Technologia i rozwiązania

PostgreSQL
Receptury dla administratora

Poznaj najlepsze przepisy na pracę z PostgreSQL!

Tu znajdziesz rozwiązania najczęściej spotykanych problemów!

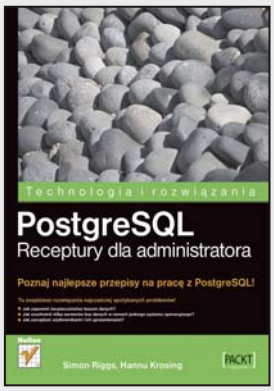
- Jak zapewnić bezpieczeństwo bazom danych?
- Jak uruchomić kilka serwerów baz danych w ramach jednego systemu operacyjnego?
- Jak zarządzać użytkownikami i ich uprawnieniami?

Simon Riggs, Hannu Krosing

Helion

PACKT PUBLISHING

PostgreSQL. Receptury dla administratora



Autor: Simon Riggs, Hannu Krosing
Tłumaczenie: Mikołaj Szczepaniak
ISBN: 978-83-246-3061-5
Tytuł oryginału: PostgreSQL 9 Administration Cookbook
Format: 170×230, stron: 408

Poznaj najlepsze przepisy na pracę z PostgreSQL!

- Jak zapewnić bezpieczeństwo bazom danych?
- Jak uruchomić kilka serwerów baz danych w ramach jednego systemu operacyjnego?
- Jak zarządzać użytkownikami i ich uprawnieniami?

PostgreSQL to jedna z najbardziej zaawansowanych baz danych o otwartym kodzie źródłowym. Przez wiele lat była niedoścignionym wzorem dla innego darmowego rozwiązania – MySQL. Dziś znajduje zastosowanie wszędzie tam, gdzie wymagana jest najwyższa niezawodność i wydajność, a brak konieczności zapłaty gra kluczową rolę. Stosunek jakości do ceny w przypadku PostgreSQL zmierza do nieskończoności!

Trzymasz w rękach książkę zawierającą liczne przepisy na najlepsze wykorzystanie PostgreSQL. System ten sprawdza się zawsze, gdy chcesz szybko i bezproblemowo osiągnąć zamierzone cele. W trakcie lektury dowiesz się, jak nawiązać połączenie z serwerem, skorzystać z graficznych lub tekstowych narzędzi administracyjnych oraz bezpiecznie zmienić hasło administratora. Ponadto nauczysz się kontrolować przestrzeń dyskową wykorzystywaną przez poszczególne bazy danych, tworzyć tabele, ładować dane oraz zarządzać użytkownikami i ich uprawnieniami. Autorzy dużo miejsca poświęcają kwestii bezpieczeństwa. W końcu dane to najcenniejsza rzecz, jaką przechowuje się w bazach! Każdy z rozdziałów przynosi ogrom wiedzy o różnym poziomie skomplikowania. Zaawansowanych użytkowników zainteresuje rozdział poświęcony replikacji, a tych początkujących rozdział traktujący o uruchamianiu i zatrzymywaniu serwera baz danych. Ta książka przyda się po prostu wszystkim użytkownikom PostgreSQL!

- Zalety PostgreSQL w kontekście innych rozwiązań bazodanowych
- Udostępnianie serwera w sieci
- Zastosowanie narzędzia psql do wykonywania zapytań
- Sprawdzanie wersji serwera
- Lista baz danych na serwerze
- Planowanie nowej bazy danych
- Parametry, ich znaczenie i ustawianie
- Uruchamianie i zatrzymywanie serwera
- Ponowne ładowanie plików konfiguracyjnych
- Przyznawanie użytkownikom własnych baz danych
- Wiele serwerów baz danych w ramach jednego systemu operacyjnego
- Generowanie danych testowych
- Tworzenie kont użytkowników i zarządzanie nimi oraz ich uprawnieniami
- Równoległe wykonywanie zadań – polecenie pg_batch
- Monitorowanie i diagnostyka serwera PostgreSQL
- Przygotowywanie kopii bezpieczeństwa

Zobacz, co możesz osiągnąć razem z bazą PostgreSQL!

Helion SA
ul. Kościuszki 1c
44-100 Gliwice
tel. 32 230 98 63
e-mail: helion@helion.pl

Spis treści

3

ROZDZIAŁ

Konfiguracja

Ten rozdział zawiera następujące podrozdziały:

- „Lektura podręcznika użytkownika (RTFM)”
- „Planowanie nowej bazy danych”
- „Zmiana parametrów na poziomie programów”
- „Jakie są bieżące ustawienia konfiguracyjne?”
- „Które parametry zawierają wartości inne niż ustawienia domyślne?”
- „Aktualizacja pliku parametrów”
- „Ustawianie parametrów dla konkretnych grup użytkowników”
- „Lista podstawowych zadań związanych z konfiguracją serwera”
- „Dodawanie modułu zewnętrznego do systemu PostgreSQL”
- „Uruchamianie serwera w trybie oszczędzania energii”

Wprowadzenie

Otrzymuję mnóstwo pytań dotyczących ustawiania parametrów w systemie PostgreSQL.

Wszyscy jesteśmy zajęci, zatem większość z nas szuka źródła informacji, które pozwoliłoby w ciągu pięciu minut rozstrzygnąć wszystkie wątpliwości. Właśnie taki jest cel każdej książki złożonej z receptur, więc jesteśmy na najlepszej drodze do osiągnięcia wymarzonego ideału.

Niektórzy sądzą, że istnieją jakieś magiczne, uniwersalne ustawienia parametrów, które podniosą wydajność ich systemów. Poświęcają długie godziny na lekturę rozmaitych książek w poszukiwaniu wskazówek. Innych nie opuszcza zadowolenie, ponieważ odkryli jakąś witrynę internetową, która **wszystko wyjaśnia** — są pewni, że ich baza danych jest skonfigurowana prawidłowo.

