



Technologia i rozwiązania

Wysoko wydajny PostgreSQL 9.0

Poznaj najlepsze techniki zwiększania wydajności PostgreSQL i sprawdzone rozwiązania najczęściej spotykanych problemów!

- Jak dobrać komponenty serwera, aby maksymalnie wykorzystać jego możliwości?
- Jak przeprowadzać testy wydajności całego systemu, od sprzętu po aplikację?
- Jak skutecznie indeksować bazę danych i optymalizować zapytania?

Hellen



Gregory Smith

PACKT
PUBLISHING

» Idź do

- Spis treści
- Przykładowy rozdział
- Skorowidz

» Katalog książek

- Katalog online
- Zamów drukowany katalog

» Twój koszyk

- Dodaj do koszyka

» Cennik i informacje

- Zamów informacje o nowościach
- Zamów cennik

» Czytelnia

- Fragmenty książek online

» Kontakt

Helion SA
ul. Kościuszki 1c
44-100 Gliwice
tel. 32 230 98 63
e-mail: helion@helion.pl
© Helion 1991–2011

Wysoko wydajny PostgreSQL 9.0

Autor: Gregory Smith
Tłumaczenie: Robert Górczyński
ISBN: 978-83-246-3062-2
Tytuł oryginału: [PostgreSQL 9.0 High Performance](#)
Format: 170×230, stron: 464



Poznaj najlepsze techniki zwiększania wydajności PostgreSQL i sprawdzone rozwiązania najczęściej spotykanych problemów!

- Jak dobrać komponenty serwera, aby maksymalnie wykorzystać jego możliwości?
- Jak przeprowadzać testy wydajności całego systemu, od sprzętu po aplikację?
- Jak skutecznie indeksować bazę danych i optymalizować zapytania?

Mający za sobą już ponad piętnaście lat rozwoju PostgreSQL jest dziś potężnym systemem baz danych typu open source, o sprawdzonej architekturze i reputacji narzędzia niezawodnego oraz nieprzeciętnie wydajnego. Współdziała on ze wszystkimi popularnymi systemami operacyjnymi i jest w pełni zgodny z warunkami ACID. Te zalety sprawiają, że można go używać jako magazynu danych dla aplikacji oraz jako bazy danych dla aplikacji sieciowych. Jednak osiągnięcie maksymalnej wydajności PostgreSQL nie jest wcale zadaniem łatwym, a w trakcie korzystania z jego serwerów można napotkać powtarzające się trudności, zwłaszcza gdy wzrasta obciążenie serwera, a wymagania stają się coraz większe. Jeśli zatem nie chcesz tygodniami dochodzić do właściwych rozwiązań swoich problemów – oto książka, w której znajdziesz całą potrzebną Ci wiedzę.

Masz w rękach kompletny podręcznik, przeznaczony dla średnio i bardzo zaawansowanych administratorów baz danych, którzy już używają PostgreSQL lub dopiero zamierzają to zrobić. Najpierw zapoznasz się z najnowszymi wersjami tej platformy oraz dowiesz się, jak dobrać komponenty serwera, aby optymalnie wykorzystać możliwości systemu.

Dzięki tej książce:

- Poznasz najlepsze praktyki pozwalające na obsłużenie wymagających aplikacji
- Odkryjesz, dlaczego sprzęt komputerowy nadaje się (lub nie) dla wysoko wydajnych aplikacji bazodanowych
- Zrozumiesz, na czym polegają kompromisy związane z szybkością i niezawodnością działania
- Zoptymalizujesz system operacyjny, aby osiągnąć najlepszą wydajność bazy danych
- Przeprowadzisz testy wydajności całego systemu, od sprzętu komputerowego po aplikację
- Przeanalizujesz rzeczywiste przykłady, co pozwoli Ci poznać wpływ różnych ustawień parametrów serwera na wydajność
- Będziesz skutecznie monitorować zdarzenia zachodzące na serwerze, zarówno w bazie danych, jak i poza nią
- Znajdziesz najlepsze dodatki, rozszerzające podstawowe możliwości bazy danych PostgreSQL
- Dowiesz się, jak przygotować replikację systemów za pomocą najnowszych funkcji wprowadzonych w PostgreSQL 9.0

Zoptymalizuj swój serwer PostgreSQL i unikaj problemów, które mogą zmniejszyć jego wydajność!

Spis treści

O autorze	13
O recenzentach	15
Wprowadzenie	17
Rozdział 1. Wersje PostgreSQL	21
Wydajność we wcześniejszych wydaniach PostgreSQL	22
Wybór odpowiedniej wersji	23
Uaktualnienie do nowszej głównej wersji	23
PostgreSQL czy inna baza danych?	26
Narzędzia PostgreSQL	27
Moduły contrib w PostgreSQL	27
pgFoundry	30
Dodatkowe oprogramowanie związane z PostgreSQL	30
Cykl życiowy aplikacji PostgreSQL	31
Optymalizacja wydajności w praktyce	32
Podsumowanie	34
Rozdział 2. Sprzęt dla bazy danych	35
Zrównoważenie wydatków na zakup sprzętu	35
Procesor	36
Pamięć	37
Dyski twarde	37
Kontrolery dysków	43
Niezawodne kontrolery i konfiguracja dysków	48
Bufor zapisu	49
Wpływ bufora bez wstrzymywania zapisu na wydajność	52
Podsumowanie	53

Rozdział 3. Testy wydajności sprzętu dla bazy danych	55
Testy wydajności procesora i pamięci	55
memtest86+	56
Testowanie pamięci za pomocą narzędzia STREAM	56
Testy wydajności procesora	59
Powody wolnego działania procesora i pamięci	60
Fizyczna wydajność dysku	61
Swobodny dostęp i liczba operacji wejścia-wyjścia na sekundę	61
Dostęp sekwencyjny i ZCAV	63
Liczba wykonywanych operacji zatwierdzenia	64
Narzędzia do testowania wydajności dysku	65
hdtune	65
dd	69
bonnie+ +	70
sysbench	75
Skomplikowane testy wydajności dysku twardego	77
Przykładowe wyniki testu wydajności dysku	78
Oczekiwana wydajność dysku	80
Podsumowanie	83
Rozdział 4. Konfiguracja dysków	85
Maksymalna wielkość systemu plików	85
Odzyskiwanie danych po awarii systemu plików	86
Systemy plików z księgowaniem	87
Systemy plików w Linuksie	88
ext2	88
ext3	89
ext4	91
XFS	91
Inne systemy plików w Linuksie	93
Bariery zapisu	94
Ogólne dostrajanie systemów plików w Linuksie	96
Systemy plików Solaris i FreeBSD	102
Solaris UFS	102
FreeBSD UFS2	104
ZFS	105
Systemy plików w Windows	107
FAT32	107
NTFS	107
Konfiguracja dysku dla PostgreSQL	108
Dowiązania symboliczne	108
Tablespace	109
Drzewo katalogów bazy danych	109
Macierze dyskowe, RAID i konfiguracja dysków	112
Podsumowanie	115

Rozdział 5. Pamięć dla bufora bazy danych	117
Jednostki pamięci w pliku konfiguracyjnym postgresql.conf	118
Zwiększenie parametrów pamięci współdzielonej w systemie Unix w celu zdefiniowania większego bufora	119
Semaforzy jądra	120
Oszacowanie wielkości pamięci współdzielonej	121
Przegląd bufora bazy danych	122
Instalacja pg_buffercache w bazie danych	123
Konfiguracja układu dysków	124
Utworzenie nowego bloku w bazie danych	126
Zapis zmodyfikowanych bloków na dysku	127
Naprawa bazy danych po awarii a wielkość bufora	128
Podstawy przetwarzania punktów kontrolnych	128
Dziennik zapisu z wyprzedzeniem a proces naprawy po awarii	128
Tworzenie punktów kontrolnych	129
Cykl życia bloku bazy danych	131
Bufor bazy danych kontra bufor systemu operacyjnego	132
Podwójnie buforowane dane	133
Przeciążenie punktu kontrolnego	134
Początkowe wskazówki dotyczące wielkości	135
Analiza zawartości bufora	136
Zapytania pozwalające na przegląd zawartości bufora	137
Przegląd wielkości bufora i jej dostosowanie	141
Podsumowanie	141
Rozdział 6. Optymalizacja konfiguracji serwera	143
Interakcja z używaną konfiguracją	144
Ustawienia domyślne i sposoby ich zerowania	144
Dozwolony kontekst do przeprowadzania zmian	144
Ponowne wczytywanie pliku konfiguracyjnego	146
Ustawienia na poziomie serwera	147
Połączenia z bazą danych	147
Pamięć współdzielona	149
Rejestrowanie zdarzeń	150
Polecenie VACUUM i dane statystyczne	152
Punkty kontrolne	155
Ustawienia mechanizmu WAL	156
Replikacja WAL i PITR	159
Ustawienia na poziomie klienta	159
Optymalizacje niezalecane	162
Optymalizacja ustawień nowego serwera	164
Wskazówki dotyczące serwerów dedykowanych	164
Wskazówki dotyczące serwerów współdzielonych	165
pgtune	166
Podsumowanie	166

Rozdział 7. Rutynowa konserwacja	167
Widoczność transakcji wraz z kontrolą współbieżności	167
Wewnętrzne mechanizmy określające widoczność	168
Uaktualnienia	169
Konflikty podczas blokowania rekordów	171
Usunięcie	173
Zalety mechanizmu MVCC	174
Wady mechanizmu MVCC	174
Wyzerowanie identyfikatora transakcji	174
Vacuum	176
Implementacja procesu vacuum	177
Operacja czyszczenia na podstawie kosztów	179
Demon autovacuum	181
Powszechnie spotykane problemy z vacuum i autovacuum	185
Automatyczna analiza	190
Nadmuchane indeksy	191
Pomiar nadmuchania indeksu	191
Szczegółowe monitorowanie stron indeksu i danych	193
Monitorowanie dzienników zdarzeń zapytań	194
Podstawowa konfiguracja rejestracji zdarzeń w PostgreSQL	194
Rejestrowanie trudnych zapytań	199
Analiza pliku dziennika zdarzeń	200
Podsumowanie	207
Rozdział 8. Sprawdzanie wydajności bazy danych	209
Domyślne testy pgbench	209
Definicja tabeli	210
Wykrywanie skali wielkości bazy danych	210
Definicja skryptu zapytania	211
Konfiguracja serwera bazy danych pod kątem pgbench	213
Ręczne uruchamianie pgbench	214
Wyniki graficzne generowane za pomocą pgbench-tools	216
Konfiguracja pgbench-tools	216
Przykładowe wyniki testów pgbench	217
Test przeprowadzający jedynie zapytania SELECT	217
Test transakcji TPC-B-like	218
Analiza opóźnienia	219
Powody otrzymywania błędnych wyników i różnic	222
Programistyczne wersje PostgreSQL	223
Wątki worker i ograniczenia programu pgbench	224
Własne testy pgbench	225
Test szybkości wstawiania danych	225
Testy wydajności Transaction Processing Performance Council	226
Podsumowanie	228

Rozdział 9. Indeksowanie bazy danych	229
Przegląd sposobów indeksowania	230
Dane statystyczne służące do pomiaru wielkości zapytania na dysku i bloku indeksu	230
Uruchomienie przykładu	231
Konfiguracja przykładowych danych	232
Proste wyszukiwania za pomocą indeksów	233
Pełne skanowanie tabeli	234
Tworzenie indeksu	235
Wyszukiwanie za pomocą nieefektywnego indeksu	235
Łączenie indeksów	237
Przejście ze skanowania indeksowanego na sekwencyjne	238
Klastry kontra indeksy	239
Polecenie Explain oraz liczniki bufora	241
Tworzenie indeksu i jego obsługa	241
Zapewnienie unikalności indeksów	242
Współbieżne tworzenie indeksu	243
Klastrowanie indeksu	243
Ponowne indeksowanie	244
Rodzaje indeksów	245
B-tree	245
Hash	246
GIN	246
GiST	247
Zaawansowane sposoby korzystania z indeksów	247
Indeksy wielokolumnowe	248
Indeksy dla operacji sortowania	248
Indeksy częściowe	249
Indeksy bazujące na wyrażeniu	249
Indeksowanie na potrzeby wyszukiwania pełnego tekstu	250
Podsumowanie	250
Rozdział 10. Optymalizacja zapytań	253
Przykładowe zbiory danych	253
Pagila	254
Dell Store 2	254
Podstawy polecenia EXPLAIN	256
Obciążenie związane z pomiarem	256
Zachowanie przy zimnym i rozgrzanym buforze	257
Struktura węzłów planu zapytania	259
Podstawy obliczania kosztu	260
Narzędzia analizy danych polecenia EXPLAIN	262
Graficzne przedstawienie danych EXPLAIN	262
Rozbudowane dane wyjściowe	263
Dane wyjściowe EXPLAIN w postaci czytelnej dla komputera	263
Narzędzia służące do analizy planu	264

Łączenie zbiorów rekordów	265
Identyfikator krotki	265
Skanowanie sekwencyjne	266
Skanowanie indeksu	266
Mapa bitowa i skanowanie indeksu	267
Przetwarzanie węzłów	268
Węzeł Sort	268
Węzeł Limit	270
Węzeł Aggregate	271
Węzeł HashAggregate	272
Węzeł Unique	273
Węzeł Result	274
Węzeł Append	275
Węzeł Group	276
Węzły Subquery Scan i Subplan	277
Operacje ustawiania	278
Materializacja	279
Skanowanie CTE	280
Złączenia	281
Pętle zagnieżdżone	281
Złączenie Merge Join	283
Złączenia Hash Join	285
Dane statystyczne	290
Przeoglądanie i szacowanie za pomocą danych statystycznych	290
Cele danych statystycznych	293
Obszary trudne do oszacowania	295
Inne parametry planowania zapytania	295
effective_cache_size	295
work_mem	297
constraint_exclusion	298
cursor_tuple_fraction	298
Wykonywanie innych typów zapytań	298
Poprawianie zapytań	299
Optymalizacja dla w pełni buforowanych zbiorów danych	300
Poszukiwanie odpowiednika zapytania	300
Wyłączanie funkcji optymalizatora	301
Rozwiązywanie błędów optymalizatora	305
Unikanie planu restrukturyzacji za pomocą OFFSET	306
Zewnętrzne źródła problemów	309
Ograniczenia SQL	309
Numerowanie rekordów w SQL	309
Używanie funkcji Window do numerowania	311
Używanie funkcji Window do kumulowania wyniku	311
Podsumowanie	313

Rozdział 11. Dane statystyczne i działanie bazy danych	315
Widoki danych statystycznych	315
Widoki kumulacyjne i żywe	317
Dane statystyczne tabel	318
Operacje wejścia-wyjścia tabel	320
Dane statystyczne indeksu	322
Operacje wejścia-wyjścia indeksu	323
Dane statystyczne dotyczące całej bazy danych	324
Połączenia i aktywność	324
Blokady	325
Transakcje wirtualne	326
Dekodowanie informacji o blokadzie	327
Oczekiwanie na blokadę transakcji	330
Oczekiwanie na blokadę tabeli	331
Rejestrowanie informacji o blokadach	332
Wykorzystanie dysku	333
Bufor, zapis w tle oraz aktywność tworzenia punktu kontrolnego	335
Zapis migawek pg_stat_bgwriter	337
Optymalizacja z użyciem danych statystycznych dotyczących zapisu w tle	339
Podsumowanie	341
Rozdział 12. Monitorowanie i trendy	343
Narzędzia monitorujące w systemie Unix	343
Przykładowa konfiguracja	344
vmstat	344
iostat	347
top	355
sysstat i sar	357
Narzędzia monitorujące dla Windows	360
Menedżer zadań	360
Monitor systemu Windows	360
Oprogramowanie trendów	362
Rodzaje monitorowania i oprogramowanie trendów	363
Nagios	364
Cacti	366
Munin	366
Inne pakiety trendów	367
Podsumowanie	369
Rozdział 13. Pula połączeń i buforowanie	371
Pula połączeń	371
Liczniki puli połączeń	372
pgpool-II	373
pgBouncer	374

Buforowanie bazy danych	376
memcached	376
pgmemcache	377
Podsumowanie	378
Rozdział 14. Skalowanie za pomocą replikacji	381
Hot Standby	381
Terminologia	382
Konfiguracja przekazywania danych mechanizmu WAL	383
Protokół Streaming Replication	384
Optymalizacja funkcji Hot Standby	384
Menedżery kolejki replikacji	386
Slony	387
Londiste	387
Skalowanie odczytu za pomocą oprogramowania replikacji bazującego na kolejce	388
Wymagania aplikacji specjalnych	388
Bucardo	388
pgpool-II	389
Inne interesujące projekty replikacji	389
Podsumowanie	391
Rozdział 15. Partycjonowanie danych	393
Partycjonowanie o zasięgu tabeli	393
Określenie pola klucza używanego do partycjonowania	394
Wielkości partycji	395
Tworzenie partycji	396
Przekierowywanie poleceń INSERT do partycji	397
Plany zapytań dla pustej partycji	399
Zmiana daty i uaktualnianie wyzwalacza	400
Migracja partycjonowanej używanej tabeli	401
Zapytania partycjonowane	403
Tworzenie nowych partycji	405
Zalety partycjonowania	406
Błędy często popełniane podczas partycjonowania	407
Partycjonowanie poziome za pomocą PL/Proxy	408
Generowanie wartości hash	408
Skalowanie za pomocą PL/Proxy	410
Skalowanie za pomocą GridSQL	411
Podsumowanie	412
Rozdział 16. Unikanie najczęściej spotykanych problemów	415
Operacja bulk-loading	415
Metody wczytywania danych	416
Optymalizacja operacji bulk-loading	417
Pominięcie optymalizacji mechanizmu WAL	418

Ponowne utworzenie indeksów i dodanie ograniczeń	419
Przywracanie równoległe	419
Czyszczenie po operacji wczytania danych	420
Najczęstsze problemy związane z wydajnością	421
Zliczanie rekordów	421
Niewyjaśnione operacje zapisu	422
Wolne wykonywanie funkcji i poleceń składowanych	423
Testy wydajności PL/pgSQL	424
Ogromne przeciążenie klucza zewnętrznego	424
Użycie pamięci przez wyzwalacz	425
Ogromne przeciążenie mechanizmu zbierającego dane statystyczne	426
Zmaterializowane widoki	427
Profilowanie bazy danych	427
gprof	427
OProfile	428
Visual Studio	428
DTrace	428
Problemy wydajności w poszczególnych wersjach	429
Agresywne uaktualnienia PostgreSQL	430
8.1	431
8.2	432
8.3	432
8.4	434
9.0	436
Podsumowanie	441
Skorowidz	443

Optymalizacja konfiguracji serwera

Podstawowe ustawienia optymalizacyjne PostgreSQL znajdują się w zwykłym pliku tekstowym o nazwie *postgresql.conf* umieszczonym w strukturze katalogów bazy danych. Struktura katalogów w systemach z rodziny Unix często określana jest przy użyciu zmiennej środowiskowej `$PGDATA`, a więc ścieżka dostępu do pliku konfiguracyjnego na tych platformach ma postać `$PGDATA/postgresql.conf`.

