRESH PUBLICATION* dem Abonnenten mitteilen, dass es etwas Neues gibt. Zusätzlich steuert man mit *WITH*, ob die Tabelle alle existierenden Daten mitschicken soll oder nur die seit Beginn der Subscription. Mit *pg_stat_subscription* lässt sich die Replikation überwachen.

Wer mit großen Datenmengen hantiert, erhofft sich von einer Datenbank schnelle Resultate der Queries. Dazu dient üblicherweise der Query Planner, der anhand verschiedener statistischer Verfahren versucht, geeignete Optimie-

rungen vorherzusehen. Nach der Einführung der Parallel Queries in der letzten Version 9.6, die man lediglich einschalten muss, gibt es nun ein weiteres Feature, das den Query Planner verbessert: Correlated Statistics bringen einen Hinweis an Tabellen an, welche Spalten korrelieren. Dies geschieht manuell durch den Entwickler und hilft dem Query Planner, Optimierungen auf korrelierte Tabellen anzuwenden. PostgreSQL soll dazu *STATISTICS* erzeugen:

```
CREATE STATISTICS stats_population_age
       (dependencies)
    ON population, age
  FROM population_germany;
```

Damit stellt man für den Query Planner eine Verbindung zwischen *age* und *population* in der Tabelle *population_germany* her. Ein typischer Anwendungsfall ist etwa die Verbindung von Postleitzahl und Stadt. Man kann verschiedene Verbindungstypen nutzen, derzeit Dependencies (direkte Verbindungen, die man gerade in mühevoller Designarbeit herausdenormalisiert hat) und Ndistinct.

Verbesserte Parallelität

PostgreSQL 9.6 erhielt parallele Queries als Basisfeature – man schaltete es lediglich ein, und es beschleunigte alle Abfragen. In dieser ersten Variante gab es den Parallel Sequential Scan; Version 10 erweitert ihn auf alle Segmente der Datenbank und auf verschiedene Formen von Queries, die nun ebenfalls parallel abgearbeitet werden: Parallel Bitmap Heap Scan, Parallel Index Scan, Parallel Index Only Scan, Gather Merge, Parallel Merge Join und andere – die kommende Version 11 der Datenbank soll die Liste erweitern, etwa um parallele Queries in Kombination mit *INSERT*s.

Auch die vorhandene Replikation hat mit dem Feature Quorum Commit for Synchronous Replication mehr Stabilität erhalten und kann dadurch den Verlust mehrerer Datenbank-Nodes ausgleichen. Dabei muss eine bestimmte Zahl Hosts bestätigen, dass sie einen Commit erhalten haben. Hinzu kommt das kleine, aber hilfreiche Feature, eine Liste von Hosts für den Failover anzugeben, anhand derer die Software sich mit dem ersten verfügbaren Host verbindet.

In Version 10 gibt es Rollen für die Rechtevergabe, die sich ums Monitoring drehen. Man kann damit ohne Login Zugriff etwa auf Settings und Tabellengrößen erhalten:

```
GRANT pg_read_all_settings TO
                              my_monitoring_user;
```

Die Top-Level-Rolle *pg_monitor* kann alle Monitoring-Rollen ausfüllen. Mit SCRAM kommt ein weiterer Authentifizierungsmechanismus hinzu, den allerdings die Clients beherrschen müssen – dazu müssen Letztere auf *libpq* basieren.

Die Entwickler haben die Benamsung der Dateien rund ums Logging geändert: Was bisher *xlog* im Namen hatte, führt stattdessen die Bezeichnung *wal* – das Verzeichnis *pg_xlog* etwa heißt nun *pg_wal*, das Tool *pg_xlogdump pg_waldump* und so weiter. Das Gleiche gilt für den Errorlog-Output. Zwar ist das neue Namensschema logischer, jedoch müssen Administratoren gegebenenfalls ihre Monitoring-Checks, Skripte und Orchestrierungskonfigurationen anpassen.

Die Debatte um Nutzen und Nachteile von NoSQL-Datenbanken ist zwar verklungen, dennoch kommt PostgreSQL 10 mit einigen Neuerungen in dieser Abteilung: Die neue *XMLTABLE*-Expression erlaubt, XML-Tabellen mit einer SQL-verwandten Syntax zu befragen, und kombiniert dazu SQL mit der XML Path Language (XPath). Auch XMLs Schwester JSON erhält ein wichtiges Feature: Volltextsuche in JSON- und JSONB-Daten – letztere ist eine komprimierte, binäre Repräsentation von JSON-Daten.

Viele Datenbanken arbeiten mit Triggern und Funktionen. Zu den bekannten Triggern *BEFORE* und *AFTER* kommt in PostgreSQL 10 das Feature Transition Table hinzu, das einen neuen *AFTER-STATEMENT*-Trigger implementiert. Er bezieht sich lediglich auf die bearbeiteten Rows. Außerdem vereinfacht die neue Version das Verfahren bei Aggregates wie *sum* und *count*: Bei Fremdtabellen werden sie nicht mehr erst per *SELECT* gewählt, transferiert und anschließend aggregiert, sondern direkt in der Fremdtabelle per *push down aggregates* verarbeitet; danach übermittelt die Datenbank das Resultat.

Beim Suchen hilft ein vereinfachtes *regex_match*, das lediglich den ersten Treffer liefert – bisher meldete *regex_matches* ein ganzes Trefferset. Und anstatt bei irrealen Datumsangaben wie „2017-10-50“ automatisch den 20.11.2017 zu errechnen, liefert PostgreSQL künftig einen Fehler bei Konvertierungen per *to_date* und *to_timestamp*.

Fazit

PostgreSQL 10 bietet zahlreiche nützliche Neuerungen, die dem Datenbankadministrator das Upgrade nahelegen – nutzt man keine der neuen Funktionen, gibt es dennoch spürbare Verbesserungen wie höheres Tempo. Gerade Kleinigkeiten wie das korrekte Melden eines unsinnigen Datums erleichtern Entwicklern den Alltag.

Die bedeutenden Verbesserungen in dieser und der kommenden Version der Datenbank stehen im Zeichen von Big Data, mehr Geschwindigkeit durch Parallelität und verbesserter Replikation und zielen damit vor allem auf den Unternehmenseinsatz mit großen Datenbeständen. (tiw)

Listing 1: Logdaten nach Monaten partitionieren

```
CREATE TABLE login_january PARTITION OF users (
  username,
  primary key (username)
) FOR VALUES FROM ('2017-01-01') to ('2017-01-31');

CREATE TABLE login_february PARTITION OF users (
  username,
  primary key (username)
) FOR VALUES FROM ('2017-02-01') to ('2017-02-28');

/* und so weiter für zusätzliche Monate */
```

Listing 2: Werte außerhalb der Range werden abgelehnt

```
INSERT INTO login_january (username, login) VALUES ('susi', '2017-03-10');

ERROR:   new row for relation "login_january" violates partition constraint
DETAIL:  Failing row contains (susi, 2017-03-10 00:00:00+00)
```

c't-Wertung

- ⊕ einfacheres natives Partitionieren von Tabellen
- ⊕ logische Replikation (Publisher-Subscriber-Modell)
- ⊕ beschleunigter Query Planner und verbessertes Parallelisieren

Susanne Schmidt

ist Politologin und arbeitet bei SysEleven GmbH als Senior-Developerin.

Alle Links: www.ix.de/ix1711064