W większości przypadków zrozumienie ustawień nie stanowi większego problemu. Dużo trudniejsze jest wypracowanie najlepszych ustawień. Co więcej, nawet optymalne ustawienia nierzadko zmieniają się wraz z rozwojem sytuacji. W tym rozdziale skoncentrujemy się przede wszystkim na tym, jak, kiedy i gdzie należy zmieniać ustawienia parametrów.

Lektura podręcznika użytkownika (RTFM)

Skrót **RTFM** (od ang. *Reading the Fine Manual*) często jest używany w roli bardziej opryskliwego odpowiednika stwierdzenia „nie przeszkadzaj, jestem zajęty". Co ciekawe, zachęta do przeczytania podręcznika użytkownika w zdecydowanej większości przypadków jest najlepszą możliwą radą. Nie należy z góry odrzucać podobnych sugestii — powinniśmy raczej korzystać z tak cennych wskazówek. Warto przy tym pamiętać, że zawsze należy sięgać po podręcznik użytkownika, którego wersja odpowiada wersji serwera, na którym pracujemy.

Podręcznik użytkownika systemu PostgreSQL jest napisany w bardzo przemyślany sposób. Co więcej, omawia wybrane zagadnienia w sposób wyczerpujący. Jedną z jego największych wad jest to, że dokumenty składające się na ten podręcznik nie są zorganizowane z myślą o osobach, które dopiero poznają system PostgreSQL. Rozkład materiału przystosowano raczej do potrzeb osób poszukujących konkretnych informacji technicznych, tak aby mogły błyskawicznie oceniać, czy napotykane problemy wynikają na przykład z błędów samych użytkowników. Podręcznik w niektórych przypadkach odpowiada na pytanie **Co?**, ale już dużo rzadziej odpowiada na pytania **Dlaczego?** czy **Jak?**.

Ponieważ sam uczestniczyłem w pisaniu wybranych sekcji dokumentacji systemu PostgreSQL, uważam za swój obowiązek zachęcić czytelnika do lektury tego materiału. Mimo wszystkich zalet podręcznika, ta książka zawiera mnóstwo przydatnych informacji, których nie można znaleźć w tamtych dokumentach.

Jak to zrobić…

Najważniejsze dokumenty dla każdego wydania są dostępne na następującej stronie internetowej: *http://www.postgresql.org/docs/manuals/*.

Największym zainteresowaniem cieszą się następujące fragmenty dokumentacji:

- dokumentacja poleceń języka SQL oraz podręcznik użytkownika narzędzi klienta i serwera (*http://www.postgresql.org/docs/9.0/interactive/reference.html*);
- konfiguracja (*http://www.postgresql.org/docs/9.0/interactive/runtime-config.html*);
- funkcje (*http://www.postgresql.org/docs/9.0/interactive/functions.html*).

Czytelnik może też pobrać podręcznik użytkownika w formie pliku PDF, który w pewnych przypadkach ułatwia wyszukiwanie informacji. Szczerze odradzam drukowanie tego dokumentu! Cała dokumentacja zajmuje ponad 2000 stron A4.

Jak to działa…

Dokumenty systemu PostgreSQL napisano w języku SGML, który pod wieloma względami przypomina język XML, choć w paru aspektach jest nieco inny. Pliki języka SGML są następnie przetwarzane i konwertowane na pliki w formatach HTML, PDF itp.

To nie wszystko…

Warto też zajrzeć do serwisu wiki w ramach witryny *postgresql.org*. Dodatkowe informacje można znaleźć także pod adresem *http://wiki.postgresql.org*.

Planowanie nowej bazy danych

Planowanie nowej bazy danych bywa żmudnym zadaniem. Dla wielu użytkowników mnogość opcji jest przytłaczająca — w tym podrozdziale zaproponujemy kilka przydatnych koncepcji. Należy wystrzegać się dróg na skróty i nieprzemyślanych działań w nadziei, że nasza ograniczona wiedza na zawsze pozwoli rozstrzygać wszelkie wątpliwości.

Przygotowania

Czytelnik powinien być przygotowany na to zadanie. Nie należy jednak oczekiwać, że ktoś dokładnie powie, co robić. Jeśli nie dysponujemy jasno sprecyzowanymi wymaganiami, powinniśmy sami je spisać, oznaczając każdą pozycję jako założenie, nie wymaganie — nie wolno mylić obu pojęć.

Należy przeprowadzić tyle operacji, ile będzie konieczne do uzgodnienia wymagań ze wszystkimi zainteresowanymi. Po sporządzeniu wymagań można przystąpić do budowy prototypu.

Jak to zrobić…

Należy przygotować dokument obejmujący następujące aspekty:

- **Projekt bazy danych:** należy zaplanować projekt bazy danych.
  - Należy obliczyć początkowy rozmiar bazy danych.
- **Analiza transakcji:** jak użytkownicy będą uzyskiwali dostęp do bazy danych?
  - Należy przeanalizować najczęściej używane ścieżki dostępu.
  - Jakie są wymagania odnośnie czasów odpowiedzi?

- **Konfiguracja sprzętowa**
  - Wstępna analiza wydajności — czy wszystkie dane zmieszczą się w pamięci operacyjnej?
- **Plan lokalizacji**
  - Należy wybrać obowiązujące na serwerze schemat kodowania, ustawienia regionalne i strefę czasową.
- **Plan dostępu i zabezpieczeń**
  - Należy zidentyfikować systemy klienckie i wskazać wymagane sterowniki.
  - Należy utworzyć role zgodnie z przyjętym wcześniej planem kontroli dostępu.
  - Należy przygotować plik *pg_hba.conf*.
- **Plan konserwacji:** kto będzie dbał o bieżące funkcjonowanie bazy danych? Jak będzie to robił?
- **Plan dostępności:** należy przemyśleć wymagania w zakresie dostępności.
  - Należy określić parametr `checkpoint_timeout`.
  - Należy zaplanować mechanizm tworzenia kopii zapasowych i poddać je testom.
- **Plan wysokiej dostępności**
  - Należy zdecydować, czy będzie konieczna replikacja, a jeśli tak — w jakiej formie.