W rozdziale przedstawiono omówienie parametrów konfiguracyjnych; praktycznie jest to powielony ogólny format stosowany w dokumentacji dostępnej na stronie <http://www.postgresql.org/docs/current/static/runtime-config.html>. Jednak tutaj, zamiast opisywania znaczenia każdego parametru, nacisk został położony na udzielenie wskazówek dotyczących ustawiania najważniejszych wartości z perspektywy użytkownika zainteresowanego optymalizacją wydajności. Rozdział należy więc potraktować jako uzupełnienie materiału umieszczonego w oficjalnej dokumentacji, a nie jej zamiennik.

Inne źródło informacji na omawiany tutaj temat znajduje się w artykule „Tuning Your PostgreSQL Server” dostępnym na stronie http://wiki.postgresql.org/wiki/Tuning_Your_PostgreSQL_Server, w którym pewne informacje są identyczne z przedstawionymi w rozdziale. Struktura artykułu wiki powoduje, że jest on aktualizowany na bieżąco i może zawierać pewne informacje dotyczące przyszłych wersji bazy danych PostgreSQL, który były niedostępne w trakcie przygotowywania tego rozdziału.

Interakcja z używaną konfiguracją

Istnieje wiele sposobów modyfikacji parametrów bazy danych, nie są to jedynie edycja pliku konfiguracyjnego i ponowne uruchomienie serwera. Poznanie i zrozumienie tych sposobów ma znaczenie krytyczne zarówno dla skrócenia czasu niedostępności serwera jedynie z powodu rutynowych zmian konfiguracyjnych, jak i zagwarantowania, że czytelnik przeprowadza modyfikację właściwych parametrów, gdy zmiana ma zostać wprowadzona natychmiast.

Ustawienia domyślne i sposoby ich zerowania

Baza danych ma dwa rodzaje ustawień, które można określić jako „ustawienia domyślne” w zależności od kontekstu. Jednym z nich są ustawienia domyślne powodujące ustalenie przez serwer pewnych wartości, jeżeli użytkownik nie zmieni żadnych opcji — są to ustawienia, z którymi serwer jest uruchamiany jeszcze przed odczytaniem pliku konfiguracyjnego *postgresql.conf*. Od wydania PostgreSQL 8.4 wartości te można sprawdzić w kolumnie *boot_val* w widoku *pg_settings*. Więcej informacji na ten temat znajduje się na stronie <http://www.postgresql.org/docs/current/static/view-pg-settings.html>.

Po uruchomieniu serwera i wprowadzeniu zmian w parametrach, które również posiadają swoje wartości domyślne, powrót do wspomnianych wartości domyślnych następuje w wyniku wydania polecenia *RESET*. Działanie tego polecenia zostało omówione na stronie <http://www.postgresql.org/docs/current/static/sql-reset.html>. Te wartości domyślne można przejrzeć w kolumnie *reset_val* znajdującej się w widoku *pg_settings*.

Dozwolony kontekst do przeprowadzania zmian

Każde ustawienie konfiguracyjne ma powiązany z nim kontekst, w ramach którego może zostać zmienione. Najlepszym sposobem określenia dozwolonego kontekstu do przeprowadzania zmian jest bezpośrednie zapytanie bazy danych. W przedstawionym poniżej przykładzie pokazano po jednym wpisie dla każdego rodzaju kontekstu (w rzeczywistości po wydaniu poniższego polecenia dane wyjściowe będą zawierały wszystkie parametry serwera):

```
postgres=# select name,context from pg_settings;
          name          | context
-----+-----
archive_command        | sighup
archive_mode           | postmaster
block_size             | internal
log_connections        | backend
log_min_duration_statement | superuser
search_path            | user
```

W oficjalnym podręczniku użytkownika pole kontekstu (context) nie jest zbyt dobrze udokumentowane. Poniżej przedstawiono więc znaczenie różnych stosowanych ustawień w kolejności od najtrudniejszego do najłatwiejszego do zmiany.

- **internal**. Ustawienia te w dużej mierze są wewnętrznymi ustawieniami bazy danych określonymi w trakcie jej kompilacji. Wyświetlane są w celu dostarczenia użytkownikowi informacji, ale nie mogą zostać zmienione bez ponownej kompilacji serwera.
- **postmaster**. Ustawienia są uaktualniane jedynie podczas pełnego uruchamiania serwera. Do tej kategorii zaliczają się wszystkie opcje dotyczące pamięci współdzielonej.
- **sigchup**. Wysłanie sygnału HUP do serwera spowoduje ponowne odczytanie pliku konfiguracyjnego *postgresql.conf*, a wszelkie zmiany wprowadzone do tego parametru będą natychmiast zastosowane. Więcej informacji na ten temat znajduje się w kolejnym punkcie zatytułowanym „Ponowne wczytywanie pliku konfiguracyjnego”.
- **backend**. Ustawienia oznaczone za pomocą tej właściwości są podobne do ustawień **sigchup**, z wyjątkiem faktu, że wprowadzone zmiany nie mają wpływu na już istniejące sesje bazy danych. Tylko sesje uruchomione po wprowadzeniu zmian będą stosowały nowe ustawienia. Istnieje niewiele parametrów oznaczonych omawianą właściwością, większość z nich wpływa jedynie na działania podejmowane w trakcie uruchamiania i zamykania sesji. Ostatnia opcja tej grupy (`log_connections`) nie może działać wstecz, to znaczy nie ma możliwości rozpoczęcia rejestrowania zdarzeń już nawiązanego połączenia. Zdarzenia będą rejestrowane tylko dla nowych połączeń ustanowionych po włączeniu opcji `log_connections`.
- **superuser**. Ustawienia tej grupy mogą być modyfikowane w dowolnym momencie przez każdego superużytkownika bazy danych (z reguły użytkownika, który utworzył bazę danych, czyli bardzo często „*postgres*”). Aktywowanie zmiany nie wymaga pełnego, ponownego wczytania pliku konfiguracyjnego. Większość ustawień tej grupy jest powiązana z rejestrowaniem w plikach dzienników zdarzeń różnych aspektów poleceń wykonywanych przez serwer.
- **user**. Te parametry poszczególni użytkownicy mogą dostosować w dowolnym momencie swojej sesji. Wprowadzone zmiany dotyczą jedynie danej sesji. Większość parametrów zmienia sposób wykonywania zapytań, co pozwala na przeprowadzenie optymalizacji w ramach sesji.

Na podstawie powyższej sesji można się przekonać, że udzielenie odpowiedzi na — wydałoby się — proste pytanie, jaka jest bieżąca wartość `work_mem`, może być bardzo trudne, w zależności od wybranego kontekstu. Początkowo wartość ta może być wartością określoną w pliku konfiguracyjnym *postgresql.conf*, następnie może być zmieniona w wyniku ponownego wczytania konfiguracji, a na końcu znów zmieniona przez użytkownika przed wykonaniem zapytania.

Ponowne wczytywanie pliku konfiguracyjnego

Istnieją trzy sposoby pozwalające bazie danych na ponowne wczytanie konfiguracji w celu uaktualnienia wartości zaliczających się do grupy `sig_hup`. Jeżeli czytelnik jest połączony z bazą danych jako superużytkownik, można użyć funkcji `pg_reload_conf()` w następujący sposób:

```
postgres=# SELECT pg_reload_conf();
pg_reload_conf
-----
t
```

Sygnal `SIGHUP` można wysłać również ręcznie za pomocą polecenia `kill` systemu Unix:

```
$ ps -eaf | grep "postgres -D"
postgres 11185      1 0 22:21 pts/0      00:00:00 /home/postgres/inst/bin/
↳postgres -D /home/postgres/data/
$ kill -HUP 11185
```

Wreszcie, sygnał `SIGHUP` można wysyłać do serwera przy użyciu narzędzia `pg_ctl`:

```
$ pg_ctl reload
server signaled
```

Niezależnie od użytego sposobu, po wysłaniu sygnału `SIGHUP` w dziennikach zdarzeń bazy danych znajdzie się następujący komunikat:

```
LOG: received SIGHUP, reloading configuration files
```

Następnie, za pomocą poleceń, takich jak `SHOW`, lub po przejrzaniu widoku `pg_settings` można potwierdzić, że zmiany zostały wprowadzone zgodnie z oczekiwaniami.

Umieszczanie ustawień w komentarzach

Co się stanie w sytuacji, gdy pewien parametr został ręcznie ustawiony, ale musi być wyłączony w działającym serwerze? Konkretna odpowiedź zależy od wersji używanego serwera. Przyjmijmy założenie, że plik `postgresql.conf` został uruchomiony wraz z ustawionym następującym parametrem:

```
checkpoint_segments = 30
```

W celu wyłączenia tego parametru trzeba wyedytować plik konfiguracyjny, a parametr umieścić w komentarzu:

```
#checkpoint_segments = 30
```

Następnie konieczne jest nakazanie serwerowi, aby ponownie wczytał plik konfiguracyjny:

```
$ pg_ctl reload
```

Parametr `checkpoint_segments` jest ustawieniem kontekstu `sig_hup`. Od PostgreSQL 8.3 powyższa zmiana spowoduje powrót do wartości domyślnej serwera (`boot_val`). W wersji 9.0 w dzienniku zdarzeń zostanie ponadto umieszczony następujący komunikat:

```
LOG: received SIGHUP, reloading configuration files
LOG: parameter "checkpoint_segments" removed from configuration file, reset
↳to default
```

Użycie wartości domyślnej można potwierdzić za pomocą polecenia `SHOW`:

```
$ psql -x -c "show checkpoint_segments"
-[ RECORD 1 ]-----+--
checkpoint_segments | 3
```

Jeżeli używany serwer to PostgreSQL w wersji 8.2 lub wcześniejszej, wprowadzona zmiana nie spowoduje żadnego efektu, to znaczy parametr `checkpoint_segments` nadal będzie miał wartość 30. Dopiero po pełnym, ponownym uruchomieniu serwera nastąpi przywrócenie wartości domyślnej, czyli 3.

Ponieważ takie zachowanie jest skomplikowane i uzależnione od używanej wersji serwera, doświadczeni administratorzy PostgreSQL zwykle dwukrotnie sprawdzają parametry, które mają zamiar zmodyfikować. Do sprawdzenia używają polecenia `SHOW` lub widoku `pg_settings` i upewniają się w ten sposób, że parametr ma oczekiwaną wartość.

W tym miejscu inną, skomplikowaną kwestią jest możliwość dołączenia dodatkowych plików konfiguracyjnych z poziomu głównego pliku `postgresql.conf`. W przypadku takiego rozwiązania skutek jest taki sam jak wstawienie zawartości określonego pliku we wskazanym miejscu. W widoku `pg_settings` można poznać nazwę pliku, z którego pochodzi dany parametr, wraz z numerem wiersza zawierającego aktywną wersję. Warto też pamiętać, że w przypadku wielokrotnego ustawiania tego samego parametru pod uwagę zawsze brane jest tylko ostatnie jego wystąpienie.

Ustawienia na poziomie serwera

Wprawdzie w pewnych przypadkach parametry mogą być modyfikowane w innych kontekstach, przedstawione w tym podrozdziale są tymi, które można zmodyfikować w pliku `postgresql.conf` odczytywanym przed uruchomieniem serwera.

Połączenia z bazą danych

Istnieje wiele parametrów konfiguracyjnych określających sposoby zdalnego i lokalnego nawiązywania połączenia z bazą danych. Pełna lista wspomnianych parametrów została przedstawiona na stronie <http://www.postgresql.org/docs/current/static/runtime-config-connection.html>.

listen_addresses

W każdej instalacji wymagającej obsługi połączeń pochodzących z systemów zdalnych konieczna jest modyfikacja parametru `listen_addresses` pozwalającego na obsługę takich połączeń. Domyślnie dozwolone są jedynie połączenia lokalne pochodzące od użytkownika zalogowanego do tego samego systemu, w którym znajduje się baza danych. Powszechnie stosowanym podejściem jest akceptacja połączeń przychodzących z każdego miejsca. W tym celu w głównym pliku konfiguracyjnym trzeba umieścić poniższy wiersz:

```
listen_addresses = '*'
```

Aby kontrolować, kto może nawiązać połączenie, trzeba przeprowadzić konfigurację pliku `pg_hba.conf`, co zostało omówione na stronie <http://www.postgresql.org/docs/current/static/auth-pg-hba-conf.html>. Stosowanie takiego podejścia wiąże się z pewnym problemem dotyczącym wydajności. Filtrowanie połączeń za pomocą dokładniej skonfigurowanego parametru `listen_addresses` może być znacznie bardziej efektywne niż pozwolenie na nawiązanie połączenia wszystkim klientom. Zezwolenie klientowi na połączenie, a następnie jego odrzucenie w wyniku wpisu znajdującego się w pliku konfiguracyjnym `pg_hba.conf` powoduje niepotrzebne zużycie pewnych zasobów serwera i naraża system na niebezpieczeństwo przeprowadzenia w ten sposób ataku typu odmowa usługi — DoS (ang. *Denial Of Service*) — przez złośliwego użytkownika.

W praktyce tylko niewielka liczba serwerów PostgreSQL pozwala na bezpośrednie przyjmowanie zapytań pochodzących z internetu. Normalnie są one filtrowane przez port domyślny PostgreSQL (5432) na poziomie zapor sieciowej, co stanowi najefektywniejsze podejście i jest często stosowaną implementacją mechanizmu współdzielonej ochrony także innych aplikacji. W systemie podłączonym do internetu, takim jak na przykład klastry zawierające bazy danych dla usług „w chmurach”, należy upewnić się o stosowaniu wszystkich trzech warstw ochrony. Na poziomie zapor sieciowej warto określić, kto może nawiązać połączenie z serwerem, dodatkowo zmniejszyć listę nasłuchiwanego adresów za pomocą opcji `listen_addresses`, a także ograniczyć użytkownikom dostęp do bazy danych za pomocą pliku konfiguracyjnego `pg_hba.conf`.

max_connections

Jednym z parametrów, dla którego czytelnik najczęściej będzie ustawiał wartość (z reguły 100) w pliku konfiguracyjnym `postgresql.conf` wygenerowanym przez `initdb`, jest `max_connections`. Ponieważ, jak wspomniano w poprzednim rozdziale, każde połączenie wykorzystuje niewielką ilość pamięci współdzielonej, w systemach domyślnie używających niewielkiej ilości pamięci współdzielonej nawiązanie większej ilości połączeń nie zostanie nawet dozwolone. W związku z powyższym, podobnie jak w przypadku parametru `shared_buffers`, po utworzeniu klastry bazy danych i określeniu w domyślnym pliku konfiguracyjnym największej dopuszczalnej wartości (do 100) przeprowadzane są pewne badania. W praktyce ilość pamięci niewspółdzielonej używanej przez każdego klienta podczas przeprowadzania operacji, takich jak sortowanie, jest wprawdzie znacznie większa, ale ilości wykorzystywanej pamięci współdzielonej nie można pominąć.

Ważne jest, aby w tym parametrze nie ustawiać wartości wyższej niż konieczna. Istnieje wiele wad wynikających z ustawienia wyższej wartości parametru `max_connections`. Jedną z nich to marnowanie pamięci współdzielonej, ale to problem, którym należy się najmniej przejmować, ponieważ ilość pamięci współdzielonej wykorzystywanej przez każde połączenie pozostaje niewielka.

Jednak są inne zasoby wykorzystywane przez klienta, na przykład alokacja pamięci dla operacji sortowania (kontrolowana za pomocą parametru `work_mem` omówionego dalej, w tym rozdziale), która z reguły obejmuje ogromny blok pamięci. Jeżeli dozwolona będzie obsługa większej liczby połączeń, to by bezpiecznie oszczędzać pamięć, trzeba również zmniejszyć wartości wspomnianych ustawień, tak aby zminimalizować niebezpieczeństwo alokacji większej ilości pamięci niż dostępna.

Ze względu na problemy związane z alokacją zasobów, serwery PostgreSQL w Windows mogą mieć bardzo ograniczoną liczbę obsługiwanych połączeń. Bardzo często zdarza się, że zanim wyczerpana zostanie pamięć w obszarze Desktop Heap, obsługiwanych będzie jedynie około 125 połączeń. Więcej informacji na temat tego problemu oraz potencjalne sposoby jego rozwiązania można znaleźć na stronie http://wiki.postgresql.org/wiki/Running_&_Installing_PostgreSQL_On_Native_Windows.

Wreszcie, nawiązywanie połączeń w bazie danych PostgreSQL należy uważać za operację intensywnie wykorzystującą zasoby. Celem bazy danych nie jest działalność w charakterze komponentu nawiązującego połączenie z bazą danych, przeprowadzającego uwierzytelnianie i dotarcie do punktu, w którym zapytanie będzie wykonane jako stosunkowo niewielka operacja. Ogólnie rzecz ujmując, obciążenie związane z nawiązaniem kilkuset połączeń staje się wąskim gardłem podczas działania serwera. Dokładna liczba połączeń, po przekroczeniu której stają się one obciążeniem, zależy od używanego sprzętu i konfiguracji serwera. Jeżeli czytelnik ma zamiar obsługiwać jednocześnie tysiące zapytań, nie może zastosować podejścia, w którym każdy klient bezpośrednio nawiązuje połączenie z bazą danych. W takim przypadku najczęściej stosowanym rozwiązaniem problemu skalowalności jest użycie oprogramowania obsługującego pulę połączeń między aplikacją i bazą danych. Temat ten został omówiony w rozdziale 13.