Jak to działa…

Jednym z najważniejszych powodów planowania bazy danych z wyprzedzeniem są utrudnienia związane ze zmianą pewnych aspektów na późniejszych etapach pracy. Problem jest szczególnie dotkliwy w przypadku schematów kodowania i ustawień regionalnych, których zmiana już po wdrożeniu bazy danych może nie tylko wymagać sporych nakładów pracy, ale też wiązać się z koniecznością długiego przestoju. Także konfigurowanie zabezpieczeń po wdrożeniu systemu jest nieporównanie trudniejsze niż definiowanie odpowiednich ustawień z wyprzedzeniem.

To nie wszystko…

Planowanie zawsze jest pomocne. O ile sami doskonale wiemy, co robimy, nasi współpracownicy mogą nie dysponować podobnymi informacjami. Należy więc wszystkim wyjaśnić nasze zamiary, aby nie tracić czasu w przyszłości. Jeśli nie jesteśmy pewni słuszności jakiegoś założenia, powinniśmy zbudować prototyp, który ułatwi nam podjęcie ostatecznej decyzji — szkielet rozwiązań administracyjnych powinien mieć priorytet nie mniejszy niż właściwe czynności. Warto też przygotować listę decyzji do rozstrzygnięcia i kolejno pracować nad jej pozycjami.

Ta receptura celowo jest bardzo krótka. Każdy ma własny sposób osiągania wyznaczonych celów, zatem niezwykle ważne jest unikanie precyzyjnego opisywania, jak zrealizować poszczególne zadania. Jeśli czytelnik dysponuje już planem, to świetnie. Jeśli nie, należy zastanowić się, co należy zrobić, sporządzić listę i wreszcie wykonać zaplanowane zadania punkt po punkcie.

Zmiana parametrów na poziomie programów

System PostgreSQL umożliwia ustawianie niektórych parametrów osobno dla każdej sesji lub każdej transakcji.

Jak to zrobić…

Wartość parametru można zmienić w czasie trwania sesji, na przykład za pomocą następującego wyrażenia:

```
SET work_mem = '16MB';
```

W takim przypadku użyta wartość będzie obowiązywała dla każdej kolejnej transakcji. Istnieje też możliwość zmiany ustawień tylko na czas trwania bieżącej transakcji:

```
SET LOCAL work_mem = '16MB';
```

Tak zdefiniowane ustawienie będzie obowiązywało tylko do momentu użycia następującego wyrażenia:

```
RESET work_mem;
```

lub

```
RESET ALL;
```

Wyrażenia `SET` i `RESET` to polecenia języka SQL, które można stosować za pośrednictwem dowolnego interfejsu. Oba wyrażenia mogą być używane tylko dla parametrów serwera PostgreSQL, czyli parametrów wpływających na funkcjonowanie tego serwera (choć niekoniecznie na cały serwer). W systemie mogą występować inne parametry, na przykład parametry sterownika JDBC, których nie można ustawiać w ten sposób. Więcej informacji na ten temat można znaleźć w rozdziale „Kontrola serwera”.

Jak to działa…

Zmiana wartości parametru w czasie trwania sesji, na przykład za pomocą następującego wyrażenia:

```
SET work_mem = '16MB';
```

powoduje, że w perspektywie katalogowej pg_settings są zapisywane następujące dane:

```
postgres=# SELECT name, setting, reset_val, source
                       FROM pg_settings WHERE source = 'session';
   name   | setting | reset_val | source
----------+---------+-----------+---------
 work_mem | 16384   | 1024      | session
```

Przytoczone ustawienia obowiązują do momentu użycia następującego polecenia:

```
RESET work_mem;
```

Powyższe wyrażenie powoduje przywrócenie wartości reset_val i — tym samym — ponowne umieszczenie w kolumnie source wartości domyślnej.

```
  name   | setting | reset_val | source
---------+---------+-----------+---------
work_mem | 1024    | 1024      | default
```

To nie wszystko…

Wartość parametru można zmienić także w czasie trwania transakcji — wystarczy użyć wyrażenia w postaci:

```
SET LOCAL work_mem = '16MB';
```

Powyższe polecenie wprowadza następującą zmianę w perspektywie katalogowej pg_settings:

```
postgres=# SELECT name, setting, reset_val
                       FROM pg_settings WHERE source = 'session';
   name   | setting | reset_val | source
----------+---------+-----------+---------
work_mem  | 1024    | 1024      | session
```

Co to? Co stało się z użytą wartością parametru? Okazuje się, że wyrażenie SET LOCAL wpływa tylko na transakcję, w ramach której jest wykonywane, czyli w naszym przypadku właśnie na polecenie SET LOCAL. Musimy więc wykonać to polecenie w ramach bloku transakcji, aby wprowadzona zmiana była uwzględniana w kolejnych operacjach składających się na tę transakcję:

```
BEGIN;
SET LOCAL work_mem = '16MB';
```

Efekt wykonania tych wyrażeń widać teraz w perspektywie katalogowej pg_settings:

```
postgres=# SELECT name, setting, reset_val, source
                        FROM pg_settings WHERE source = 'session';
   name   | setting | reset_val | source
----------+---------+-----------+---------
work_mem  | 16384   | 1024      | session
```

Warto też zwrócić uwagę na to, że w kolumnie source widnieje wartość session, nie wartość transaction, która w tych okolicznościach wydawałaby się bardziej naturalna.

Jakie są bieżące ustawienia konfiguracyjne?

W pewnym momencie każdy z nas zada sobie pytanie „Jakie są **bieżące** ustawienia konfiguracyjne?”.