Pamięć współdzielona

Właściwe ustawienie wartości parametrów związanych z pamięcią współdzieloną jest ważne, ponieważ ich zmiana zawsze wymaga pełnego, ponownego uruchomienia serwera bazy danych — serwer nie ma możliwości dynamicznej ponownej alokacji pamięci współdzielonej.

`shared_buffers`

Parametr `shared_buffers` był szczegółowo omawiany w poprzednim rozdziale.

Ustawienia Free Space Map (FSM)

Przestrzeń pozostała w wyniku operacji usuwania bądź modyfikacji danych jest umieszczana przez polecenie `VACUUM` w przestrzeni FSM (ang. *Free Space Map*). Nowe operacje alokacji wykorzystują przestrzeń pochodzącą z FSM, zamiast alokować nową przestrzeń na dysku.

Od wydania PostgreSQL 8.4 przestrzeń FSM jest przechowywana na dysku, a tym samym automatycznie skaluje swoją wielkość. W PostgreSQL do wersji 8.3 przestrzeń FSM była przechowywana w pamięci współdzielonej, co wymagało dokładnego monitorowania ilości tej pamięci i potencjalnie prowadziło do zwiększenia jej zużycia. Upewnienie się, że wartości parametrów `max_fsm_pages` i `max_fsm_relations` w pliku konfiguracyjnym są wystarczające, powinno być częścią regularnych operacji konserwacyjnych w tych wersjach serwera PostgreSQL. Operację tę można przeprowadzić ręcznie bądź wydać polecenie `VACUUM VERBOSE` przeprowadzające pomiar bieżącego użycia pamięci współdzielonej w bardziej zautomatyzowany sposób. Więcej informacji na ten temat przedstawiono w rozdziałach 5. i 7.

Rejestrowanie zdarzeń

Ogólne ustawienia dotyczące rejestracji zdarzeń są ważne, ale w pewnym sensie pozostają poza zakresem tematycznym tej książki. Czytelnik może być zmuszony do ustawienia parametrów, takich jak `log_destination`, `log_directory` i `log_filename`, w sposób zgodny z systemem oraz wymaganiami administratora używanego środowiska. Parametry te mają domyślnie ustawione rozsądne wartości pozwalające na rozpoczęcie pracy w większości systemów. W rozdziale 7. zostanie poruszony problem dostosowania omawianych parametrów w celu przeprowadzenia rejestracji zdarzeń w plikach CSV, co może być użyteczne podczas pomiaru czasu wykonywania zapytań.

W większości systemów Unix bardzo często zdarza się, że opcje dotyczące rejestracji zdarzeń bazy danych są ustawiane w skryptach uruchamiających i zatrzymujących serwer, a nie bezpośrednio w pliku konfiguracyjnym `postgresql.conf`. Jeżeli w celu ręcznego uruchomienia serwera stosowane jest polecenie `pg_ctl`, czytelnik może odkryć, że rejestracja zdarzeń następuje poprzez ich bezpośrednie wyświetlenie na ekranie. W takim przypadku, aby dowiedzieć się, o co chodzi, trzeba spojrzeć do skryptu uruchamiającego serwer w zwykły sposób (najczęściej `/etc/init.d/postgresql`). W większości przypadków do polecenia `pg_ctl` wystarczy dodać opcję `-l nazwa_pliku`, która powoduje przekierowanie danych wyjściowych do standardowego położenia.

log_line_prefix

Wartość domyślna opcji `log_line_prefix` jest pusta, a to zdecydowanie niepożądane. Wartość dobra na początek będzie miała następującą postać:

```
log_line_prefix='%t:%r:%u@d:[%p]: '
```

Powyższy parametr powoduje umieszczenie w każdym wierszu dziennika zdarzeń takich informacji jak:

- %t: znacznik czasu;
- %u: nazwa użytkownika bazy danych;
- %r: nazwa zdalnego komputera, z którego nawiązano połączenie;
- %d: nazwa bazy danych, z którą nawiązano połączenie;
- %p: identyfikator procesu połączenia.

Na początku może nie być takie oczywiste, po co zostały zastosowane powyższe wartości domyślne, a zwłaszcza identyfikator procesu. Jednak po próbie rozwiązania kilku problemów związanych z wydajnością konieczność zapisania w pliku dziennika zdarzeń wymienionych informacji stanie się bardziej oczywista i czytelnik będzie zadowolony z faktu posiadania tych informacji.

Inne podejście warte rozważenia to ustawienie wartości parametru `log_line_prefix` w taki sposób, aby dzienniki zdarzeń były zgodne z programem `pgFouine`, omówionym w rozdziale 7. Jest to rozsądny, przeznaczony do ogólnego rejestrowania zdarzeń prefiks, a wiele witryn i tak ostatecznie stosuje pewne mechanizmy analizy zapytań.

log_statement

Oto opcje do zastosowania w tym parametrze:

- none. Nie są rejestrowane żadne informacje na poziomie poleceń.
- ddl. Rejestrowane są jedynie polecenia **DDL** (ang. *Data Definition Language*), na przykład `CREATE` lub `DROP`. Takie ustawienie można zastosować nawet w systemie produkcyjnym i jest przydatne podczas wychwytywania najważniejszych zmian przypadkowo lub celowo wprowadzonych przez administratorów.
- mod. Rejestrowane są wszystkie polecenia modyfikujące wartość, czyli praktycznie wszystkie, poza prostymi poleceniami `SELECT`. Jeżeli obciążenie w serwerze to przede wszystkim polecenia `SELECT` przeprowadzające niewielką ilość zmian w danych, praktycznym rozwiązaniem może być pozostawienie tej opcji włączonej przez cały czas.
- all. Rejestrowane są wszystkie polecenia. Ogólnie rzecz ujmując, pozostawienie tej opcji włączonej w serwerze produkcyjnym jest niepraktyczne, ze względu na obciążenie powodowane przez operację rejestrowania zdarzeń. Gdy jednak serwer jest na tyle potężny, aby poradzić sobie z takim obciążeniem, pozostawienie tej opcji włączonej przez cały czas może okazać się pomocne.

Rejestrowanie poleceń to użyteczna technika pozwalająca na wyszukiwanie problemów związanych z wydajnością. Analiza informacji zebranych po ustawieniu parametru `log_statement` i powiązanych źródeł informacji szczegółowych na poziomie poleceń może ukazać prawdziwą przyczynę wielu rodzajów problemów wydajności. Zebrane w ten sposób informacje można połączyć z odpowiednimi narzędziami analizy, które omówiono w rozdziale 7.

log_min_duration_statement

Gdy znamy już ilość czasu wymaganą do wykonania typowego zapytania, użycie parametru `log_min_duration_statement` pozwoli na rejestrowanie tylko tych poleceń, których wykonanie przekroczy ustalony czas. Wartość jest podawana w milisekundach, tak więc parametr można ustawić w następujący sposób:

```
log_min_duration_statement=1000
```

Powyższe polecenie powoduje rejestrację w dzienniku zdarzeń tylko poleceń wykonywanych dłużej niż sekundę. Parametr może być użyteczny do wyszukiwania źródeł „odstających” poleceń, których wykonanie trwa znacznie dłużej niż pozostałych.

Jeżeli używany jest PostgreSQL w wersji 8.4 lub nowszej, zamiast tej funkcji preferowanym rozwiązaniem może być moduł `auto_explain`. Więcej informacji na temat modułu można znaleźć na stronie <http://www.postgresql.org/docs/8.4/static/auto-explain.html>. Moduł umożliwia sprawdzenie operacji wykonywanych przez powolne zapytania po przejrzaniu powiązanych z nimi planów EXPLAIN.

Polecenie VACUUM i dane statystyczne

Baza danych PostgreSQL wymaga dwóch podstawowych form regularnej obsługi podczas dodawania, uaktualniania i usuwania danych.

Polecenie `VACUUM` powoduje usunięcie śmieci po starych transakcjach, łącznie z usunięciem informacji, które nie są dłużej widoczne, oraz zwrócenie zwolnionego miejsca, tak aby mogło być ponownie wykorzystane. Im więcej w bazie danych wykonywanych poleceń `UPDATE` i `DELETE`, tym częściej trzeba przeprowadzać operację usuwania śmieci. Nawet w statycznych tabelach, których zawartość nigdy nie ulega zmianie, od czasu do czasu trzeba wykonać operację usuwania śmieci.

Polecenie `ANALYZE` przegląda tabele w bazie danych i zbiera na ich temat dane statystyczne — informacje, takie jak liczba posiadanych rekordów bądź liczba odmiennych wartości w tabelach. Wiele aspektów planowania zapytania zależy od prawidłowego zebrania tych danych.

Więcej informacji na temat polecenia `VACUUM` znajduje się w rozdziale 7., natomiast temat stosowania danych statystycznych poruszono w rozdziale 10.

Demon autovacuum

Ponieważ zadanie usuwania śmieci ma w dłuższej perspektywie znaczenie krytyczne dla bazy danych, od wydania PostgreSQL 8.1 w serwerze umieszczono demon `autovacuum`, który działa w tle i zajmuje się wspomnianym zadaniem. Aktywacja demona następuje po wykonaniu w bazie danych liczby zmian przekraczającej wartość obliczoną na podstawie wielkości tabeli.

Parametr dla demona autovacuum jest włączony domyślnie w PostgreSQL 8.3, a wartość domyślna jest ustalona w sposób wystarczający do działania mniejszej bazy danych i wymaga jedynie niewielkiej ręcznej modyfikacji. Ogólnie rzecz biorąc, trzeba uważać, aby ilość danych w obszarze FSM nie przekraczała wartości ustalonej przez parametr `max_fsm_pages`. Od PostgreSQL 8.4 opisane wymaganie nie stanowi już powodu do zmartwień dla użytkownika.

Włączanie demona autovacuum w starszych wersjach

Jeżeli demon autovacuum jest dostępny, ale nie jest włączony domyślnie, jak ma to miejsce w bazach danych PostgreSQL 8.1 i 8.2, istnieje kilka powiązanych z nim parametrów, które muszą być włączone w celu zapewnienia jego prawidłowej pracy. Więcej informacji na ten temat przedstawiono na stronach <http://www.postgresql.org/docs/8.1/interactive/maintenance.html> i <http://www.postgresql.org/docs/8.2/interactive/routine-vacuuming.html>.

W wymienionych wersjach w pliku konfiguracyjnym *postgresql.conf* trzeba ustawić trzy parametry:

```
stats_start_collector=true
stats_row_level=true
autovacuum=on
```

Warto zwrócić uwagę, że — zgodnie z ostrzeżeniem przedstawionym w dokumentacji — zalecane jest również użycie parametru `superuser_reserved_connections` zezwalającego na działanie demonów autovacuum w wymienionych wersjach bazy danych.

Demon autovacuum dostępny w wersjach 8.1 i 8.2 nie jest aż tak wydajny jak dostarczany w wersjach 8.3 i nowszych. Uzyskanie odpowiedniego balansu zapewniającego prawidłową obsługę bazy danych bez zbyt dużego obciążenia serwera będzie wymagało pewnych eksperymentów i dopasowywania wartości. Ponieważ jest to tylko jeden proces, pozostawienie go, by działał w tle, gdy serwer jest zajęty, będzie mniejszym obciążeniem dla serwera. Zagadnienie to nie będzie tutaj obszernie omawiane. Ogólnie rzecz biorąc, znacznie lepszym rozwiązaniem jest poświęcenie czasu na uaktualnienie bazy danych PostgreSQL do wydania zawierającego nowszą wersję demona autovacuum niż próba dostosowania starszej wersji. Szczególnie dotyczy to sytuacji, kiedy równocześnie występują inne problemy z wydajnością, których nie można tak łatwo rozwiązać w starszej wersji serwera.

maintenance_work_mem

Kilka operacji w serwerze bazy danych wymaga pamięci roboczej dla operacji większych niż zwykle sortowanie. Polecenia `VACUUM`, `CREATE INDEX` oraz `ALTER TABLE ADD FOREIGN KEY` mogą alokować maksymalną ilość pamięci określoną przez parametr `maintenance_work_mem`. Jest mało prawdopodobne, aby wiele sesji przeprowadzało jednocześnie tego rodzaju operacje. Dla tego parametru można określić wartość znacznie wyższą niż w standardowej opcji `work_mem` ustawianej dla każdego klienta. Warto pamiętać, że przynajmniej parametr `autovacuum_max_workers`

(dla którego od wersji 8.3 wartość domyślna wynosi 3) zaalokuje tę ilość pamięci, dlatego też podczas określania tej wartości należy uważać na takie sesje (prawdopodobnie z jedną lub dwiema sesjami wykonującymi operację CREATE INDEX).

Przy założeniu, że liczba demonów autovacuum nie została zwiększona, typowe wysokie ustawienie omawianej wartości w nowoczesnym serwerze będzie miało wartość 5% całkowitej ilości pamięci RAM. Tak więc nawet pięć procesów nie wykorzysta więcej niż jedną czwartą dostępnej pamięci RAM. Oznacza to, że `maintenance_work_mem` wykorzystuje około 50 MB pamięci na każdy dostępny gigabajt pamięci RAM w serwerze.

default_statistics_target

PostgreSQL podejmuje decyzje dotyczące sposobu wykonywania zapytań na podstawie danych statystycznych zebranych dla każdej tabeli i przechowywanych w bazie danych. Tego rodzaju informacje są zbierane podczas analizy tabel za pomocą polecenia ANALYZE lub demona autovacuum. W każdym przypadku ilość informacji zbierana w trakcie analizy jest określana przez parametr `default_statistics_target`. Zwiększenie wartości tego parametru wydłuża analizę. Analiza regularnie przeprowadzana przez demon autovacuum w tle stanowi coraz większe obciążenie podczas konserwacji bazy danych. Jednak brak odpowiednich danych statystycznych dotyczących tabeli skutkuje ryzykiem użycia niewłaściwego planu wykonywania zapytań względem tej tabeli.

Wartość domyślna wymienionej opcji z reguły jest bardzo niska (to znaczy 10), ale w PostgreSQL 8.4 została zwiększona do 100. Stosowanie większej wartości było popularne także w starszych wersjach bazy danych w celu ogólnego poprawienia zachowania zapytań. Indeksy korzystające z operatora LIKE działają znacznie lepiej, gdy opisany parametr ma wartość wyższą niż 100, a nie niższą, co wiąże się ze zdefiniowaną na stałe zmianą sposobu ich działania po użyciu wartości większej niż 100 dla tego parametru.

Warto zwrócić uwagę, że zwiększenie wartości parametru `default_statistics_target` skutkuje ogólnym spowolnieniem systemu, jeżeli w ogóle nie są wykonywane zapytania, w których dodatkowe dane statystyczne powodują wybór lepszego planu wykonywania. Z tego powodu pewne proste testy wydajności pokazują, że baza danych PostgreSQL w wersji 8.4 jest nieco wolniejsza od 8.3, jeżeli obie stosują wartości domyślne parametrów. W niektórych przypadkach po instalacji wersji 8.4 można zmniejszyć wartość wymienionego parametru. Ustawianie wyjątkowo dużej wartości parametru `default_statistics_target` jest odradzane, ze względu na generowane wówczas ogromne obciążenie dla bazy danych.

Jeżeli w tabeli znajduje się kolumna, o której wiadomo, że potrzebuje lepszych danych statystycznych, można względem tej kolumny wykonać polecenie ALTER TABLE SET STATISTICS w celu dostosowania dla niej wartości parametru. Takie rozwiązanie jest lepsze niż zwiększanie wartości domyślnej parametru dla całego systemu, gdyż wtedy każda tabela „płaci” za wymaganie jednej kolumny w bazie danych. Zazwyczaj kolumny wymagające do prawidłowej pracy naprawdę dużej ilości danych statystycznych muszą mieć wartość parametru `default_statistics_target` równą 1000 (w nowszych wersjach PostgreSQL zwiększono ją do 10000), aby zmiana była

widoczna. Jest to wartość dużo większa niż ilość danych statystycznych konieczna do zebrania w każdej tabeli bazy danych.

Punkty kontrolne

Sposób działania mechanizmu punktów kontrolnych i dotyczące go parametry przedstawiono w poprzednim rozdziale. W tym miejscu nacisk położono na powszechnie stosowane praktyki ustawiania wartości początkowych dla tych parametrów.

checkpoint_segments

Każdy segment mechanizmu WAL zabiera do 16 MB. Jak przedstawiono na stronie <http://www.postgresql.org/docs/current/interactive/wal-configuration.html>, maksymalną liczbę segmentów, która może być w użyciu w danym momencie, można obliczyć ze wzoru:

$$(2 + \text{checkpoint_completion_target}) * \text{checkpoint_segments} + 1$$

Warto zwrócić uwagę, że baza danych PostgreSQL w wersjach wcześniejszych niż 8.3 nie obsługuje rozproszonych punktów kontrolnych, ale nadal można użyć powyższego wzoru i po prostu dla brakującej wartości wykorzystać poniższy fragment kodu:

```
checkpoint_completion_target=0
```

Wynik to całkowita wielkość wszystkich segmentów mechanizmu WAL, które mogą znaleźć się na dysku. Pozwala to zarówno na ustalenie ilości zajmowanego miejsca na dysku twardym, jak i oszacowanie ilości czasu potrzebnego na przeprowadzenie naprawy po awarii bazy danych. Oczekiwany wzrost wielkości dziennika zdarzeń pg_xlog został przedstawiony w poniższej tabeli.

checkpoint_segments	checkpoint_completion_target=0	target=0.5	target=0.9
3	112 MB	144 MB	160 MB
10	336 MB	416 MB	480 MB
32	1040 MB	1296 MB	1504 MB
64	2064 MB	2576 MB	2992 MB
128	4112 MB	5136 MB	5968 MB
256	8208 MB	10256 MB	11904 MB

Ogólny wniosek, który można wyciągnąć na podstawie przedstawionych tutaj danych, jest następujący: na każde 32 segmenty punktów kontrolnych trzeba się liczyć z akumulacją około 1 GB danych mechanizmu WAL. Ponieważ w przypadku takiej ilości danych naprawa bazy danych po awarii może zająć sporo czasu, wartość 32 to największa, jaką warto zastosować w poważnym, produkcyjnym serwerze bazy danych. Wartość domyślna 3 jest jednak dla większości systemów stanowczo za mała, nawet w niewielkich instalacjach należy rozważyć jej zwiększenie do 10.