Jak to zrobić…

W pierwszym odruchu większość użytkowników odpowiada: „wystarczy zajrzeć do pliku *postgresql.conf*”. To rozwiązanie zdaje egzamin pod warunkiem, że dysponujemy tylko jednym plikiem z parametrami. Jeśli istnieją dwa takie pliki, może okazać się, że czytamy niewłaściwy plik! (Skąd właściwie mielibyśmy wiedzieć, który jest właściwy)? W tej sytuacji najbezpieczniejszym sposobem jest rezygnacja z przeglądania plików tekstowych na rzecz odwołania się do samego serwera.

Z podrozdziału „Zmiana parametrów na poziomie programów” wiemy też, że każdy parametr ma określony zasięg, decydujący o tym, kiedy jego wartość może być skutecznie zmieniana. O ile niektóre parametry można ustawiać za pośrednictwem pliku *postgresql.conf*, o tyle pozostałe podlegają zmianom już po załadowaniu zawartości tego pliku. Oznacza to, że bieżący stan ustawień konfiguracyjnych może być inny, niż to wynika z zapisów we wspomnianym pliku.

Do uzyskania obowiązujących ustawień można użyć polecenia SHOW, na przykład w tej formie:

```
postgres=# SHOW work_mem;
work_mem
----------
1MB
(1 row)
```

Warto jednak pamiętać, że polecenie SHOW prezentuje wartość wskazanego parametru obowiązującą w momencie wywołania i że ta wartość może być w tym czasie zmieniana.

Innym sposobem uzyskiwania bieżących ustawień jest dostęp do perspektywy katalogowej systemu PostgreSQL, nazwanej pg_settings:

```
postgres=# \x
Expanded display is on.
postgres=# SELECT * FROM pg_settings WHERE name = 'work_mem';
[ RECORD 1 ] -------------------------------------------------------------
name       | work_mem
setting    | 1024
unit       | kB
category   | Resource Usage / Memory
short_desc | Sets the maximum memory to be used for query workspaces.
extra_desc | This much memory can be used by each internal sort operation and
           ↪hash table before switching to temporary disk files.
context    | user
vartype    | integer
source     | default
min_val    | 64
max_val    | 2147483647
enumvals   |
boot_val   | 1024
reset_val  | 1024
sourcefile |
sourceline |
```

Oznacza to, że polecenia `SHOW` można używać zarówno do uzyskiwania wybranych ustawień, jak i do uzyskiwania dostępu do pełnego wykazu szczegółów za pośrednictwem odpowiedniej tabeli katalogowej.

Jak to działa…

Każdy parametr jest buforowany w ramach każdej sesji, dzięki czemu dysponujemy błyskawicznym dostępem do ustawień reprezentowanych przez poszczególne parametry. Zastosowany model powoduje, że odczytywanie ustawień zapisanych w parametrach jest dziecinnie proste.

Warto pamiętać, że wyświetlane wartości nie zawsze odzwierciedlają ustawienia dotyczące całego serwera — wiele spośród tych parametrów jest ściśle związanych z bieżącą sesją. To odróżnia system PostgreSQL od wielu innych baz danych i w dużej mierze decyduje o jego elastyczności.

Które parametry zawierają wartości inne niż ustawienia domyślne?

Często musimy sprawdzić, które parametry zostały już zmienione lub czy nasze zmiany zostały prawidłowo uwzględnione w ustawieniach systemu PostgreSQL.

Jak to zrobić…

```
postgres=# SELECT name, source, setting
                      FROM pg_settings
                      WHERE source != 'default
                        AND source != 'override'
                      ORDER by 2, 1;
           name             |        source        | setting
----------------------------+----------------------+----------
application_name            | client               | psql
log_timezone                | command line         | PL
TimeZone                    | command line         | PL
timezone_abbreviations      | command line         | Default
archive_command             | configuration file   | (disabled)
archive_mode                | configuration file   | off
archive_timeout             | configuration file   | 5
bgwriter_delay              | configuration file   | 10
checkpoint_timeout          | configuration file   | 30
log_checkpoints             | configuration file   | on
log_destination             | configuration file   | stderr
log_filename                | configuration file   | log%Y
logging_collector           | configuration file   | on
log_line_prefix             | configuration file   | %t[%p]
log_min_messages            | configuration file   | log
max_prepared_transactions   | configuration file   | 5
max_standby_delay           | configuration file   | 90
port                        | configuration file   | 5443
max_stack_depth             | environment variable | 2048
work_mem                    | session              | 204800
(29 rows)
```

(Wartość `Override` wykluczono z wyników dla poprawy czytelności).

Jak to działa…

Na podstawie danych widocznych w perspektywie `pg_settings` można bez trudu stwierdzić, które wartości są różne od ustawień domyślnych i jakie jest źródło bieżących wartości.

Polecenie `SHOW` nie określa, czy poszczególne parametry mają przypisane wartości domyślne. Działanie tego polecenia ogranicza się do zwrócenia interesującej nas wartości, co w żaden sposób nie rozstrzyga, co i dlaczego zostało zmienione.

Jeśli źródłem zmiany są zapisy w pliku konfiguracyjnym, wartości w kolumnach `sourcefile` i `sourceline` także są ustawione. Analiza tych wartości bywa przydatna w procesie identyfikacji źródeł poszczególnych ustawień konfiguracyjnych.

To nie wszystko…

Kolumna `setting` perspektywy `pg_settings` zawiera co prawda bieżącą wartość parametru, jednak warto zwrócić uwagę na kolumny `boot_val` i `reset_val`, które zawierają odpowiednio wartość ustawioną na etapie inicjalizacji klastra baz danych PostgreSQL (`"initdb"`) oraz wartość, która zostanie przywrócona w danym parametrze wskutek użycia komendy `RESET`.

Kto to ustawił?

Parametr `max_stack_depth` jest wyjątkowy — z danych prezentowanych przez perspektywę `pg_settings` wynika, że jego wartość została ustawiona przez zmienną środowiskową, mimo że w rzeczywistości została zmieniona przez polecenie `ulimit -s` (w systemach Linux i Unix). Tylko w systemie Windows parametr `max_stack_depth` wymaga bezpośredniego ustawienia.

Także ustawienia strefy czasowej są określane na podstawie środowiska systemu operacyjnego, zatem ich bezpośrednie ustawianie nie jest konieczne. Co ciekawe, z danych prezentowanych przez perspektywę `pg_settings` wynika, że są to ustawienia definiowane z poziomu wiersza poleceń.

Aktualizacja pliku parametrów

Podstawowym miejscem definiowania wartości parametrów na potrzeby serwera PostgreSQL jest plik parametrów. W pliku parametrów nazwanym *postgresql.conf* można ustawić wszystkie parametry tego serwera.

Istnieją też dwa inne pliki parametrów, nazwane *pg_hba.conf* i *pg_ident.conf*. Zapisy zawarte w obu tych plikach mają związek z zarządzaniem połączeniami i zabezpieczeniami, zatem wrócimy do nich w dalszych rozdziałach poświęconych tym zagadnieniom.

Przygotowania

W pierwszym kroku należy znaleźć plik *postgresql.conf* (odpowiednią procedurę opisano już wcześniej).

Jak to zrobić…

Wszystkie parametry można ustawić w pliku parametrów nazwanym *postgresql.conf*. Niektóre spośród tych parametrów są uwzględniane tylko podczas pierwszego uruchamiania serwera. Typowym przykładem jest parametr `shared_buffers`, definiujący rozmiar współdzielonej pamięci podręcznej.

Wiele innych parametrów można zmienić w czasie działania serwera. Po zmianie wybranych parametrów należy wykonać na serwerze operację `reload`, aby wymusić na systemie PostgreSQL ponowne załadowanie pliku *postgresql.conf*:

```
pg_ctl -D data reload
```

Plik konfiguracyjny *postgresql.conf* ma postać normalnego pliku tekstowego, którego zawartość można bez trudu edytować. Większość parametrów od początku jest zapisana w tym pliku, zatem nasza rola sprowadza się do ich wyszukania i zastąpienia dotychczasowych ustawień własnymi wartościami.

Jak to działa…

W przypadku dwukrotnego ustawienia tego samego parametru w dwóch różnych miejscach pliku konfiguracyjnego, zostanie uwzględnione ostatnie ustawienie. Taka sytuacja może rodzić sporo nieporozumień, jeśli na przykład dopisujemy ustawienia na końcu pliku, zatem należy tego unikać.

Zalecaną praktyką jest pozostawienie oryginalnej struktury pliku konfiguracyjnego i sama edycja wybranych wartości lub wręcz rozpoczęcie pracy od pustego pliku i umieszczenie w nim tylko tych parametrów, których wartości chcemy zmienić. Sam preferuję model polegający na stosowaniu plików złożonych z wartości innych niż domyślne. Takie rozwiązanie ułatwia ocenę sytuacji.

Niezależnie od wybranej metody zaleca się zapisywanie kopii starszych wersji plików *.conf*. Wystarczy albo ręcznie skopiować pliki przeznaczone do modyfikacji, albo posłużyć się systemem kontroli wersji, na przykład systemem SVN.

To nie wszystko…

W pliku *postgresql.conf* można też stosować dyrektywę `include`. Oznacza to, że plik *postgresql.conf* można skonstruować w taki sposób, aby odwoływał się do innych plików, które z kolei mogą odwoływać się do jeszcze innych plików itd. Takie rozwiązanie może nam ułatwić lepszą organizację parametrów, o ile nie zastosujemy zbyt skomplikowanej struktury.

Ustawianie parametrów dla konkretnych grup użytkowników

System PostgreSQL obsługuje wiele różnych sposobów definiowania wartości parametrów dla poszczególnych grup użytkowników.

Jak to zrobić…

Ustawienia dla wszystkich użytkowników bazy danych `saas` można ustawiać za pomocą wyrażenia:

```
ALTER DATABASE saas
SET configuration_parameter = value1;
```

Ustawienia dla użytkownika `simon` połączonego z dowolną bazą danych można ustawić za pomocą wyrażenia:

```
ALTER ROLE simon
SET configuration_parameter = value2;
```

Istnieje też możliwość ustawienia parametru dla konkretnego użytkownika połączonego z określoną bazą danych:

```
ALTER ROLE simon
IN DATABASE saas
SET configuration_parameter = value3;
```

Sam użytkownik nie wie, że specjalnie z myślą o nim wykonano tego rodzaju wyrażenie. Tak zdefiniowane ustawienia w większości przypadków pełnią funkcję wartości domyślnych i jako takie mogą być zmieniane przez użytkownika potrzebującego innych wartości parametrów.

Jak to działa…

Parametry można ustawiać na następujących poziomach:

- bazy danych;
- użytkownika (w systemie PostgreSQL użytkownika określa się mianem roli);
- kombinacji bazy danych i użytkownika.

Wartość domyślne tych parametrów są nadpisywane wartościami definiowanymi na kolejnych poziomach powyższej listy.

Oznacza to, że w przypadku wykonania powyższych trzech wyrażeń języka SQL będą obowiązywały następujące ustawienia:

- użytkownik `hannu` łączy się z bazą danych `saas` — zastosowanie ma wartość `value1`;
- użytkownik `simon` łączy się z bazą danych inną niż `saas` — zastosowanie ma wartość `value2`;
- użytkownik `simon` łączy się z bazą danych `saas` — zastosowanie ma wartość `value3`.

System PostgreSQL implementuje te reguły dokładnie tak, jakby użytkownik ręcznie użył odpowiednich wyrażeń `SET` bezpośrednio po nawiązaniu połączenia.