Wartość większą niż 32 stosuje się zazwyczaj tylko w mniejszych serwerach przeprowadzających operacje typu *bulk-loading*, ponieważ może to znacznie zwiększyć wydajność, a czas naprawy bazy danych po ewentualnej awarii jest nieistotny. Bazy danych, które regularnie przeprowadzają operacje *bulk-loading*, mogą wymagać większej wartości parametru `checkpoint_segments`.

checkpoint_timeout

Wartość domyślna tego parametru wynosząca 5 minut jest wystarczająca dla większości instalacji. Jeżeli system nie nadąża z operacjami zapisu i zwiększono wartość parametru `checkpoint_segments` w taki sposób, że parametr `checkpoint_timeout` stał się podstawowym sposobem inicjowania operacji tworzenia punktu kontrolnego, rozsądne będzie rozważenie zwiększenia wartości także dla parametru `checkpoint_timeout`. Ustalenie przerwy w długości dziesięciu minut lub nawet więcej między tworzeniem punktów kontrolnych nie jest niebezpieczne, po prostu wydłuża czas naprawy bazy danych po ewentualnej awarii. Ponieważ jest to jeden z parametrów, które mają wpływ na długość czasu niedostępności bazy danych po awarii, jego wartość należy ustawiać bardzo ostrożnie.

checkpoint_completion_target

Po zwiększeniu wartości parametru `checkpoint_segments` do przynajmniej 10 rozsądne wydaje się również zwiększenie wartości parametru `checkpoint_completion_target` do jego praktycznego maksimum wynoszącego 0,9. W ten sposób uzyskiwane jest maksymalne rozproszenie punktów kontrolnych, co przynajmniej teoretycznie oznacza mniejsze obciążenie związane z operacjami wejścia-wyjścia. Jednak w niektórych sytuacjach utrzymywanie wartości domyślnej 0,5 nadal będzie lepszym rozwiązaniem, ponieważ zmniejsza prawdopodobieństwo, że operacje zapisu jednego punktu kontrolnego zajądą się z operacjami drugiego.

Wydaje się nieprawdopodobne, aby wartość poniżej 0,5 była w ogóle efektywna dla rozpraszania punktów kontrolnych. Co więcej, o ile liczba segmentów nie jest naprawdę ogromna, małe zmiany wartości parametru mają niewielkie znaczenie. Jedyne sensowne podejście polega na wypróbowaniu obu wartości (0,5 i 0,9) w aplikacji i sprawdzeniu na poziomie monitorowania systemu operacyjnego, która powoduje mniejsze obciążenie związane z operacjami wejścia-wyjścia.

Ustawienia mechanizmu WAL

Mechanizm WAL (ang. *Write-Ahead Log*) używany w PostgreSQL został omówiony w rozdziale 5.

wal_buffers

Dokumentacja parametru `wal_buffers` sugeruje, że wartość domyślna wynosząca 64 kB jest wystarczająca tak długo, dopóki pojedyncza transakcja nie przekracza wymienionej wartości.

Jednak w praktyce testy wydajności intensywnie wykorzystujące operacje zapisu pokazują, że optymalną wydajność można osiągnąć po ustawieniu znacznie większej wartości, co najmniej 1 MB lub więcej. Jediną wadą jest zwiększony poziom wykorzystania pamięci współdzielonej. Biorąc po uwagę, że nie ma potrzeby, aby więcej niż pojedynczy segment WAL musiał być buforowany, a także ilość pamięci instalowaną w nowoczesnych serwerach, parametrowi `wal_buffers` można obecnie przydzielić znacznie większą wartość:

```
wal_buffers=16MB
```

Po ustawieniu powyższej wartości można zapomnieć o parametrze `wal_buffers` jako potencjalnym wąskim gardle lub komponencie, który później trzeba optymalizować. W przypadku niewielkiej ilości pamięci w serwerze można rozważyć użycie mniejszej wartości.

wal_sync_method

W rozdziale 2. wyjaśniono, jak ważne jest skonfigurowanie serwera w taki sposób, aby unikał wykorzystywania nietrwałych buforów zapisu. Jednym z celów parametru `wal_sync_method` jest optymalizacja zachowania podczas korzystania z nietrwałych buforów zapisu.

Zachowanie domyślne jest w pewien sposób odmienne od większości opcji. Podczas kompilacji kodu źródłowego serwera rozważane są różne możliwe sposoby zapisu. Jeden, uznany za najbardziej efektywny, staje się później stosowany domyślnie w trakcie kompilacji. Jednak wartość ta nie jest zapisana w pliku konfiguracyjnym *postgresql.conf* wygenerowanym przez `initdb`, co odróżnia ją od innych automatycznie wykrywanych i charakterystycznych dla platform wartości, takich jak `shared_buffers`.

Przed zmianą czegokolwiek należy za pomocą polecenia `SHOW` sprawdzić, co w używanej przez czytelnika platformie zostało wykryte jako najszybsza, bezpieczna metoda. W systemie Linux wynik jest następujący:

```
postgres=# show wal_sync_method;
wal_sync_method
-----
fdatasync
```

Z kolei na platformach Windows i Mac OS X istnieje specjalny parametr gwarantujący, że system operacyjny wyczyści wszystkie bufory zapisu. Na wspomnianych platformach wartością bezpieczną, która włącza takie zachowanie, jest:

```
wal_sync_method=fsync_writethrough
```

Jeżeli ustawienie takie jest dostępne, naprawdę należy je wykorzystać! Pracuje ono prawidłowo i gwarantuje wykonywanie bezpiecznych operacji zapisu w bazie danych bez spowalniania innych aplikacji, co ma miejsce w przypadku całkowitego wyłączenia bufora zapisu w dysku twardym.

Jednak powyższe ustawienie nie będzie funkcjonowało na wszystkich platformach. Warto pamiętać, że po zmianie wartości domyślnej na przedstawioną powyżej nastąpi pewien spadek wydajności. Dzieje się tak zawsze podczas przejścia z buforowania niebezpiecznego do bezpiecznego.

Na innych platformach optymalizacja parametru `wal_sync_method` może być znacznie bardziej skomplikowana. Teoretycznie istnieje możliwość poprawienia wydajności operacji zapisu w systemach z rodziny Unix poprzez zmianę dowolnej metody zapisu używającej `write/fsync` lub `write/fdatasync` na stosującą zapis prawdziwie synchroniczny. Można się przekonać, po wydaniu polecenia `SHOW`, że na platformach obsługujących bezpieczne zachowanie zapisu `DSYNC` jest to opcja stosowana domyślnie:

```
wal_sync_method=open_datasync
```

Jednak także w tym przypadku opcja ta nie jest jawnie podana w pliku konfiguracyjnym. Jeżeli tak jest na używanej przez czytelnika platformie, można przeprowadzić jedynie niewielką optymalizację. Wartość `open_datasync` to ogólnie optymalne podejście, a kiedy wymieniona opcja jest dostępna, pozwala nawet na bezpośrednie używanie urządzeń wejścia-wyjścia, a także pomijanie bufora systemu operacyjnego.

W systemie Linux sytuacja jest prawdopodobnie najbardziej skomplikowana. Jak przedstawiono we wcześniejszym fragmencie kodu, platforma Linux domyślnie używa metody `fdatasync`. Za pomocą poniższego polecenia można włączyć tryb synchronicznych operacji zapisu:

```
wal_sync_method=open_sync
```

Ponadto w wielu przypadkach można odkryć, że metoda ta działa szybciej — czasami nawet znacznie szybciej — niż określona domyślnie. Jednak bezpieczeństwo jej stosowania zależy od używanego systemu plików. W większości systemów Linux domyślnym systemem plików jest `ext3`, który w wielu przypadkach nie obsługuje bezpiecznych operacji zapisu `O_SYNC`, co może doprowadzić do uszkodzenia danych. Przykłady niebezpieczeństw takiego ustawienia na platformie Linux przedstawiono w wątku „PANIC caused by `open_sync` on Linux” znajdującym się na stronie <http://archives.postgresql.org/pgsql-hackers/2007-10/msg01310.php>. Istnieją dowody, że w ostatnich wersjach jądra (2.6.32) ten problem nie istnieje, jeśli używany jest system plików `ext4`, ale takie rozwiązanie nie zostało jeszcze intensywnie przetestowane na poziomie bazy danych.

W każdym przypadku podczas przeprowadzania własnych testów wartości parametru `wal_sync_method` należy wykonać również test polegający na „wyciągnięciu z gniazda sieciowego wtyczki przewodu zasilającego serwer”. Spowoduje to nagłe odcięcie zasilania serwera i pozwoli na sprawdzenie, czy nastąpiła utrata jakichkolwiek danych w wyniku stosowania testowanej metody. Zalecane jest również przeprowadzenie długotrwałych testów przy dużym obciążeniu, aby znaleźć sporadyczne błędy, które mogą doprowadzić do awarii.

Replikacja WAL i PITR

Parametry `archive_mode`, `archive_command` oraz `archive_timeout` zostały omówione w rozdziale 14.

Ustawienia na poziomie klienta

Wprowadź wszystkie parametry omawiane w tym podrozdziale mogą być dostosowane na poziomie klienta, pożądane jest ustawienie dla nich odpowiednich wartości początkowych w głównym pliku konfiguracyjnym. Gdy trzeba zmienić wartość parametru w poszczególnych klientach, zawsze można ją wprowadzić na czas trwania sesji, wykorzystując do tego polecenie SET.

`effective_cache_size`

Jak wspomniano w poprzednim rozdziale, baza danych PostgreSQL używa zarówno własnej dedykowanej jej pamięci (`shared_buffers`), jak i korzysta z bufora systemu plików. W pewnych sytuacjach, podczas podejmowania decyzji dotyczących efektywności użycia indeksu bądź nie, baza danych porównuje obliczone przez siebie wielkości względem efektywnej sumy wszystkich wymienionych buforów. Wartość tę spodziewa się znaleźć w parametrze `effective_cache_size`.

Ta sama reguła, wedle której wartość parametru `shared_buffers` powinna wynosić około 25% ilości pamięci systemowej, mówi także, że wartość parametru `effective_cache_size` powinna wynosić od 50 do 75% ilości pamięci RAM. Aby znacznie dokładniej oszacować wartości, trzeba na początek przyjrzeć się buforowi systemu plików:

- w systemach z rodziny Unix oszacowanie wielkości bufora systemu plików wymaga dodania wartości `free` i `cached` wyświetlanych przez polecenia `free` i `top`;
- w systemach Windows należy wyświetlić okno Menedżera zadań Windows, przejść na zakładkę *Wydajność* i odczytać wartość w linii *Bufor systemu*.

Po przyjęciu, że baza danych została już uruchomiona, do obliczonych wartości trzeba dodać jeszcze wartość `shared_buffers`, w ten sposób otrzyma się `effective_cache_size`. Jeżeli baza danych nie została jeszcze uruchomiona, bufor systemu operacyjnego będzie zwykle wystarczająco dokładny do oszacowania wartości parametru. Po uruchomieniu bazy danych większość pamięci przeznaczony dla bazy danych zwykle i tak będzie zaalokowana na potrzeby jej bufora.

Parametr `effective_cache_size` nie powoduje alokacji jakiegokolwiek pamięci. Jest używany wyłącznie po to, by określić sposób wykonywania zapytań, i jego ogólne oszacowanie jest wystarczające do większości celów. Kiedy jednak wartość ta będzie zbyt wysoka, wykonywanie zapytań może skutkować zakłóceniami w buforze zarówno bazy danych, jak i systemu operacyjnego

wynikającymi z konieczności odczytania ogromnej liczby bloków wymaganych do wykonania zapytania, które bardzo łatwo zmieści się w pamięci RAM.

Bardzo rzadko zdarza się, aby parametr ten był optymalizowany na poziomie klienta, nawet jeśli jest to możliwe.

synchronous_commit

W rozdziale 2. obciążenie związane z oczekiwaniem, aż fizyczny dysk zakończy operację zatwierdzania, wskazano jako potencjalne wąskie gardło podczas zatwierdzania transakcji. Jeżeli czytelnik nie posiada podtrzymywanego bateryjnie bufora zapisu, który mógłby przyspieszyć takie operacje, ale koniecznie chce przyspieszyć operacje zatwierdzania transakcji, rodzi się pytanie, w jaki sposób można to zrobić. Standardowym podejściem jest wyłączenie opcji `synchronous_commit`, która czasami nazywana jest opcją włączającą asynchroniczne zatwierdzenia. Powoduje ona zgromadzenie w większą grupę operacji zatwierdzenia w częstotliwości określonej przez powiązany z nią parametr `wal_writer_delay`. Ustawienia domyślne gwarantują realne zatwierdzenie transakcji na dysku po upływie co najwyżej 600 milisekund od chwili zatwierdzenia transakcji przez klienta. W trakcie tego okresu czasu, który można skrócić, licząc się jednocześnie ze spadkiem szybkości, niezatwierdzone dane nie będą odzyskane po ewentualnej awarii serwera.

Warto zwrócić uwagę, że opisywany parametr można wyłączyć dla pojedynczego klienta na czas trwania jego sesji, zamiast wyłączyć go dla całego serwera. Wyłączenie dla klienta odbywa się za pomocą polecenia SET:

```
SET LOCAL synchronous_commit TO OFF;
```

W ten sposób czytelnik dysponuje opcją stosowania różnych fizycznych gwarancji zatwierdzania dla odmiennych typów danych wstawianych do bazy danych. Aktywnie monitorowana tabela — do której dane są najczęściej wstawiane, a ułamek sekundy strat akceptowany — jest dobrym kandydatem dla asynchronicznych zatwierdzeń. W przypadku rzadziej zapisywanej tabeli przechowującej rzeczywiste transakcje finansowe preferowanym rozwiązaniem jest stosowanie standardowych, synchronicznych zatwierdzeń.

work_mem

Kiedy wykonywane jest zapytanie wymagające sortowania danych, baza danych szacuje ilość danych koniecznych do użycia i następnie porównuje obliczoną wartość do określonej w parametrze `work_mem`. Jeżeli jest większa (a wartość domyślna parametru `work_mem` wynosi 1 MB), zamiast sortować dane w pamięci, zapisze je na dysku i przeprowadzi operację sortowania dyskowego. Taka operacja jest znacznie wolniejsza niż sortowanie przeprowadzane w pamięci. Dlatego też, w przypadku regularnego sortowania danych i posiadania wolnej pamięci, zwiększenie wartości parametru `work_mem` może być jednym z najefektywniejszych sposobów przyspieszenia działania serwera. Magazyny danych generujące ogromne raporty mają w swoich serwerach gigabajty pamięci przypisane parametrowi `work_mem`.

Problem polega na tym, że niekoniecznie można łatwo przewidzieć liczbę operacji sortowania przeprowadzanych przez klienta, a parametr `work_mem` jest określany dla każdej operacji sortowania, a nie dla każdego klienta. Oznacza to, że ilość pamięci używanej przez `work_mem` może być teoretycznie nieograniczona, jeżeli liczba klientów jednocześnie przeprowadzających sortowanie będzie wystarczająco duża.

W praktyce w typowym zapytaniu nie ma zbyt wielu operacji sortowania, najczęściej tylko jedna bądź dwie. Ponadto nie wszyscy aktywni klienci będą w tym samym czasie przeprowadzali operacje sortowania. W trakcie określania ilości pamięci dla parametru `work_mem` zwykle bierze się pod uwagę ilość wolnej pamięci RAM po alokacji `shared_buffers` (ta sama wielkość bufora systemu operacyjnego obliczana na potrzeby parametru `effective_cache_size`), dzieli ją przez wartość `max_connections`, a następnie bierze tylko część obliczonej wartości. Połowa obliczonej liczby będzie wartością dość dużą dla parametru `work_mem`. W takim przypadku w serwerze może zabraknąć pamięci, jeżeli każdy klient będzie przez cały czas przeprowadzał po dwie operacje sortowania, ale prawdopodobieństwo wystąpienia takiej sytuacji jest znikome.

Obliczenia wartości `work_mem` są coraz częściej stosowane w najnowszych wersjach PostgreSQL w celu oszacowania, czy struktura hash może zostać zbudowana w pamięci. Pamięć jest używana także przez klienta, a nie jedynie przeznaczona dla operacji sortowania. Przedstawiono po prostu najłatwiejszy sposób rozwiązywania problemów dotyczących rodzajów alokacji pamięci.

Podobnie jak `synchronous_commit`, parametr `work_mem` może być również ustawiany na poziomie klienta. W ten sposób wartość domyślną można zdefiniować na średnim poziomie i zwiększać pamięć do sortowania dla użytkowników, którzy wykonują zapytania generujące ogromne raporty.

random_page_cost

Parametr często jest optymalizowany, ale jego objaśnienie wymaga przedstawienia wielu informacji dotyczących planowania zapytań. Temat ten został omówiony w rozdziale 10. Szczególnie we wcześniejszych wersjach PostgreSQL zmniejszenie wartości parametru `random_page_cost` poniżej wartości domyślnej — na przykład z 4.0 na 2.0 — było powszechnie stosowanym rozwiązaniem. Celem było zwiększenie prawdopodobieństwa, że planista zapytania wykorzysta zapytania indeksowane zamiast alternatywy w postaci skanowania sekwencyjnego. Ponieważ w nowszych wersjach serwera wbudowany jest sprytniejszy planista zapytania, wspomnianej techniki nie należy stosować. O wiele rozsądniejszym rozwiązaniem jest zebranie lepszych danych statystycznych i wykorzystanie parametrów dotyczących pamięci jako podstawowych sposobów wpływania na planistę zapytań.

constraint_exclusion

Jeżeli używana jest baza danych PostgreSQL w wersji 8.3 lub wcześniejszej, a do partycjonowania danych zastosowano oferowaną przez bazę danych funkcję tabeli dziedziczenia, parametr `constraint_exclusion` musi być włączony. Powody takiego stanu rzeczy wyjaśniono w rozdziale 15.