Lista podstawowych zadań związanych z konfiguracją serwera

System PostgreSQL domyślnie jest skonfigurowany pod kątem pracy w roli systemu współdzielonego, jednak wielu użytkowników woli dysponować dedykowanymi systemami baz danych. Jednym z celów projektu PostgreSQL jest zapewnienie możliwości zgodnego współistnienia tego systemu z innymi programami działającymi po stronie serwera, zatem wykorzystywanie pełnych zasobów serwera byłoby niepożądane. Jeśli jednak administrator systemu wie, że w tym samym systemie nie będą działały inne ważne serwery, może bez obaw zmienić parametry serwera PostgreSQL na wyższe.

Przygotowania

Zanim przystąpimy do modyfikowania parametrów systemu, musimy dysponować dwiema informacjami:

Po pierwsze, musimy znać wielkość fizycznej pamięci RAM przeznaczonej dla systemu PostgreSQL.

Po drugie, musimy wiedzieć, jakiego rodzaju aplikacje będą używały systemu PostgreSQL.

Jak to zrobić…

Jeśli rozmiar bazy danych przekracza 32 MB, zwiększenie wartości parametru `shared_buffers` powinno podnieść wydajność systemu. Teoretycznie można tę wartość zwielokrotnić, jednak musimy mieć na uwadze, że w systemach Linux nieużywany bufor może trafić do pliku wymiany, zatem powinniśmy zachować ostrożność. Nową wartość można ustawić w pliku *postgresql.conf* i stopniowo zwiększać, każdorazowo sprawdzając, czy zmiana przyniosła pożądany efekt.

W przypadku zwiększenia wartości parametru `shared_buffers` na serwerze z systemem operacyjnym innym niż Windows niemal zawsze należy dodatkowo zwiększyć wartość parametru `SHMMAX` samego systemu operacyjnego (na niektórych platformach możemy stanąć przez koniecznością zmiany także innych parametrów).

W systemach Linux, Mac OS i FreeBSD należy albo zmienić zawartość pliku */etc/sysctl.conf*, albo użyć polecenia `sysctl -w` z następującymi wartościami:

- Linux: `kernel.shmmax=`*`wartość`*
- Mac OS: `kern.sysv.shmmax=`*`wartość`*
- FreeBSD: `kern.ipc.shmmax=`*`wartość`*

Więcej informacji na temat tych wartości można znaleźć na stronie *http://www.postgresql.org/docs/8.4/static/kernel-resources.html#SYSVIPC*.

Na przykład w systemie Linux należałoby dodać do pliku */etc/sysctl.conf* następujący wiersz:

```
kernel.shmmax=wartość
```

Wartość parametru `effective_cache_size` nie jest taka ważna. Wbrew pozorom ma dużo mniejsze znaczenie niż parametr `shared_buffers`, zatem wybór właściwej wartości nie powinien nam zająć zbyt wiele czasu.

Jeśli w naszym systemie planujemy wykonywanie wielu operacji zapisu, powinniśmy rozważyć znaczne zwiększenie wartości parametru `wal_buffers` względem wartości domyślnej.

Jeśli planujemy dużą liczbę operacji zapisu i (lub) operowanie na wielkich ilościach danych, być może powinniśmy zwiększyć wartość parametru `checkpoint_segments` względem ustawień domyślnych.

Jeśli na bazie danych są wykonywane wielkie zapytania, powinniśmy rozważyć zmianę domyślnej wartości parametru `work_mem` na większą.

Warto upewnić się, że parametr `autovacuum` jest włączony, chyba że z jakiegoś powodu świadomie podjęliśmy decyzję o jego wyłączeniu. W zdecydowanej większości przypadków wspomniany parametr powinien jednak być włączony (więcej informacji na ten temat można znaleźć w dalszej części tej książki).

Sporym uproszczeniem jest użycie narzędzia dostępnego pod następującym adresem URL: *http://pgfoundry.org/projects/pgtune/*.

Na tym możemy zakończyć proces wstępnego ustawiania parametrów. Nie powinniśmy poświęcać zbyt dużo czasu na poszukiwanie optymalnych ustawień. Większość tych parametrów można zmienić w przyszłości, zatem warto zastosować model przyrostowego doskonalenia poszczególnych wartości.

Początkowo należy koncentrować uwagę przede wszystkim na najważniejszych parametrach i zadbać o ich prawidłowe ustawienia. Po opanowaniu podstaw warto zaopatrzyć się w książkę Grega Smitha poświęconą wydajności systemu PostgreSQL.

W szczególności nie należy zmieniać parametru `fsync`, który zapewnia bezpieczeństwo naszemu systemowi.

Dodawanie modułu zewnętrznego do systemu PostgreSQL

Jedną z zalet systemu PostgreSQL jest jego rozszerzalność. Właśnie rozszerzalność była jednym z oryginalnych celów projektowych stawianych sobie przez twórców tego systemu już w latach osiemdziesiątych ubiegłego wieku. Dla współczesnej wersji PostgreSQL 9.0 istnieje wiele dodatkowych modułów, które można łatwo dołączać do podstawowego serwera PostgreSQL.

Dodatkowe moduły mogą wprowadzać do rozszerzanego systemu udoskonalenia różnych typów:

- dodatkowe funkcje,
- dodatkowe typy danych,
- dodatkowe operatory,
- dodatkowe indeksy.

Co ciekawe, wiele narzędzi i interfejsów klienckich współpracuje z systemem PostgreSQL bez konieczności jakiejkolwiek specjalnej instalacji. W tym podrozdziale omówimy moduły rozszerzające lub zmieniające zachowanie serwera, w tym standardową składnię, funkcje i zachowania wyrażeń języka SQL.

Przygotowania

Musimy najpierw wybrać właściwy moduł do zainstalowania.

Ponieważ do tej pory nie powstał system zarządzania pakietami dla projektu PostgreSQL, dostępne moduły znajdują się w wielu różnych miejscach, w tym na następujących witrynach internetowych:

- Contrib — tzw. jądro systemu PostgreSQL — zawiera wiele przydatnych funkcji. Okazuje się jednak, że istnieje też oficjalna sekcja modułów dodatkowych, określanych mianem modułów contrib. Ich dokumentację można znaleźć pod następującym adresem URL:
  - *http://www.postgresql.org/docs/9.0/static/contrib-dblink-connect.html*
- pgFoundry — witryna internetowa tego projektu open source ma na celu przede wszystkim udostępnianie modułów i narzędzi dla systemu PostgreSQL. Witryna PgFoundry korzysta z tego samego oprogramowania co popularny serwis *SourceForge.net*. Warto więc zajrzeć pod następujący adres URL:
  - *http://pgFoundry.org/*
- Odrębne projekty — wielkie projekty zewnętrzne, na przykład PostGIS, oferują rozbudowane, złożone moduły dodatkowe dla systemu PostgreSQL. Zachęcam do odwiedzenia następującego adresu URL:
  - *http://www.postgis.org/*

Jak to zrobić…

W pewnych przypadkach moduły można dodawać już na etapie instalacji, jeśli korzystamy z autonomicznej aplikacji instalacyjnej, na przykład z programu instalatora **OneClick**.

W pozostałych przypadkach istnieje możliwość instalacji modułu z pakietu, na przykład modułu zapewniającego zgodność z systemem Oracle (*http://www.postgres.cz/index.php/Oracle_functionality*).

W pierwszym kroku należy pobrać interesujący nas pakiet, na przykład: *http://pgfoundry.org/frs/download.php/2420/orafce-3.0.1-1.pg82.rhel5.i386.rpm*.

Pobrany pakiet należy zainstalować, stosując następujące polecenia:

```
rpm -ivh orafce-3.0.1-1.pg90.rhel5.i386.rpm
sudo apt-get install postgresql-8.4-orafce
```

W wielu przypadkach przydatne moduły nie są udostępniane w formie gotowych pakietów. Takie moduły należy instalować ręcznie. Instalacja nie jest trudna i może być ciekawym ćwiczeniem ułatwiającym zrozumienie odpowiednich procedur.

Każdy moduł wymaga nieco innej instalacji. Ogólnie należy zwracać szczególną uwagę na następujące dwa aspekty instalacji modułu:

- instalację obiektów języka SQL dla danego modułu;
- instalacje wymaganych przez ten moduł bibliotek ładowanych dynamicznie (DLL).

Instalacja większości przydatnych modułów wymaga od użytkownika zadbania o oba wymienione powyżej aspekty. Istnieją też moduły, na przykład AutoExplain, które korzystają wyłącznie z bibliotek ładowanych dynamicznie.

- Należy skompilować odpowiednie biblioteki.

Należy postępować według instrukcji dla danego modułu:

- Należy zainstalować bibliotekę w miejscu, w którym będzie dostępna dla serwera:

```
shared_preload_libraries = '$libdir/modlib'
```

Należy utworzyć obiekty bazy danych:

```
psql -d dbname -f SHAREDIR/contrib/module.sql
```

Jak to działa…

System PostgreSQL może dynamicznie ładować biblioteki na trzy sposoby:

- wskutek wydania przez użytkownika bezpośredniego polecenia `LOAD` w ramach sesji;

- na podstawie parametru `shared_preload_libraries` zapisanego w pliku *postgresql.conf* i używanego podczas uruchamiania serwera;
- na początku sesji na podstawie parametru `local_preload_libraries` ustawionego dla konkretnego użytkownika (zdefiniowanego przy użyciu polecenia `ALTER ROLE`).

Funkcje i obiekty systemu PostgreSQL mogą odwoływać się do kodu zawartego w tych bibliotekach — takie rozwiązanie umożliwia ścisłe wiązanie rozszerzeń z działającym procesem serwera. Tak ścisłe związki powodują, że opisana metoda sprawdza się nawet w aplikacjach, które muszą spełniać najwyższe wymagania w zakresie wydajności. Dodatkowe opcje i funkcje nie odbiegają wydajnością od rdzennych elementów systemu.

Uruchamianie serwera w trybie oszczędzania energii

Zużycie energii elektrycznej jest obecnie jednym z najczęściej dyskutowanych tematów. Dziś każdy szuka sposobów, by choć w minimalnym stopniu przyczynić się do ochrony środowiska naturalnego. Nie inaczej jest w przypadku użytkowników systemu PostgreSQL.

Przygotowania

Jeśli serwer PostgreSQL ma być używany sporadycznie lub pozostaje nieaktywny przez dłuższe okresy, warto rozważyć zastosowanie przynajmniej części z poniższych sugestii. Taki serwer może działać na laptopie lub mieć postać nieaktywnego serwera wirtualnego.

Jak to zrobić…

System PostgreSQL to przykład bazy danych działającej po stronie serwera, zatem w czasie, gdy nie ma aktywnych klientów, serwer w praktyce pozostaje bezczynny. Aby zminimalizować aktywność serwera, wystarczy ustawić następujące parametry w pliku *postgresql.conf*:

- `autovacuum = off`
- `wal_writer_delay = 10000`
- `bgwriter_delay = 10000`

Przytoczone ustawienia w wielu zastosowaniach są dalece nieoptymalne i jako takie powinny być stosowane tylko dla serwera, o którym wiadomo, że przez znaczną część czasu będzie nieaktywny. Kiedy obciążenie serwera ponownie wzrośnie, należy przywrócić w powyższych parametrach poprzednie wartości.

Jak to działa…

Istnieje kilka procesów, które stale pozostają aktywne w oczekiwaniu na aktywnych klientów, których żądania będą wymagały efektywnej obsługi. W systemie PostgreSQL istnieje pięć takich procesów:

- proces zapisujący (znany też jako proces zapisujący w tle);
- proces zapisujący dziennik WAL;
- archiwizator (aktywny tylko wtedy, gdy archiwizacja dziennika WAL jest włączona);
- proces odbiorcy dziennika WAL (aktywny tylko wtedy, gdy jest stosowany mechanizm replikacji strumieniowej);
- proces automatycznego czyszczenia (ang. *autovacuum*).