Począwszy od wersji 8.4, wartością domyślną parametru `constraint_exclusion` jest nowe, sprytniejsze ustawienie o nazwie `partition`, które w większości przypadków sprawdza się doskonale i nie musi być modyfikowane.

Optymalizacje niezalecane

W pliku konfiguracyjnym `postgresql.conf` znajduje się kilka parametrów, dla których w innych poradnikach przedstawiono kiepskie wskazówki lub w serwerze administrowanym przez czytelnika mają ustawione nieprawidłowe wartości. Inne mają nazwy sugerujące użycie wraz z parametrami, które w rzeczywistości nie istnieją. W tym punkcie opisano najczęściej spotykane opcje, których optymalizacja jest niezalecana.

fsync

Jeżeli czytelnik chce zignorować proces naprawy po wystąpieniu awarii, może to zrobić poprzez wyłączenie parametru `fsync`. W ten sposób wartość parametru `wal_sync_method` nie będzie miała znaczenia, ponieważ serwer i tak nie wykona żadnych wywołań `sync` mechanizmu WAL.

Trzeba w tym miejscu zdać sobie sprawę, że w przypadku wystąpienia jakiegokolwiek awarii przy wyłączonym parametrze `fsync` baza danych prawdopodobnie będzie uszkodzona i uruchomienie serwera stanie się niemożliwe. Wprawdzie to okropna sytuacja, ale zwiększenie wydajności w wyniku wyłączenia procesu naprawy po awarii jest tak ogromne, że czytelnik może natknąć się na sugestie wyłączenia parametru `fsync`. Równie nieufnie należy traktować inne rady pochodzące z tych samych źródeł, które sugerują wyłączenie parametru `fsync`, ponieważ wyłączenie tego parametru jest szalenie niebezpieczne.

Jedynym powodem, dla którego taki pomysł zyskał zwolenników we wcześniejszych wersjach PostgreSQL, był brak innego sposobu na zmniejszenie liczby wywołań `fsync` — to taki kompromis: nieco większa wydajność kosztem mniejszej niezawodności. Od wersji 8.3 użytkownicy, zamiast wyłączać parametr `fsync`, zrobią lepiej, wyłączając parametr `synchronous_commit`.

Istnieje jedna sytuacja, gdy użycie parametru `fsync=off` nadal ma sens — to początkowe operacje typu *bulk-loading*. Jeżeli do bazy wstawiane są ogromne ilości danych, a sprzęt nie jest wyposażony w bateryjnie podtrzymywany bufor zapisu, taka operacja wstawiania danych będzie trwała zdecydowanie za długo, aby uznać ją za praktyczną. W takim przypadku wyłączenie parametru `fsync` podczas przeprowadzania operacji wstawiania danych — w przypadku awarii serwera wszystkie dane i tak można łatwo odtworzyć — może być jedynym sposobem przyspieszenia całej operacji. Po zakończeniu procesu wstawiania należy z powrotem włączyć parametr `fsync`.

W niektórych systemach parametr `fsync` jest wyłączany w serwerach posiadających redundancyjną kopię bazy danych — na przykład w systemach zapasowych używanych do generowania raportów. Jeżeli w takim przypadku dane zostaną uszkodzone, zawsze można przeprowadzić ponowną synchronizację z głównym systemem.

full_page_writes

Podobnie jak przy `fsync`, wyłączenie parametru `full_page_writes` zwiększa wydajność za cenę większego ryzyka uszkodzenia bazy danych. Przed wyłączeniem tego parametru należy bardzo dokładnie i ostrożnie przeanalizować możliwości systemu plików oraz używanego sprzętu, aby mieć gwarancję, że nie dojdzie do częściowego zapisu stron.

commit_delay i commit_siblings

Przed implementacją `synchronous_commit` podejmowane były próby dodania tego rodzaju funkcji za pomocą parametrów `commit_delay` i `commit_siblings`. W większości przypadków nie są to parametry efektywne do optymalizacji. Uzyskanie realnego przyśpieszenia poprzez dostosowanie wartości tych parametrów jest niezwykle trudne, za to bardzo łatwo można doprowadzić do spowolnienia wykonywania każdej transakcji. Jedyny przypadek, kiedy opisywane parametry mogą okazać się przydatne, dotyczy systemów o ogromnej ilości operacji wejścia-wyjścia. Ustawienie bardzo małej wartości opóźnienia może spowodować przeprowadzanie operacji zapisu w większych blokach, co czasami w połączeniu z większą wartością jednostki macierzy RAID okazuje się lepszym rozwiązaniem.

max_prepared_transactions

Wielu użytkowników po spojrzeniu na nazwę tego parametru będzie przekonanych, że dotyczy transakcji składowanych, czyli techniki powszechnie stosowanej w celu uniknięcia wstawiania złośliwego kodu SQL, i odczuje potrzebę zwiększenia wartości tego parametru. To jednak błędne założenie, omawiany parametr i transakcje składowane nie są ze sobą powiązane. Transakcja składowana to taka, która używa polecenia `PREPARE TRANSACTION` w celu użycia dwuetapowego zatwierdzenia (ang. *two-phase commit*, 2PC). Jeżeli czytelnik nie używa tego polecenia oraz techniki 2PC, może pozostawić dla tego parametru wartość domyślną. Natomiast w przypadku używania funkcji dwuetapowego zatwierdzania konieczność zwiększenia wartości parametru `max_prepared_transaction` wystąpi prawdopodobnie tylko wtedy, kiedy trzeba będzie dopasować ją do liczby połączeń.

Zapytania włączające parametry

Istnieje możliwość wyłączenia wielu technik stosowanych przez planistę zapytania, co ma na celu uniknięcie użycia niewłaściwego typu zapytania. Czasami jest to stosowane jako rodzaj obejścia problemu polegającego na tym, że PostgreSQL nie obsługuje bezpośrednich podpowiedzi optymalizatora dotyczących sposobu wykonania zapytania. Czytelnik mógł się spotkać z przedstawionym poniżej wierszem kodu jako sugerowanym sposobem wymuszenia użycia indeksów zamiast skanowania sekwencyjnego:

```
enable_seqscan = off
```

Ogólnie rzecz ujmując, to zła praktyka. Zamiast niej, aby podejmować lepsze decyzje, należy dążyć do poprawienia informacji przekazywanych optymalizatorowi zapytań. Temat został omówiony w rozdziale 10.

Optymalizacja ustawień nowego serwera

Istnieje kilka sposobów połączenia wszystkich przedstawionych dotąd informacji w jeden proces optymalizacji ustawień nowego serwera. Wybór najlepszego sposobu zależy od oczekiwań stawianych serwerowi, a także od samego użytkownika — czy chce przeprowadzić samodzielną optymalizację, czy będzie stosował zalecane tutaj wartości parametrów.

Wskazówki dotyczące serwerów dedykowanych

Początkową optymalizację serwera można sprowadzić do całkiem mechanicznego procesu. Oto on.

1. Dostosowanie rejestracji zdarzeń w taki sposób, aby dostarczała większej ilości informacji.
2. Ustalenie wielkości parametru `shared_buffers`. Należy rozpocząć od wartości równej 25% pamięci systemowej. Jeżeli czytelnik używa najnowszych wersji PostgreSQL obsługujących rozpraszanie punktów kontrolnych i wie, że dane obciążenie serwera odniesie korzyści po przydzieleniu buforowi większej ilości pamięci, może spróbować zwiększania wartości `shared_buffers`. W przypadku platformy, na której parametr ten nie jest tak użyteczny, jego wartość należy odpowiednio dopasować lub nawet zmniejszyć.
3. Rozsądne oszacowanie maksymalnej liczby połączeń. Jest to ograniczenie bezwzględne. Po osiągnięciu ustalonej liczby połączeń próby nawiązania połączenia przez nowych klientów będą odrzucane.
4. Uruchomienie serwera z tymi parametrami początkowymi. Następnie trzeba sprawdzić ilość pamięci pozostałej dla bufora systemu plików.
5. Kolejny krok to określenie wielkości parametru `effective_cache_size` na podstawie wielkości parametru `shared_buffers` i bufora systemu operacyjnego.
6. Podzielenie wielkości bufora systemu operacyjnego przez wartość `max_connections`, a później jeszcze przez dwa. W ten sposób czytelnik otrzymuje rozsądną wartość dla parametru `work_mem`. Jeżeli aplikacja w żaden sposób nie jest uzależniona od wydajności operacji sortowania, odpowiednie będzie ustawienie znacznie niższej wartości parametru `work_mem`.
7. Ustawienie dla parametru `maintenance_work_mem` wartości około 50 MB na każdy gigabajt pamięci RAM.
8. Zwiększenie wartości parametru `checkpoint_segments` do przynajmniej 10. Jeżeli czytelnik posiada sprzęt klasy serwerowej z bateryjnie podtrzymywanym buforem zapisu, wartość 32 dla parametru `checkpoint_segments` będzie znacznie lepszą wartością domyślną.

9. Zmiana wartości domyślnej parametru `wal_sync_method`. Jeżeli używana jest platforma, na której wartość domyślna tego parametru jest niebezpieczna, należy zmienić tę wartość.
10. Wartość parametru `wal_buffers` trzeba zwiększyć do 16 MB.
11. Dla baz danych PostgreSQL w wersjach niższych niż 8.4 warto rozważyć zwiększenie wartości parametrów `default_statistics_target` (do 100, wartość domyślna w nowszych wersjach PostgreSQL) i `max_fsm_pages` na podstawie informacji o przewidywanym obciążeniu dla tej bazy danych.

Po skonfigurowaniu pewnej liczby serwerów przeznaczonych do obsługi danej aplikacji czytelnik będzie miał znacznie lepsze rozeznanie, które wartości początkowe mają sens, aby rozpocząć z nimi pracę. Szczególnie wartości końcowe parametrów `checkpoint_segments` i `work_mem` mogą bardzo istotnie odbiegać od tutaj zalecanych.

Wskazówki dotyczące serwerów współdzielonych

Jeżeli serwer bazy danych współdzieli sprzęt z inną aplikacją — często spotykana jest sytuacja umieszczania wielu aplikacji bazodanowych w pojedynczym systemie — podczas optymalizacji nie można stosować tak agresywnego podejścia, jakie przedstawiono w poprzednim podrozdziale. W takim przypadku dokładna procedura jest trudna do nakreślenia. Należy podjąć próbę przeprowadzenia optymalizacji wartości parametrów związanych z pamięcią i zastosować dolne granice zalecanych wartości:

- parametrowi `shared_buffers` należy przydzielić tylko 10% pamięci RAM, nawet na platformach, na których normalnie zaleca się przydzielenie większej wartości;
- parametrowi `effective_cache_size` należy przydzielić 50% lub mniej pamięci RAM, prawdopodobnie mniej, jeśli wiadomo, że aplikacja będzie używać dużej ilości pamięci;
- trzeba bardzo ostrożnie podchodzić do zwiększania wartości parametru `work_mem`.

Inne sugestie przedstawione w poprzednim podrozdziale mają zastosowanie również tutaj — na przykład użycie większych wartości dla parametru `checkpoint_segments` i odpowiedni wybór wartości parametru `wal_sync_method` nie różnią się niczym w obu systemach.

Następnie trzeba zasymulować działanie aplikacji z pełnym obciążeniem i sprawdzić ilość dostępnej pamięci RAM, aby się przekonać, czy możliwe jest przydzielenie bazie danych większej ilości pamięci. Proces może wymagać kilkakrotnego powtórzenia i na pewno powinien być powiązany z przeprowadzaniem testów wydajności na poziomie aplikacji, o ile jest taka możliwość. W systemie współdzielonym nie ma sensu przydzielanie bazie danych większej ilości pamięci, jeżeli aplikacja bądź inna warstwa buforująca, na przykład na poziomie puli połączeń, nie będzie efektywnie wykorzystywała tej pamięci. Ta sama idea — użycie rozsądnych wartości początkowych, a następnie stopniowe przeprowadzanie optymalizacji na podstawie monitorowania — sprawdza się doskonale także w serwerach dedykowanych.

pgtune

Od PostgreSQL w wersji 8.4 narzędzie pgtune dostępne na stronie <http://pgfoundry.org/projects/pgtune/> może być użyte do utworzenia początkowego pliku konfiguracyjnego *postgresql.conf* dla nowego serwera. Narzędzie pozwala użytkownikowi na zasugerowanie przewidywanego rodzaju obciążenia bazy danych z zakresu od stacji roboczej programisty aż do dedykowanego serwera magazynu danych. Na podstawie tych danych wejściowych oraz parametrów, takich jak ilość pamięci RAM zainstalowana w serwerze, narzędzie generuje plik konfiguracyjny wstępnie zoptymalizowany dla najważniejszych parametrów systemowych, stosuje przy tym metody podobne do omówionych w rozdziale. Uzyskany wynik nie będzie tak dobry jak zastosowanie wskazówek przedstawionych dla serwera dedykowanego i samodzielne przeprowadzanie pomiarów, ale pozwoli na szybkie rozpoczęcie pracy z ogólnie dobrą konfiguracją. Każdy rozsądnie zmodyfikowany plik konfiguracyjny *postgresql.conf* będzie znacznie lepszy od domyślnego, który dostrojony jest jedynie pod kątem alokacji jak najmniejszej ilości pamięci współdzielonej.

Podsumowanie

W konfiguracji bazy danych PostgreSQL istnieje niemal 200 wartości, które można zmodyfikować; poprawne ustawienie w aplikacji wszystkich może być całkiem dużym wyzwaniem. Wskazówki przedstawione w rozdziale powinny pomóc czytelnikowi w rozpoczęciu procesu optymalizacji konfiguracji i uniknięciu najczęściej spotykanych pułapek. Gdy użytkownik napotka problemy, powinny też pokazać, które ustawienia są cenniejsze.

- Wartości domyślne w pliku konfiguracyjnym serwera powodują zapisywanie w dzienniku zdarzeń niewielu informacji i charakteryzują się wyjątkowo skąpymi ustawieniami pamięci. W każdym serwerze należy przynajmniej przeprowadzić podstawową optymalizację w celu rozwiązania najgorszych problemów.
- Ustawienia parametrów związanych z pamięcią, czyli przede wszystkim `shared_buffers` i `work_mem`, powinny być przeprowadzane bardzo ostrożnie, aby nie doprowadzić do sytuacji, w której systemowi zabraknie pamięci.
- W celu przygotowania właściwego planu wykonania zapytania planista zapytania musi mieć prawidłowe informacje o pamięci oraz dobre dane statystyczne dotyczące tabeli.
- Demon `autovacuum` ma znaczenie krytyczne dla zapewnienia planiście zapytania odpowiednich informacji. Ponadto demon `autovacuum` zapewnia tabelom uruchamianie właściwych procesów konserwacyjnych.
- W wielu przypadkach wprowadzenie zmiany konfiguracyjnej nie wymaga ponownego uruchomienia serwera, a wiele parametrów można nawet modyfikować dla poszczególnych klientów, zapewniając w ten sposób naprawdę dokładną optymalizację.

Skorowidz

\$PGDATA, 124, 143
2warm, 383
3ware, 45

A

ACID, 49
Adaptive Replacement Cache, 105
AFTER, 425
AFTER DELETE ON, 377
AFTER UPDATE ON, 377
Aggregate, 271
agregacja WindowAgg, 274
agresywne uaktualnienia PostgreSQL, 430
aktywność bufora bazy danych, 335
aktywność tworzenia punktu kontrolnego, 335
algorytm clock-sweep, 133
algorytm szybkiego sortowania, 269
alloc_per_second, 339
alokacja pamięci współdzielonej, 119
ALTER TABLE, 189, 425, 435, 440
ALTER TABLE ADD FOREIGN KEY, 417
ALTER TABLE SET STATISTICS, 154
ALTER TABLESPACE, 437
AMD, 36, 57
analiza dzienników zdarzeń, 203
analiza opóźnienia, 219
analiza planu zapytania, 264
analiza pliku dziennika zdarzeń, 200
analiza zawartości bufora bazy danych, 136
ANALYZE, 152, 154, 232, 259, 290, 291, 407, 421
 cele danych statystycznych, 293
 dostosowanie wartości celu
 dla poszczególnych kolumn, 294

 obszary trudne do oszacowania, 295
 różne wartości, 294
AND, 409
Anti Join, 286
Anticipatory Scheduling, 100
aplikacje PostgreSQL, 31
Append, 275
application_name, 438
ARC, 105
architektura SN, 410
archive_cleanup_command, 436
archive_command, 383, 419
archive_mode, 433
archive_timeout, 383
Areca, 45
array_to_string(), 291
as, 100
ATA, 37
ataki
 DoS, 148
 SQL Injection, 122, 423
atime, 97
auto_explain, 199, 200, 435, 439
auto_explain.log_min_duration, 200
automatyczna analiza, 190
autovacuum, 121, 152, 153, 175, 179, 180, 181, 182, 186, 187, 232, 417, 432, 435
 błędy spowodowane brakiem pamięci, 187
 długo wykonywane transakcje, 188
 monitorowanie procesu, 182
 obciążony serwer, 187
 opcje dla poszczególnych tabel, 184
 problemy, 185
 rejestrwanie zdarzeń, 182

autovacuum
 uruchamianie procesu, 183
 włączanie demona w starszych wersjach,
 153
 autovacuum_cost_limit, 187
 autovacuum_freeze_max_age, 175
 autovacuum_max_workers, 121, 153, 186, 187
 autovacuum_naptime, 186
 autovacuum_vacuum_cost_delay, 180, 187
 autovacuum_vacuum_cost_limit, 180
 autovacuum_vacuum_scale_factor, 183
 autovacuum_vacuum_threshold, 183
 AVG(), 271

B

backend, 145
 bariery zapisu, 94
 obsługa przez napędy, 94
 obsługa przez systemy plików, 95
 base, 110
 Battery-Backed Write Cache, 50
 baza danych, 26
 Dell Store 2, 254
 Pagila, 254
 BBC, 50
 BBWC, 50, 53
 BEGIN, 169, 326, 416, 418
 Berkley Fast File System, 102
 bezpośrednio podłączona pamięć masowa, 46
 Binary Tree, 191
 blktrace, 349
 block_size, 121, 336
 blockdev, 97
 blok indeksu, 230
 blokady, 325
 informacje, 327
 oczekiwanie na blokadę tabeli, 331
 oczekiwanie na blokadę transakcji, 330
 rejestrowanie informacji o blokadach, 332
 zakleszczenia, 332
 bloki bazy danych, 131
 sposoby zapisu zmodyfikowanych
 bloków, 132
 bloki LRU, 132
 blokowanie rekordów, 171
 bonnie++ 2.0, 72
 bonnie++ ZCAV, 73
 boot_val, 144
 B-tree, 191, 245
 Btrfs, 93
 Bucardo, 388
 buffers_backend, 132
 buffers_checkpoint, 132, 340
 buffers_clean, 132
 bufor ARC, 105
 bufor bazy danych, 117, 122, 132, 335
 analiza zawartości, 136
 dostosowanie wielkości bufora, 141
 generowanie podsumowania, 138
 jednostki pamięci, 118
 konfiguracja układu dysków, 124
 licznik użycia tabel, 138
 naprawa bazy danych po awarii, 128
 natężenie operacji dyskowych, 135
 ograniczenia, 135
 ogromne systemy, 136
 pamięć współdzielona w systemie Unix,
 119
 pg_buffercache, 123
 początkowe wskazówki dotyczące
 wielkości, 135
 podsumowanie zawartości bufora
 z użyciem danych procentowych, 139
 podwójne buforowanie, 134
 przeciążenie punktu kontrolnego, 134
 przegląd wielkości bufora, 141
 przegląd zawartości bufora, 137
 relacje, 138
 rozproszenie użycia bufora, 140
 serwer współdzielony, 136
 tworzenie bloku w bazie danych, 126
 wczesne wersje, 135
 Windows, 135
 zapis zmodyfikowanych bloków
 na dysku, 127
 bufor bez wstrzymywania zapisu, 49
 wydajność, 52
 bufor systemu operacyjnego, 132, 134
 bufor z opóźnionym zapisem, 49, 50
 bufor zapisu, 49, 98
 wyłączanie, 52

- buforowanie bazy danych, 376
 - memcached, 376
 - pgmemcache, 377
- buforowanie odczytu, 98
- bulk-loading, 415
 - czyszczenie po operacji wczytania danych, 420
 - dodawanie ograniczeń, 419
 - metody wczytywania danych, 416
 - optymalizacja operacji, 417
 - pominięcie optymalizacji mechanizmu WAL, 418
 - ponowne tworzenie indeksów, 419
 - przywracanie równoległe, 419

C

- Cacti, 366
 - PostgreSQL, 366
 - Windows, 366
- Cassandra, 26
- cele danych statystycznych, 293
 - dostosowanie wartości celu dla poszczególnych kolumn, 294
 - różne wartości, 294
- cfq, 100, 101
- cfs, 224
- CHECK, 425
- check_postgres, 193, 194, 365
- CHECKPOINT, 129
- checkpoint_completion_target, 130, 155, 156, 432
- checkpoint_segments, 129, 130, 147, 155, 164, 340, 417
- checkpoint_timeout, 129, 156, 339, 417
- checkpoints_req, 129
- chkdsk, 87
- clock-sweep, 133
- CLUSTER, 125, 185, 189, 192, 239, 243, 433, 440
- COMMIT, 52, 64, 65, 75, 416
- commit_delay, 163
- commit_siblings, 163
- Common-Table Expression, 280, 434
- Completely Fair Queuing, 100
- Completely Fair Scheduler, 224
- constraint_exclusion, 161, 298, 403, 407, 434
- context, 145
- contrib, 27, 125, 199, 201
- COPY, 96, 416, 417
- CouchDB, 26
- COUNT(), 271, 421
- CPU governor, 61
- cpu_index_tuple_cost, 260, 300
- cpu_operator_cost, 260
- cpu_tuple_cost, 260
- cpuinfo, 61
- CREATE DATABASE, 109
- CREATE INDEX, 154, 235, 248, 417
- CREATE INDEX CONCURRENTLY, 189, 243
- CREATE OR REPLACE FUNCTION, 397, 398
- CREATE RULE, 399
- CREATE TABLE, 435
- CREATE TABLE...WITH OIDS, 265
- CREATE TEMPORARY TABLE, 111
- CREATE TRIGGER, 397
- CREATE UNIQUE INDEX, 249
- CSV, 197
- CTE, 280, 434
- ctid, 265
- CURRENT ROWS, 437
- CURRENT_DATE(), 407
- current_setting, 336
- current_timestamp(), 325
- cursor_tuple_fraction, 298
- cust_hist, 287
- custom_variable_classes, 200
- cykl życia aplikacji PostgreSQL, 31
- cykl życia bloku bazy danych, 131
- cykl życia widoczności rekordu, 168
- czas dostępu do plików, 97
- czas wyszukiwania, 76
- częstotliwość kreowania punktów kontrolnych, 129
- czyszczenie bufora, 257
- czyszczenie po operacji wczytania danych, 420

D

- dane statystyczne, 152, 290, 315
 - cele danych statystycznych, 293
 - dane dotyczące całej bazy danych, 324

- dane statystyczne
 - dane dotyczące indeksu, 322
 - dane dotyczące tabel, 230, 318
 - histogram, 291
 - pg_stats, 290
 - przeglądanie, 290
 - szacowanie, 290
 - widoki, 315
 - zakres sprawdzanych danych, 291
- DAS, 46
- data_directory, 124
- Database Connection Pool, 375
- DBCP, 375
- dbt, 227
- dd, 69
- DDF, 44
- DDL, 151
- DDR SDRAM, 57
- deadline, 100, 101
- deadlock_timeout, 332, 333
- debugger PL/pgSQL, 424
- default_statistics_target, 154, 165, 293, 294, 434
- DEFERRABLE, 425, 438
- dekodowanie informacji o blokadzie, 327
- DELETE, 172, 176, 298, 299, 400
- Dell Store 2, 254
- demon autovacuum, 152, 181
- Denial Of Service, 148
- Direct Attached Storage, 46
- Direct Memory Access, 43
- dirty_background_ratio, 99
- dirty_ratio, 99
- DISCARD ALL, 372
- Disk Data Format, 44
- DISTINCT, 272, 273
- DMA, 43
- dmesg, 95
- DoS, 148
- dostęp sekwencyjny, 63
- dostrajanie systemów plików, 96
- dowiązania symboliczne, 108
- DROP INDEX, 241
- DROP TABLE, 169, 384
- drop_caches, 258
- drzewo katalogów bazy danych, 109
- DTrace, 32, 428, 435
- DTrace in FreeBSD, 429
- dual-channel, 60
- dynamiczne wywoływanie funkcji, 398
- dyski twarde, 37
 - bariery zapisu, 94
 - bonnie++, 70
 - dd, 69
 - dostęp sekwencyjny, 63
 - hdtune, 65
 - IOPS, 68
 - konfiguracja, 85
 - liczba operacji wejścia-wyjścia na sekundę, 61
 - liczba wykonywanych operacji zatwierdzania, 64
 - macierz RAID, 38
 - maksymalna wielkość systemu plików, 85
 - narzędzia do testowania wydajności, 65
 - NCQ, 95
 - nieprzewidywalna wydajność w Windows, 69
 - niezawodność, 41
 - obsługa błędów, 40
 - oczekiwana wydajność dysku, 80
 - odczytywanie bloków danych, 69
 - oprogramowanie firmware, 42
 - optymalizacja pojemności, 63
 - powody niskiej wydajności, 81
 - Random Access, 68
 - schemat partycjonowania, 86
 - short stroking, 63
 - spindle, 62
 - SSD, 42
 - swobodny dostęp, 61
 - średni czas dostępu, 62
 - test_fsyc, 64
 - testy typu short stroking, 67
 - testy wydajności, 61, 77
 - wyniki testu wydajności, 78
 - zapisywanie bloków danych, 69
 - ZCAV, 63
- dziennik zapisu z wyprzedzeniem, 128
- dziennik zatwierdzeń, 133
- dzienniki zdarzeń zapytań, 194

E

effective_cache_size, 159, 161, 164, 165, 283, 295, 296
 effective_io_concurrency, 434
 EFI, 86
 elevator, 100
 emulacja DTrace, 429
 enable_bitmapscan, 303
 enable_hashagg, 303
 enable_hashjoin, 303, 305
 enable_indexscan, 303
 enable_material, 303
 enable_mergejoin, 303
 enable_nestloop, 303
 enable_seqscan, 163, 303, 305
 enable_sort, 303
 enable_tidscan, 303
 EPQA, 205
 eSATA, 41
 EVERY(), 271
 EXCEPT, 278
 EXCEPT ALL, 278
 exclusion constraints, 439
 EXISTS, 286, 300
 EXPLAIN, 152, 200, 231, 241, 256, 259, 436

- czyszczenie bufora, 257
- dane wyjściowe, 437
- dane wyjściowe w postaci czytelnej dla komputera, 263
- graficzne przedstawienie danych, 262
- narzędzia analizy danych, 262, 264
- obciążenie związane z pomiarem, 256
- rozbudowane dane wyjściowe, 263
- rozgrzany bufor, 257
- struktura węzłów planu zapytania, 259
- zimny bufor, 257

 EXPLAIN ANALYZE, 230, 236, 256, 257
 EXPLAIN BUFFERS, 436
 EXPLAIN VERBOSE, 263, 435
 ext2, 88
 ext3, 89, 341

- journal, 90
- ordered, 89, 90
- poziomy księgowania, 89
- writeback, 89, 90

ext4, 91, 341
 ext4 Howto, 91
 Extensible Firmware Interface, 86
 external merge sort, 269
 external SATA, 41
 EXTRACT, 280
 Extreme Memory Profile, 60

F

fałszywa macierz RAID, 46
 FAT32, 87, 107
 fdatasync, 50
 Fiber Channel, 45
 filler, 225
 fillfactor, 244
 FILLFACTOR, 432
 find, 28
 fio, 78
 Firewire, 41
 fizyczna wydajność dysku, 61
 FLUSH CACHE, 95
 FOLLOWING, 437
 Force Unit Access, 95
 forcedirectio, 103
 format CSV dzienników zdarzeń, 197, 198
 Free Space Map, 150, 188
 freebehind, 103
 FreeBSD, 102, 104
 from_collapse_limit, 288
 FrozenXID, 175
 fsck, 87, 88
 FSM, 150, 194

- wyczerpanie mapy, 188

 fstab, 91
 fsync, 50, 77, 95, 114, 162, 418
 fsync_writethrough, 107
 FTS, 250
 FUA, 95
 full_page_writes, 128, 163, 210
 Full-Text Search, 250
 funkcje, 397
 funkcje agregujące, 271
 funkcje Window, 311, 437

G

Generalized Inverted Index, 246
 Generalized Search Tree, 247
 generate_series(), 59, 310
 generowanie wartości hash, 408
 Genetic Query Optimizer, 289, 437
 genetyczny optymalizator zapytań, 289
 GEQO, 289, 437
 geqo_threshold, 289
 getconf, 119
 GIN, 246
 GiST, 247
 global, 110
 Golconde, 390
 gprof, 32, 427
 GPT, 86
 GridSQL, 411
 GROUP BY, 272, 276
 Group, 276
 GUID, 108
 GUID Partition Table, 86

H

harmonogram operacji wejścia-wyjścia, 100
 Hash Join, 285, 286, 300, 302, 304
 Hash Semi Join, 277, 286
 HashAggregate, 272
 HashSetOp, 278
 hashtext(), 408, 409
 hdparm, 52
 hdtune, 65, 68
 Heap Only Tuples, 178, 432
 heap_blks_hit, 233
 HighPoint, 46
 HOT, 173, 178, 319, 432
 hot spare, 112
 Hot Standby, 374, 381, 436
 optymalizacja funkcji, 384
 Hot Swap, 388
 hstore, 250, 439
 htop, 357
 Hyperic HQ, 368

I

identyfikator krotki, 265
 identyfikator obiektu, 125, 265
 idx_blks_hit, 233
 idx_tup_fetch, 322
 idx_tup_read, 322
 ilość miejsca na dysku, 333
 implementacja procesu vacuum, 177
 IN, 277
 indeks, 229, 230, 419
 dane statystyczne, 322
 fillfactor, 244
 indeks bazujący na wyrażeniu, 249
 indeks częściowy, 249
 indeks dla operacji sortowania, 248
 indeks GIN, 246
 indeks GiST, 247
 indeks hash, 246
 indeks wielokolumnowy, 248
 klastrowanie, 243
 łączenie indeksów, 237
 naprawa, 243
 operacje wejścia-wyjścia, 323
 sposoby korzystania z indeksów, 247
 tworzenie, 235, 241
 tworzenie współbieżne, 243
 unikalność indeksu, 242
 indeks B-tree, 37, 191, 245
 klasy operatora tekstowego, 245
 indeksowanie bazy danych, 229
 blok indeksu, 230
 FTS, 250
 klastry, 239, 243
 klucz podstawowy, 233
 liczniki bufora, 241
 pełne skanowanie tabeli, 234
 indeksowanie bazy danych, 229
 planowanie zmiany planu, 238
 ponowne indeksowanie, 244
 rodzaje indeksów, 245
 skanowanie indeksowane, 238
 skanowanie sekwencyjne, 238
 sposoby indeksowania, 230
 wielkość zapytania na dysku, 230

wyszukiwanie, 233
 wyszukiwanie pełnego tekstu, 250
 wyszukiwanie za pomocą nieefektywnego indeksu, 235
 informacje o blokadach, 327, 332
 informowanie o wystąpieniu problemu, 363
 initdb, 109, 119
 INITIALLY DEFERRED, 425
 INITIALLY IMMEDIATE, 425
 INNER JOIN, 289
 Input/Output Per Second, 61
 INSERT, 64, 298, 397, 400
 instalacja modułu contrib
 z kodu źródłowego, 28
 instalacja pg_buffercache, 123
 Intel, 36, 57
 internal, 145
 INTERSECT, 278
 INTERSECT ALL, 278
 INVALID, 243
 IOPS, 61, 68
 iostat, 32, 69, 346, 347, 349, 354
 przykłady dobrej wydajności, 349
 przykłady przeciążonego systemu, 352
 iotop, 349
 iozone, 78
 ipcs, 120
 IS NOT NULL, 437

J

Java, 375
 JBOD, 53
 JBoss, 375
 JDBC, 375, 411
 jednostka FUA, 95
 jednostka pamięci, 118
 jednostka pamięci masowej, 109
 język PL/pgSQL, 424, 438
 JFS, 93
 JOIN, 287
 join_collapse_limit, 287
 JSON, 263
 Just a Bunch of Disks, 53

K

katalogi bazy danych, 109
 kernel.sem, 120
 kill, 146
 klastrowanie indeksu, 243
 fillfactor, 244
 klastry, 239
 klucz podstawowy, 233, 242
 klucze zewnętrzne, 424
 klustysize, 103
 kłamstwa dysku twardego, 49
 kolejność złączeń, 287
 kolekcjoner danych statystycznych, 426
 konfiguracja dysków twardych, 48, 85
 konfiguracja przekazywania danych
 mechanizmu WAL, 383
 konfiguracja rejestracji zdarzeń, 194
 log_line_prefix, 195
 rejestrwanie CSV, 197
 rejestrwanie trudnych zapytań, 199
 syslog, 197
 zapytania obejmujące wiele wierszy, 196
 zbiór dzienników zdarzeń, 195
 konfiguracja serwera, 143
 autovacuum, 152
 checkpoint_completion_target, 156
 checkpoint_segments, 155
 checkpoint_timeout, 156
 commit_delay, 163
 commit_siblings, 163
 constraint_exclusion, 161
 default_statistics_target, 154
 effective_cache_size, 159
 enable_seqscan, 163
 fsync, 162
 full_page_writes, 163
 interakcja z używaną konfiguracją, 144
 kontekst do przeprowadzania zmian, 144
 listen_addresses, 148
 log_line_prefix, 150
 log_min_duration_statement, 152
 log_statement, 151
 maintenance_work_mem, 153

- konfiguracja serwera
 - max_connections, 148
 - max_prepared_transactions, 163
 - optymalizacja ustawień nowego serwera, 164
 - optymalizacje niezalecane, 162
 - pamięć współdzielona, 149
 - pgtune, 166
 - połączenia z bazą danych, 147
 - ponowne wczytywanie pliku konfiguracyjnego, 146
 - punkty kontrolne, 155
 - random_page_cost, 161
 - rejestrwanie zdarzeń, 150
 - shared_buffers, 149
 - synchronous_commit, 160
 - umieszczanie ustawień w komentarzach, 146
 - ustawienia domyślne, 144
 - ustawienia na poziomie klienta, 159
 - ustawienia na poziomie serwera, 147
 - WAL, 156
 - wal_buffers, 156
 - wal_sync_method, 157
 - work_mem, 160
 - wskazówki dotyczące serwerów dedykowanych, 164
 - wskazówki dotyczące serwerów współdzielonych, 165
 - zapytania włączające parametry, 163
 - zerowanie ustawień, 144
 - konfiguracja układu dysków, 124
 - konflikty podczas blokowania rekordów, 171
 - konserwacja, 167
 - kontekst do przeprowadzania zmian, 144
 - kontrola współbieżności, 167
 - kontrolery dysków, 43, 45
 - monitorowanie, 51
 - niezawodność, 48
 - konwersja podzapytania, 277
 - kopia zapasowa, 382
 - koszt odczytu, 260
 - koszt przetworzenia pojedynczego wpisu w indeksie, 260
 - koszt przetworzenia prostego operatora lub funkcji, 260
 - koszty oszacowane, 262
 - koszty rzeczywiste, 262
 - kSar, 359
 - księgowanie, 87, 89
- L**
- Least Recently Used, 132
 - LEFT JOIN, 288, 289
 - libevent, 375
 - liczba obsługiwanych połączeń, 149
 - liczba operacji wejścia-wyjścia na sekundę, 61
 - liczba wykonywanych operacji zatwierdzania, 64
 - fsync, 77
 - liczba operacji zatwierdzania w Windows, 65
 - liczba wykonywanych operacji INSERT, 64
 - test_fsync, 64
 - liczniki bufora, 241
 - liczniki puli połączeń, 372
 - liczniki użycia tabel, 138
 - LIKE, 245
 - Limit, 270
 - LIMIT, 249, 270, 435
 - Linux, 88
 - Linux Software RAID, 44
 - Linux SystemTap, 429
 - LISTEN/NOTIFY, 438
 - listen_addresses, 148
 - listy IN, 277
 - locate, 28, 114
 - log_autovacuum_min_duration, 181, 182
 - log_checkpoints, 130, 341
 - log_destination, 150, 194, 197, 198
 - log_directory, 150, 194, 195
 - log_duration, 199
 - log_filename, 150, 194, 195
 - log_line_prefix, 150, 194, 195, 203, 205, 439
 - log_lock_waits, 332
 - log_min_duration_statement, 152, 199, 203
 - log_min_messages, 182
 - log_statement, 151, 199
 - log_temp_files, 111, 269, 297, 439
 - logging_collector, 194, 195, 198, 203