Proces zapisujący w tle domyślnie jest aktywowany co 200 ms. Wartość maksymalna dzieląca kolejne próby wybudzania tego procesu wynosi 10 s, co nie wydaje się szczególnie długim czasem bezczynności. Okazuje się jednak, że można ten proces całkowicie wyłączyć, przypisując parametrowi `bgwriter_lru_maxpages` wartość `0`.

Proces zapisujący w dzienniku WAL domyślnie jest aktywowany co 200 ms. Także w tym przypadku maksymalna wartość odpowiedniego parametru to 10 s. Tego procesu nie można jednak wyłączyć. W przypadku braku operacji zapisu proces nie podejmuje żadnych działań — jego działanie sprowadza się do wybudzania i sprawdzania, czy istnieją ewentualne operacje oczekujące.

Proces archiwizatora (ang. *archiver*) jest aktywowany co 15 sekund i sprawdza, czy zapisano jakieś nowe pliki dziennika WAL. Każda aktywacja powoduje wykonanie operacji na katalogu systemu plików. Okresu dzielącego kolejne wybudzanie tego procesu nie można zmienić za pośrednictwem żadnego parametru.

Proces odbiorcy dziennika WAL jest aktywowany co 100 ms w celu sprawdzenia, czy do serwera nie dotarły nowe dane replikacji. W razie braku takich danych proces ponownie przechodzi w stan uśpienia. Okresu dzielącego kolejne wybudzenia tego procesu nie można zmienić za pośrednictwem żadnego parametru.

Proces automatycznego czyszczenia domyślnie jest aktywowany co minutę. Częstotliwość wybudzania tego procesu można zmienić, przypisując nową wartość parametrowi `autovacuum_naptime`. Proces automatycznego czyszczenia można całkowicie wyłączyć, stosując parametr `autovacuum = off`.

Jeśli stosujemy mechanizm replikacji strumieniowej, nasz serwer będzie aktywowany co 200 ms. Jeśli nie korzystamy z tego mechanizmu, możemy wydłużyć czas pomiędzy kolejnymi wybudzeniami do 10 sekund (zamiast domyślnych 200 ms).

Skorowidz

B

C

D

E

F

G

H

I

J

K

L

Ł

M

N

O

P

Q

R

S

Ś

T

U

V

W

X

Z

Zobacz, co możesz osiągnąć razem

z bazą PostgreSQL!

PostgreSQL to jedna z najbardziej zaawansowanych baz danych o otwartym kodzie źródłowym. Przez wiele lat była niedoścignionym wzorem dla innego darmowego rozwiązania — MySQL. Dziś znajduje zastosowanie wszędzie tam, gdzie wymagana jest najwyższa niezawodność i wydajność, a brak konieczności zapłaty gra kluczową rolę. Stosunek jakości do ceny w przypadku PostgreSQL zmierza do nieskończoności!

Trzymasz w rękach książkę zawierającą liczne przepisy na najlepsze wykorzystanie PostgreSQL. System ten sprawdza się zawsze, gdy chcesz szybko i bezproblemowo osiągnąć zamierzone cele. W trakcie lektury dowiesz się, jak nawiązać połączenie z serwerem, skorzystać z graficznych lub tekstowych narzędzi administracyjnych oraz bezpiecznie zmienić hasło administratora. Ponadto nauczysz się kontrolować przestrzeń dyskową wykorzystywaną przez poszczególne bazy danych, tworzyć tabele, ładować dane oraz zarządzać użytkownikami i ich uprawnieniami. Autorzy dużo miejsca poświęcają kwestii bezpieczeństwa. W końcu dane to najcenniejsza rzecz, jaką przechowuje się w bazach! Każdy z rozdziałów przynosi ogrom wiedzy o różnym poziomie skomplikowania. Zaawansowanych użytkowników zainteresuje rozdział poświęcony replikacji, a tych początkujących rozdział traktujący o uruchamianiu i zatrzymywaniu serwera baz danych. Ta książka przyda się po prostu wszystkim użytkownikom PostgreSQL!

- Zalety PostgreSQL w kontekście innych rozwiązań bazodanowych
- Udostępnianie serwera w sieci
- Zastosowanie narzędzia psql do wykonywania zapytań
- Sprawdzanie wersji serwera
- Lista baz danych na serwerze
- Planowanie nowej bazy danych
- Parametry, ich znaczenie i ustawianie
- Uruchamianie i zatrzymywanie serwera
- Ponowne ładowanie plików konfiguracyjnych
- Przyznawanie użytkownikom własnych baz danych
- Wiele serwerów baz danych w ramach jednego systemu operacyjnego
- Generowanie danych testowych
- Tworzenie kont użytkowników i zarządzanie nimi oraz ich uprawnieniami
- Równoległe wykonywanie zadań — polecenie pg_batch
- Monitorowanie i diagnostyka serwera PostgreSQL
- Przygotowywanie kopii bezpieczeństwa

w katalogu 8182

Księgarnia internetowa
http://helion.pl

Zamówienia telefoniczne:
0 801 339900
0 601 339900

helion.pl
księgarnia internetowa

Sprawdź najnowsze promocje:
http://helion.pl/promocje
Książki najchętniej czytane:
http://helion.pl/bestsellery
Zamów informacje o nowościach:
http://helion.pl/nowosci

Helion

Helion SA
ul. Kościuszki 1c, 44-100 Gliwice
tel.: 32 230 98 63
e-mail: helion@helion.pl
http://helion.pl

Cena: 79.00 zł
ISBN 978-83-246-3061-5

Informatyka w najlepszym wydaniu