Londiste, 387
 LRU, 132
 LSI MegaRAID, 45
 LVM, 82, 95

Ł

łączenie indeksów, 237, 267
 łączenie zbiorów rekordów, 265

- identyfikator krotki, 265
- identyfikator obiektu, 265
- mapa bitowa skanowania indeksu, 267
- skanowanie indeksu, 266, 267
- skanowanie sekwencyjne, 266

M

macierz RAID, 38, 112

- DDF, 44
- falszywa macierz RAID, 46
- JBOD, 53
- konfiguracja dysków, 112
- powody niskiej wydajności, 81
- programowa macierz RAID, 44
- sprzętowa macierz RAID, 44
- wskazówki dotyczące konfiguracji dysków, 114

 maintenance_work_mem, 153, 154, 164, 186, 187, 417
 maksymalna wielkość systemu plików, 85
 Mammoth Replicator, 390
 Management Information Base, 369
 mapa bitowa skanowania indeksu, 267, 268
 mapowanie sposobu widzenia fizycznych dysków przez system, 82
 Master Boot Record, 86
 materializacja, 279
 materializacja złączenia Merge Join, 284
 Materialize, 437
 MAX(), 271, 431
 max_connections, 121, 148, 324
 max_fsm_pages, 150, 153, 165, 181, 189, 434
 max_fsm_relations, 150, 189, 434
 max_locks_per_transaction, 121
 max_prepared_transactions, 121, 163
 max_standby_archive_delay, 385
 max_standby_streaming_delay, 385
 maxphys, 103
 MBR, 86
 MCV, 286
 mechanizm kontroli współbieżności, 167
 mechanizm WAL, 382
 MegaRAID, 45
 memcached, 376
 memtest86+, 56
 Menedżer zadań, 360
 menedżery kolejki replikacji, 386
 Merge Join, 280, 283, 304

- materializacja złączenia, 284

 metadane systemu plików, 86
 metody replikacji, 391
 metody wczytywania danych, 416
 miękkie dowiązania symboliczne, 109
 migracja partycjonowanej używanej tabeli, 401
 MIN(), 271, 431
 mirroring, 39
 mk-query-digest, 206
 model MVCC, 325
 model wykonania zstępującego, 270
 moduły contrib, 27

- instalacja z kodu źródłowego, 28
- pg_buffercache, 29
- używanie modułu, 29
- wyszukiwanie modułów, 28

 modyfikacja parametrów bazy danych, 144
 MongoDB, 26
 Monitor systemu Windows, 360

- liczniki, 361
- zapisywanie danych, 362

 monitorowanie, 343, 439

- baza danych, 315
- dane, 193
- dzienniki zdarzeń zapytań, 194
- kontroler dyskowy, 51
- monitorowanie pod kątem przerwania usług, 363
- poziom wykorzystania pamięci, 358
- proces demona autovacuum, 182
- strony indeksu, 193

 Most Common Values, 286
 MRTG, 364
 Multi Router Traffic Grapher, 364

Multiversion Concurrency Control, 167
 Munin, 366
 MVCC, 167, 325
 tryb Read Committed, 174
 wady mechanizmu, 174
 zalety mechanizmu, 174
 MySQL, 26

N

n_distinct_inherited, 295
 n_tup_hot_upd, 179
 n_tup_upd, 179
 nadmuchane indeksy, 191
 pomiar nadmuchania indeksu, 191
 Nagios, 193, 364
 PostgreSQL, 365
 rejestrwanie liczników, 364
 Windows, 366
 napędy SSD, 42
 naprawa bazy danych po awarii, 128
 dziennik zapisu z wyprzedzeniem, 128
 punkty kontrolne, 128
 naprawa indeksu, 243
 narzędzia monitorujące
 system Unix, 343
 system Windows, 360
 narzędzia służące do analizy planu, 264
 narzędzia SNMP, 369
 NAS, 46
 natężenie operacji wejścia-wyjścia, 130
 Native Command Queuing, 95
 NCQ, 95
 Nested Loop, 278, 280
 Network Attached Storage, 46
 niewyjaśnione operacje zapisu, 422
 niezawodne kontrolery, 48
 niezawodność dysku twardego, 41
 No Operation, 100
 noatime, 108
 nodiratime, 97
 noop, 100
 NOT DEFERRABLE, 425
 NOT EXISTS, 286
 NOT NULL, 242

NTFS, 87, 107
 dostrajanie zachowania operacji
 montowania systemu plików, 108
 null, 242
 numerowanie rekordów, 309

O

Object Identification Number, 265
 Object Identifier, 125
 Object-Relational Mapper, 437
 obsługa barier zapisu
 napęd, 94
 system plików, 95
 obsługa błędów na dyskach, 40
 oczekiwana wydajność dysku, 80
 oczekiwanie na blokadę tabeli, 331
 oczekiwanie na blokadę transakcji, 330
 odczyt z wyprzedzeniem, 96
 odporność na awarie, 383
 odtworzenie serwera na podstawie plików
 WAL innego serwera, 382
 odzyskiwanie danych
 po awarii systemu plików, 86
 OFFSET, 271, 306, 307, 435
 ograniczenia, 419
 ograniczenia SQL, 309
 ogromne przeciążenie klucza
 zewnętrznego, 424
 ogromne przeciążenie mechanizmu
 zbierającego dane statystyczne, 426
 OID, 125, 265
 oid2name, 125, 126
 okno, 311
 określanie wielkości bufora zapisu, 98
 OLTP, 22, 105, 226
 OmniPITR, 383
 Online Transaction Processing, 22, 105
 OOM, 441
 operacja bulk-loading, 415
 operacja czyszczenia na podstawie
 kosztów, 179
 operacja sync, 60
 operacje ustawiania, 278
 operacje wejścia-wyjścia indeksu, 323

- operacje wejścia-wyjścia tabel, 320
 - operacje zapisu, 422
 - OProfile, 32, 428
 - oprogramowanie, 30
 - oprogramowanie firmware, 42
 - oprogramowanie trendów, 362, 363
 - optymalizacja wydajności, 32
 - Hot Standby, 384
 - konfiguracja serwera, 143
 - operacja bulk-loading, 417
 - optymalizacja z użyciem danych statystycznych dotyczących zapisu w tle, 339
 - pojemność dysku twardego, 63
 - ustawienia nowego serwera, 164
 - optymalizacja zapytań, 253
 - constraint_exclusion, 298
 - cursor_tuple_fraction, 298
 - dane statystyczne, 290
 - DELETE, 298
 - effective_cache_size, 295
 - EXPLAIN, 256
 - INSERT, 298
 - koszty oszacowane, 262
 - koszty rzeczywiste, 262
 - łączenie zbiorów rekordów, 265
 - obliczanie kosztu, 260
 - OFFSET, 306
 - ograniczenia SQL, 309
 - poprawianie zapytań, 299
 - poszukiwanie odpowiednika zapytania, 300
 - przetwarzanie węzłów, 268
 - rozwiązywanie błędów optymalizatora, 305
 - SELECT, 298
 - struktura węzłów planu zapytania, 259
 - unikanie planu restrukturyzacji, 306
 - UPDATE, 298
 - w pełni buforowane zbiory danych, 300
 - work_mem, 297
 - wyłączanie funkcji optymalizatora, 301
 - zewnętrzne źródła problemów, 309
 - złączenia, 281
 - optymalizator GEQO, 289
 - ORDER BY, 248, 249, 268, 270
 - ORM, 437
 - overcommit, 98
- P**
- Pagila, 254
 - pamięć, 37
 - DDR SDRAM, 57
 - pamięć dla bufora bazy danych, 117
 - pamięć współdzielona, 149
 - SPD, 60
 - testy wydajności, 55
 - tryb dwukanałowy, 60
 - XMP, 60
 - parametry konfiguracyjne, 143
 - parametry planowania zapytania, 295
 - partition, 434
 - partycje, 405
 - partycjonowanie danych, 393
 - błędy, 407
 - tworzenie partycji, 405
 - zalety partycjonowania, 406
 - partycjonowanie o zasięgu tabeli, 393
 - lista partycjonowania, 395
 - migracja partycjonowanej używanej tabeli, 401
 - plany zapytań dla pustej partycji, 399
 - pole klucza używanego do partycjonowania, 394
 - przekierowywanie poleceń INSERT do partycji, 397
 - reguły partycjonowania, 398
 - tworzenie partycji, 396, 405
 - uaktualnianie wyzwalacza, 400
 - wielkości partycji, 395
 - zapytania partycjonowane, 403
 - zmiana daty, 400
 - partycjonowanie poziome, 408
 - generowanie wartości hash, 408
 - GridSQL, 411
 - PL/Proxy, 408
 - Sharding, 410
 - parzystość, 38
 - pdflush, 98
 - pełne skanowanie tabeli, 234
 - PERC, 45
 - perfmn, 360
 - pętle zagnieżdżone, 281
 - materializacja złączenia Merge Join, 284
 - wewnętrzne skanowanie indeksu, 282

- pg_archivecleanup, 436
- pg_autovacuum, 181
- pg_buffercache, 29, 37, 122, 125, 134, 201, 340, 433
 - instalacja, 123
- pg_buffercache.sql, 28
- pg_bulkload, 417
- pg_class, 125
- pg_clog, 110, 133, 326, 422
- pg_column_size(), 333
- pg_ctl, 146, 150, 439
- pg_database, 125
- pg_database_size(), 333
- pg_default, 109
- pg_dump, 23
- pg_dumpall, 23
- pg_freemap, 194
- pg_hba.conf, 148, 371
- pg_indexes_size(), 334, 439
- pg_last_xlog_receive_location(), 436
- pg_last_xlog_replay_location(), 436
- pg_locks, 318, 326, 327, 328, 330, 332
- pg_migrator, 24
- pg_multixact, 110
- pg_prepared_xacts, 318
- pg_relation_size(), 139, 333, 334
- pg_reload_conf(), 146
- pg_restore, 24, 435
- pg_settings, 118, 144, 147, 336
- pg_size_pretty(), 333
- pg_sleep(), 231
- pg_start_backup, 382
- pg_stat_activity, 188, 318, 324, 325, 328, 330, 372, 432, 433, 438
- pg_stat_all_tables, 181, 318
- pg_stat_bgwriter, 130, 132, 134, 316, 317, 318, 335, 336, 337, 340, 363, 426, 432
- pg_stat_database, 317, 324
- pg_stat_reset(), 233, 317
- pg_stat_reset_shared(), 317, 439
- pg_stat_reset_single_function_counters(), 317, 427, 439
- pg_stat_reset_single_table_counters(), 317, 427, 439
- pg_stat_statements, 201, 202, 435, 440
- pg_stat_sys_tables, 182, 318
- pg_stat_tables, 141
- pg_stat_tmp, 110
- pg_stat_user_functions, 435
- pg_stat_user_indexes, 322, 323
- pg_stat_user_tables, 131, 179, 182, 190, 230, 316, 318, 324, 426, 432, 433
- pg_statio_user_tables, 230, 320, 324
- pg_stats, 290
- pg_stop_backup, 382
- pg_subtrans, 110, 422
- pg_table_size(), 139, 334, 439
- pg_tblspc, 110
- pg_total_relation_size(), 139, 334
- pg_twophase, 110
- pg_upgrade, 24, 25, 440
- pg_xlog, 103, 110, 155
- pgAdmin III, 184, 262
- pgbench, 28, 59, 137, 201, 209, 344, 433, 439
 - analiza opóźnienia, 219
 - definicja skryptu zapytania, 211
 - definicja tabeli, 210
 - inicjalizacja tabel, 211
 - konfiguracja serwera bazy danych, 213
 - ograniczenia programu, 224
 - powody otrzymywania błędnych wyników i różnic, 222
 - programistyczne wersje PostgreSQL, 223
 - skrypt transakcji, 211
 - test przeprowadzający jedynie zapytania SELECT, 217
 - test szybkości wstawiania danych, 225
 - test transakcji TPC-B-like, 218
 - testy domyślne, 209
 - uruchamianie, 214
 - użyteczność wyników, 223
 - wątki worker, 224
 - własne testy, 225
 - wykrywanie skali wielkości bazy danych, 210
 - wyniki graficzne, 216
 - wyniki testów, 217
- pgbench_accounts, 141, 210, 212, 319
- pgbench_accounts_key, 140
- pgbench_accounts_pkey, 141
- pgbench_branches, 210, 212, 319, 320
- pgbench_history, 210, 212, 320, 321

- pgbench_tellers, 210, 319, 320
- pgbench-tools, 216
 - konfiguracja, 216
- pgBouncer, 374
- PgCluster, 390
- pgfincore, 134
- pgFouine, 189, 203
 - format danych wyjściowych, 204
- pgFoundry, 30
- pgiosim, 78
- pgloader, 416, 417
- pgmemcache, 377
- pgpool, 389
- pgpool-II, 373, 389
 - równoważenie obciążenia dla skalowania replikacji, 374
- pgsi, 206
- pgstat_bgwriter, 130
- pgstat_bgwriter.buffer_alloc, 131
- pgstat_bgwriter.checkpoints_req, 129
- pgstat_bgwriter.checkpoints_timed, 129
- pgstatspack, 367
- pgstattuple, 194
- pgtune, 166
- PITR, 382, 419
- PITRtools, 383
- PL/pgSQL, 424, 438
- PL/Proxy, 408
 - skalowanie, 410
- plan zapytania, 256
 - plan zapytań dla pustej partycji, 399
 - węzły, 259
- plik dziennika zdarzeń, 200
- pliki tymczasowe, 111
- podtrzymywany bateryjnie bufor zapisu, 50
- podwójnie buforowane dane, 133
- Point-in-time recovery, 382
- pole kontekstu, 145
- polecenia składowane, 423
- połączenia z bazą danych, 147, 324
- pomiar nadmuchiwanie indeksu, 191
- porzucenie optymalizacji
 - mechanizmu WAL, 418
- ponowne indeksowanie, 244
- ponowne wczytywanie pliku
 - konfiguracyjnego, 146
- poprawianie zapytań, 299
- postgres, 196
- PostgreSQL, 21
 - uaktualnienie bezpośrednio, 24
 - uaktualnienie do nowszej głównej wersji, 23
 - wersje, 23, 429
 - wydajność wydań, 22
- PostgreSQL 8.1, 22, 431
- PostgreSQL 8.2, 432
- PostgreSQL 8.3, 25, 432
- PostgreSQL 8.4, 434
- PostgreSQL 9.0, 24, 436
- PostgreSQL System Impact, 206
- postgres.conf, 118, 143, 439
 - komentarze, 146
- postgres-contrib, 28
- Postgres-XC, 390
- postmaster, 145
- poszukiwanie odpowiednika zapytania, 300
- PQA, 205
- PRECEDING, 437
- PREPARE, 423
- PREPARE TRANSACTION, 318
- PRIMARY KEY, 232
- problemy związane z wydajnością, 421
 - wersje PostgreSQL, 429
- procesor, 36
 - testy wydajności, 55, 59
- profilowanie, 32
 - dyskowe operacje wejścia-wyjścia, 349
- profilowanie bazy danych, 427
 - DTrace, 428
 - gprof, 427
 - OProfile, 428
 - Visual Studio, 428
- programowa macierz RAID, 44
- programowanie bazy danych, 438
- programowanie CTE, 434
- projekt dbt, 227
- projekty replikacji, 389
- Promise, 46
- protokoły
 - SMART, 40
 - SNMP, 363
 - Streaming Replication, 384
 - XMP, 60
- prstat, 356

- przeciążenie
 - klucz zewnętrzny, 424
 - mechanizm zbierający dane
 - statystyczne, 426
 - punkt kontrolny, 134
 - przegląd wielkości bufora, 141
 - przegląd zawartości bufora, 137
 - przekierowywanie poleceń INSERT
 - do partycji, 397
 - przetwarzanie punktów kontrolnych, 128
 - przetwarzanie węzłów, 268
 - Aggregate, 271
 - Append, 275
 - CTE, 280
 - Gruop, 276
 - HashAggregate, 272
 - Limit, 270
 - Nested Loop, 278, 280
 - OFFSET, 271
 - operacje ustawiania, 278
 - Result, 274
 - Sort, 268
 - sortowanie, 268
 - Subplan, 277
 - Subquery Scan, 277
 - Unique, 273
 - WindowAgg, 274
 - przywracanie równowagi, 419
 - przywrócenie do pewnego punktu w czasie, 382
 - ps, 355, 426
 - pset, 231
 - psql, 59
 - pula połączeń, 371
 - liczba połączeń, 372
 - liczniki, 372
 - pgBouncer, 374
 - pgpool-II, 373
 - serwer aplikacji, 375
 - punkty kontrolne, 128
 - aktywność tworzenia, 335
 - natężenie operacji wejścia-wyjścia, 130
 - przeciążenie, 134
 - rozproszenie, 130
 - tworzenie, 129
 - ustawienia, 155
- ## Q
- QUERY EXECUTE...USING, 423
 - quicksort, 269
 - quote_literal, 424
- ## R
- RAID, 38, 112
 - RAID 0, 39
 - RAID 1, 39, 112
 - RAID 1+0, 39
 - RAID 10, 39
 - RAID 5, 39
 - RAID 6, 39
 - RAM, 37
 - Random Access, 68
 - random_page_cost, 161, 239, 260, 262, 300
 - Read Committed, 171
 - Reconnoiter, 368
 - redirect_stderr, 195
 - REDO, 48
 - Redundant Array of Independent Disks, 38
 - reguły partycjonowania, 398
 - REINDEX, 125, 189, 192, 243, 320, 406, 440
 - ReiserFS, 93
 - rejestrwanie CSV, 197
 - rejestrwanie danych wydajności, 363
 - rejestrwanie informacji o blokadach, 332
 - rejestrwanie trudnych zapytań, 199
 - rejestrwanie zdarzeń, 150, 194
 - zdarzenia procesu demona
 - autovacuum, 182
 - relatime, 97
 - replikacja, 24, 381, 388, 436
 - Bucardo, 388
 - Hot Standby, 381, 384
 - konfiguracja za pomocą danych
 - mechanizmu WAL, 383
 - Londiste, 387
 - menedżery kolejki replikacji, 386
 - pgpool-II, 389
 - projekty replikacji, 389
 - skalowanie odczytu, 388
 - Slony, 387
 - Streaming Replication, 384

RESET, 144
 reset_val, 144
 resource fork, 334
 Result, 274
 ROLLBACK, 170, 176, 401
 Round Robin Database tool, 364
 row_number(), 311
 rozgrzany bufor, 257
 rozproszenie punktów kontrolnych, 130
 rozproszenie użycia bufora, 140
 rozwiązywanie błędów optymalizatora, 305
 rozwiązywanie problemów związanych
 z wydajnością, 33
 równoważenie obciążenia pgpool-II dla
 skalowania replikacji, 374
 RPM, 38
 RRDtool, 364
 Rubyrep, 390
 runtime(), 328

S

SAN, 46, 47
 sar, 357
 graficzna prezentacja danych, 359
 SAS, 38
 SATA, 38
 SATA RAID, 45
 schemat partycjonowania, 86
 segmap_percent, 103
 segmapsize, 103
 sekwencyjny dostęp, 61
 SELECT, 59
 semafore jądra, 120
 Semi Join, 307
 Seq Scan, 266, 270
 seq_page_cost, 260, 261, 300, 432
 Serial ATA, 38
 Serial Attached SCSI, 38
 Serial Presence Detect, 30
 serializacja, 172
 serwer dedykowany, 164
 serwer współdzielony, 365
 serwer zapasowy, 382
 session_line_num, 198
 SET, 269
 SET CONSTRAINTS, 425
 Sharding, 410
 Shared Nothing, 410
 shared_buffers, 32, 117, 121, 134, 135, 149,
 159, 164, 255, 295, 320, 336, 340, 341, 440
 shared_preload_libraries, 200
 short stroking, 63, 67
 SHOW, 118, 147
 sighthup, 145, 146
 SIGHUP, 146
 Simple Network Management Protocol, 363
 single-channel, 60
 skala wielkości bazy danych, 210
 skalowanie odczytu, 388
 skalowanie replikacji, 374
 skalowanie za pomocą PL/Proxy, 410
 skalowanie za pomocą replikacji, 381
 skanowanie CTE, 280
 skanowanie indeksu, 238, 266, 267
 skanowanie mapy bitowej indeksu, 434
 skanowanie sekwencyjne, 238, 266
 Slony, 374, 387
 SMART, 40
 SN, 410
 SNMP, 363
 narzędzia, 369
 SNMP MIB, 369
 soft updates, 104
 Solaris, 102
 Solid State Drive, 42
 Sort, 268
 sortowanie, 248
 sortowanie zewnętrzne przez scalanie, 269
 SPD, 60
 spindle, 61, 62
 sposoby zapisu zmodyfikowanych bloków, 132
 sprawdzanie wydajności bazy danych, 209
 sprzęt, 35
 dyski twarde, 37
 konfiguracja dysków, 48
 kontrolery dysków, 43
 pamięć, 37
 procesor, 36
 testy wydajności, 55
 wydatki na zakup sprzętu, 35
 sprzętowa macierz RAID, 44
 SQL Injection, 423
 SQLite, 26

- SSD, 42
- Standby, 382
- Staplr, 369
- Statistic Collector, 315
- stats_block_level, 182
- stats_reset_on_server_start, 317
- stats_row_level, 153, 182
- stats_start_collector, 153, 182
- stats_temp_directory, 435
- STDDEV(), 271
- Storage Area Network, 46
- STREAM, 56
 - platformy sprzętowe, 57
- Streaming Replication, 384, 385
- stream-scaling, 57
- stripe, 82
- striped z parzystością, 39
- striping, 39
- struktura węzłów planu zapytania, 259
- Subplan, 277
- Subquery Scan, 277
- SUM(), 271
- superuser, 145
- superuser_reserved_connections, 153
- swappiness, 98
- swapping, 98
- sygnał SIGHUP, 146
- symlink, 108
- SYNCHRONIZE CACHE, 95
- synchronous_commit, 160, 418, 432
- sysbench, 65, 75
 - czas wyszukiwania, 76
 - liczba operacji zatwierdzenia za pomocą fsync, 77
- sysctl, 120
- sysctl.conf, 98, 120
- Sysinternals, 360
- syslog, 197
- sysstat, 357, 358
- system plików, 85
 - bariery zapisu, 95
 - Btrfs, 93
 - buforowanie odczytu, 98
 - czas dostępu do plików, 97
 - dostrajanie, 96
 - ext2, 88
 - ext3, 89
 - ext4, 91
 - FAT32, 107
 - FreeBSD, 102, 104
 - JFS, 93
 - księgowanie, 87, 88, 89
 - Linux, 88
 - maksymalna ilość danych, 85
 - metadane, 86
 - naprawa, 87
 - NTFS, 107
 - odczyt z wyprzedzeniem, 96
 - odzyskiwanie danych po awarii, 86
 - określanie wielkości bufora zapisu, 98
 - ReiserFS, 93
 - Solaris, 102
 - swapping, 98
 - UFS, 102
 - UFS2, 104
 - winda harmonogramu operacji wejścia-wyjścia, 100
 - Windows, 107
 - XFS, 91, 96
 - ZFS, 105
- szacowanie wielkości pamięci współdzielonej, 121
- szybkie sortowanie, 269

Ś

średni czas dostępu, 62

T

- tabele, 318
 - dane statystyczne, 318
 - operacje wejścia-wyjścia, 320
 - partycjonowanie danych, 393
- table_stats, 233
- tablespace, 109
- technika HOT, 173, 178
- technika TOAST, 139
- temp_tablespaces, 111, 433
- test_fsyc, 64, 65
- testowanie pamięci, 56
- testowanie wydajności dysku, 65

testy OLTP, 22
 testy szybkości wstawiania danych, 225
 testy TPC-H, 26
 testy transakcji TPC-B-like, 218
 testy wydajności, 32, 227
 PL/pgSQL, 424
 Transaction Processing Performance Council, 226
 testy wydajności sprzętu, 55
 dysk, 61, 77
 memtest86+, 56
 pamięć, 55
 procesor, 55, 59
 TEXT, 25
 The Database Test, 227
 TID, 265
 timeofday(), 325
 timing, 59
 TOAST, 139, 321, 334
 Tomcat, 375
 top, 355
 zamienniki, 356
 top-down, 270
 TPC, 209, 227
 TPC-B, 209
 TPC-B (sort of), 211
 TPC-B-like, 211, 218, 223
 TPC-C, 227
 TPC-H, 26, 227
 TPS, 22, 219, 344
 track_functions, 424
 transaction id, 326
 Transaction Per Second, 22
 Transaction Processing Performance Council, 209, 226
 Transactions Per Second, 219, 344
 transakcje, 433
 transakcje składowane, 122, 318
 transakcje wirtualne, 326
 trendy, 362, 363
 przechowywanie historycznych danych trendów, 363
 TRUNCATE, 125, 419
 trwały identyfikator, 326
 tryb JBOD, 53

tworzenie
 blok w bazie danych, 126
 indeks, 235, 241
 przykładowe dane, 232
 punkty kontrolne, 129
 tworzenie partycji, 396, 405
 tworzenie dynamiczne, 406
 tworzenie zgodnie z harmonogramem, 405
 txid_current(), 168, 176, 433
 txid_current_snapshot(), 168, 433

U

uaktualnianie wyzwalacza, 400
 uaktualnienia, 169
 uaktualnienie do nowszej głównej wersji, 23
 uaktualnienie bezpośrednie, 24
 uaktualnienie do PostgreSQL 8.3, 25
 uaktualnienie mniej znaczących wersji, 25
 UFS, 102
 ufs:freebehind, 103
 UFS1, 102
 UFS2, 104
 unikalność indeksu, 242
 UNION, 275
 Unique, 273
 UNIQUE, 425
 Unix File System, 102
 UPDATE, 170, 171, 172, 173, 176, 200, 201, 298, 299
 update_process_title, 355
 uruchamianie
 pgbench, 214
 proces demona autovacuum, 183
 urządzenia, 347
 NAS, 46, 47
 SAN, 46, 47
 USB, 41
 user, 145
 ustawienia na poziomie klienta, 159
 usunięcia, 173
 usuwanie złączeń, 288
 uuid-osp, 29
 użycie pamięci przez wyzwalacz, 425
 używanie modułu contrib, 29

V

vacuum, 176, 179, 180
 autovacuum, 181
 błędy spowodowane brakiem pamięci, 187
 HOT, 178
 implementacja procesu, 177
 odzyskanie wolnego miejsca na dysku, 177
 operacja czyszczenia na podstawie kosztów, 179
 problemy, 185
 proces vacuum, 177
 strony brudne, 180
 strony pominięte, 180
 strony trafione, 180
 VACUUM, 150, 152, 435
 VACUUM ANALYZE, 232
 VACUUM FULL, 178, 185, 189, 191, 192, 433, 440
 VACUUM VERBOSE, 150, 189
 vacuum_cost_delay, 180
 vacuum_cost_page_dirty, 180
 vacuum_cost_page_hit, 180
 vacuum_cost_page_miss, 180
 vacuum_defer_cleanup_age, 385
 vacuum_freeze_max_age, 175
 vacuum_freeze_min_age, 175, 418, 420
 VARIANCE(), 271
 vfs.hirunningspace, 104
 vfs.read_max, 104
 Visual Studio, 428
 vm.overcommit_memory, 98
 vm.swappiness, 98
 vmstat, 32, 69, 344, 346

W

WAL, 48, 49, 88, 90, 128, 338, 347, 382
 konfiguracja przekazywania danych, 383
 ustawienia, 156
 wal_block_size, 121
 wal_buffers, 118, 121, 156, 165, 418
 wal_level, 436
 wal_sync, 158
 wal_sync_method, 50, 51, 65, 157, 158, 165
 wal_writer_delay, 160

walmgr, 383
 wartości hash, 408
 wartości null, 242
 wąskie gardło, 32
 wczytywanie pliku konfiguracyjnego, 146
 wersje PostgreSQL, 21, 429
 wewnętrzne skanowanie indeksu, 282
 węzły planu zapytania, 259
 WHERE, 171, 266, 281, 287, 407
 widoczność transakcji, 167
 cykl życiowy widoczności rekordu, 168
 konflikty podczas blokowania rekordów, 171
 serializacja, 172
 uaktualnienia, 169
 usunięcie, 173
 vacuum, 176
 wewnętrzne mechanizmy określające widoczność, 168
 xmax, 168
 xmin, 168
 zerowanie identyfikatora transakcji, 174
 widoki danych statystycznych, 315
 pg_stat_bgwriter, 316
 pg_stat_user_tables, 316
 widoki kumulacyjne, 317
 widoki zmateralizowane, 427
 Widoki żywe, 317
 wielkość zapytania na dysku, 230
 winda harmonogramu operacji
 wejścia-wyjścia, 100
 Window, 311, 434, 437
 kumulowanie wyniku, 311
 numerowanie, 311
 WindowAgg, 274
 Windows, 107
 WITH, 434
 WITH RECURSIVE, 434
 wolne wykonywanie funkcji i poleceń składowanych, 423
 work_mem, 111, 117, 160, 269, 296, 297, 418
 Write-Ahead Log, 48, 156, 382
 writeback, 107
 written_per_sec, 339, 340
 współbieżne tworzenie indeksu, 243
 wybór wersji PostgreSQL, 23

wyczerpanie mapy Free Space Map, 188
 wydajność bazy danych, 209
 wydajność bufora bez wstrzymywania zapisu, 52
 wydajność dysku, 61
 wydajność wydań PostgreSQL, 22
 wydajność zapytań IN, 277
 wydatki na zakup sprzętu, 35
 wykorzystanie dysku, 333
 wykorzystanie pamięci, 358
 wykrywanie skali wielkości bazy danych, 210
 wyłączenie

- bufor zapisu w napędzie, 52
- funkcje optymalizatora, 301

 wymuszanie kolejności złączeń, 287
 wyniki testów pgbench, 217
 wyszukiwanie, 233

- wyszukiwanie za pomocą nieefektywnego indeksu, 235

 wyszukiwanie modułów contrib, 28
 wyzwalacze, 397, 438

- AFTER, 425
- użycie pamięci, 425

X

XFS, 91, 96, 341
 XID, 168, 175, 326, 327
 xmax, 168, 173, 174
 xmin, 168, 174
 XML, 263
 XMP, 60

Y

YAML, 263

Z

zakleszczenia, 332
 zakup sprzętu, 35
 zapis migawek pg_stat_bgwriter, 337
 zapis w tle, 132, 335, 339
 zapis zmodyfikowanych bloków na dysku, 127
 zapytania, 253

- zapytania obejmujące wiele wierszy, 196
- zapytania partycjonowane, 403

zbiór dzienników zdarzeń, 195
 ZCAV, 63
 Zenoss, 368
 zerowanie danych statystycznych, 426
 zerowanie identyfikatora transakcji, 174
 zewnętrzne dyski twarde, 41
 zewnętrzne programy do wczytywania danych, 416
 ZFS, 44, 105
 zfs_nocacheflush, 106
 zimny bufor, 257
 zliczanie rekordów, 421
 złączenia, 281

- Anti Join, 286
- genetyczny optymalizator zapytań, 289
- GEQO, 289
- Hash Join, 285, 300, 302
- Hash Semi, 277, 286
- INNER JOIN, 289
- kolejność złączeń, 287
- LEFT JOIN, 289
- Merge Join, 280, 283
- pętle zagnieżdżone, 281
- pętle zagnieżdżone wraz z wewnętrznym skanowaniem indeksu, 282
- Semi Join, 307
- usuwanie złączeń, 288
- wymuszanie kolejności złączeń, 287
- złączenia zewnętrzne, 437

 zmaterializowane widoki, 427
 znormalizowane ślady wykonywanych zapytań, 201
 rzucanie zawartości bazy danych do pliku, 24, 25
 zwiększenie parametrów pamięci współdzielonej w systemie Unix, 119

Zoptymalizuj swój serwer PostgreSQL

i unikaj problemów, które mogą zmniejszyć jego wydajność!

Mający za sobą już ponad piętnaście lat rozwoju PostgreSQL jest dziś potężnym systemem baz danych typu open source, o sprawdzonej architekturze i reputacji narzędzia niezawodnego oraz nieprzeciętnie wydajnego. Współdziała on ze wszystkimi popularnymi systemami operacyjnymi i jest w pełni zgodny z warunkami ACID. Te zalety sprawiają, że można go używać jako magazynu danych dla aplikacji oraz jako bazy danych dla aplikacji sieciowych. Jednak osiągnięcie maksymalnej wydajności PostgreSQL nie jest wcale zadaniem łatwym, a w trakcie korzystania z jego serwerów można napotkać powtarzające się trudności, zwłaszcza gdy wzrasta obciążenie serwera, a wymagania stają się coraz większe. Jeśli zatem nie chcesz tygodniami dochodzić do właściwych rozwiązań swoich problemów – oto książka, w której znajdziesz całą potrzebną Ci wiedzę.

Masz w rękach kompletny podręcznik, przeznaczony dla średnio i bardzo zaawansowanych administratorów baz danych, którzy już używają PostgreSQL lub dopiero zamierzają to zrobić. Najpierw zapoznasz się z najnowszymi wersjami tej platformy oraz dowiesz się, jak dobrać komponenty serwera, aby optymalnie wykorzystać możliwości systemu. Zobaczysz, jak testować wydajność sprzętu dla bazy danych oraz konfigurować dyski i system plików, aby zwiększać ich efektywność. Poznasz także parametry, których zmiana może powodować problemy, a ponadto najważniejsze ustawienia, ich znaczenie i zasady prawidłowego stosowania. Przeczytasz o tym, jak uzyskać użyteczne wyniki testów wydajności, a także o skutecznym indeksowaniu bazy danych, optymalizacji zapytań i partycjonowaniu danych na podzbiory. Na koniec dowiesz się, jak unikać najczęściej spotykanych problemów i rozwiązywać je, gdy już się pojawią.

Dzięki tej książce:

- poznasz najlepsze praktyki pozwalające na obsłużenie wymagających aplikacji
- odkryjesz, dlaczego sprzęt komputerowy nadaje się (lub nie) dla wysoko wydajnych aplikacji bazodanowych
- zrozumiesz, na czym polegają kompromisy związane z szybkością i niezawodnością działania
- zoptymalizujesz system operacyjny, aby osiągnąć najlepszą wydajność bazy danych
- przeprowadzisz testy wydajności całego systemu, od sprzętu komputerowego po aplikację
- przeanalizujesz rzeczywiste przykłady, co pozwoli Ci poznać wpływ różnych ustawień parametrów serwera na wydajność
- będziesz skutecznie monitorować zdarzenia zachodzące na serwerze, zarówno w bazie danych, jak i poza nią
- znajdziesz najlepsze dodatki, rozszerzające podstawowe możliwości bazy danych PostgreSQL
- dowiesz się, jak przygotować replikację systemów za pomocą najnowszych funkcji wprowadzonych w PostgreSQL 9.0

W katalogowy 0186



Księgarnia Internetowa:
<http://helion.pl>



Zamówienia telefonicznie:
0 801 339900



0 601 339900

helion.pl
księgarnia
internetowa

Sprawdź najnowsze promocje:
• <http://helion.pl/promocje>
Książki najchętniej czytane:
• <http://helion.pl/wszystkie>
Zamów informacje o nowościach:
• <http://helion.pl/nowosci>



Helion

Helion SA
ul. Bolesława Śc. 44-100 Gliwice
tel.: 32 230 98 43
e-mail: helion@helion.pl
<http://helion.pl>

Cena: 79,00 zł

ISBN 978-83-246-3062-2



Informatyka w najlepszym wydaniu