

Содержание

Об авторе	11
О рецензенте	12
Предисловие	13
Глава 1. Обзор PostgreSQL	16
Что нового в PostgreSQL 11.0?	16
Новые средства администрирования базы данных	17
Усовершенствования в индексировании и оптимизации	18
Улучшенное управление кешем	20
Улучшенные оконные функции	20
Добавление JIT-компиляции	21
Улучшенное секционирование	21
Поддержка хранимых процедур	22
Улучшение команды ALTER TABLE	23
Резюме	24
Вопросы	24
Глава 2. Транзакции и блокировка	25
Работа с транзакциями в PostgreSQL	25
Обработка ошибок внутри транзакции	27
Использование команды SAVEPOINT	28
Транзакционные DDL-команды	29
Основы блокировки	30
Предотвращение типичных ошибок и явная блокировка	31
Использование фраз FOR SHARE и FOR UPDATE	33
Уровни изоляции транзакций	36
SSI-транзакции	38
Взаимоблокировки и смежные вопросы	38
Рекомендательные блокировки	40
Оптимизация хранилища и управление очисткой	41
Настройка VACUUM и autovacuum	41
Наблюдение за работой VACUUM	43
Ограничение длительности транзакций с помощью времени жизни снимка	46
Резюме	47
Вопросы	47
Глава 3. Использование индексов	49
Простые запросы и стоимостная модель	49
Использование команды EXPLAIN	50

Стоимостная модель PostgreSQL.....	52
Развертывание простых индексов	54
Сортировка результатов	55
Эффективное использование просмотра по битовой карте	57
Разумное использование индексов	57
Повышение быстродействия с помощью кластеризованных таблиц.....	59
Кластеризация таблиц	62
Просмотр только индекса.....	62
Дополнительные свойства B-деревьев.....	63
Комбинированные индексы	63
Добавление функциональных индексов	64
Уменьшение занятого места на диске	65
Добавление данных во время индексирования.....	66
Введение в классы операторов	67
Создание класса операторов для B-дерева	68
Типы индексов в PostgreSQL.....	73
Хеш-индексы	73
GiST-индексы.....	73
GIN-индексы.....	76
SP-GiST-индексы	77
BRIN-индексы.....	77
Добавление новых типов индексов	79
Получение более точных ответов с помощью нечеткого поиска.....	80
Расширение pg_trgm и его достоинства	80
Ускорение запросов с предикатом LIKE	82
Регулярные выражения	83
Полнотекстовый поиск.....	83
Сравнение строк.....	84
Определение GIN-индексов	85
Отладка поиска.....	86
Сбор статистики по словам	87
О пользе операторов исключения	87
Резюме.....	88
Вопросы.....	88
Глава 4. Передовые средства SQL	90
Введение в наборы группирования.....	90
Загрузка тестовых данных.....	90
Применение наборов группирования	91
Сочетание наборов группирования с фразой FILTER.....	94
Использование упорядоченных наборов.....	95
Гипотетические агрегаты.....	97
Оконные функции и аналитические средства	97
Разбиение данных.....	98
Упорядочение данных внутри окна.....	99
Скользящие окна.....	100
Абстрагирование окон.....	102

Использование встроенных оконных функций.....	103
Создание собственных агрегатов	109
Создание простых агрегатов	109
Добавление поддержки параллельных запросов.....	112
Повышение эффективности.....	113
Написание гипотетических агрегатов.....	114
Резюме.....	116

Глава 5. Журналы и статистика системы..... 117

Сбор статистических данных о работе системы	117
Системные представления в PostgreSQL.....	117
Создание файлов журналов	134
Конфигурационный файл postgresql.conf	134
Резюме.....	139
Вопросы.....	139

Глава 6. Оптимизация запросов для достижения максимальной производительности..... 141

Что делает оптимизатор.....	141
Оптимизация на примере	142
Разбираемся в планах выполнения.....	151
Систематический подход к планам выполнения	151
Выявление проблем	153
Соединения: осмысление и исправление.....	157
Как соединять правильно	157
Обработка внешних соединений	158
Параметр join_collapse_limit	159
Включение и выключение режимов оптимизатора.....	160
Генетическая оптимизация запросов.....	163
Секционирование данных	164
Создание секций	164
Применение табличных ограничений.....	166
Модификация наследуемой структуры	167
Перемещение таблицы в наследуемую структуру и из нее.....	168
Очистка данных	168
Секционирование в PostgreSQL 11.0.....	169
Настройка параметров для повышения производительности запросов	171
Ускорение сортировки	173
Ускорение административных задач	175
Распараллеливание запросов	176
Что PostgreSQL умеет делать параллельно?	179
Распараллеливание на практике.....	179
Введение в JIT-компиляцию	180
Настройка JIT.....	181
Выполнение запросов.....	181
Резюме.....	183

Глава 7. Написание хранимых процедур	184
Языки хранимых процедур.....	184
Фундаментальные основы – хранимые процедуры и функции.....	185
Анатомия функции.....	186
Языки хранимых процедур.....	189
Введение в PL/pgSQL.....	190
Создание хранимых процедур на PL/pgSQL.....	204
Введение в PL/Perl.....	205
Введение в PL/Python.....	211
Улучшение функций.....	214
Уменьшение числа вызовов функций.....	214
Резюме.....	218
Вопросы.....	218
Глава 8. Безопасность в PostgreSQL	220
Управление сетевой безопасностью.....	220
Подключения и адреса привязки.....	221
Файл pg_hba.conf.....	224
Безопасность на уровне экземпляра.....	228
Задание безопасности на уровне базы данных.....	232
Задание прав на уровне схемы.....	233
Работа с таблицами.....	235
Задание прав на уровне столбцов.....	236
Задание привилегий по умолчанию.....	237
Безопасность на уровне строк.....	238
Просмотр прав.....	242
Передача объектов и удаление пользователей.....	243
Резюме.....	245
Вопросы.....	245
Глава 9. Резервное копирование и восстановление	246
Простая выгрузка.....	246
Запуск pg_dump.....	247
Задание пароля и информации о подключении.....	248
Извлечение подмножества данных.....	250
Форматы резервной копии.....	250
Восстановление из резервной копии.....	252
Сохранение глобальных данных.....	253
Резюме.....	253
Вопросы.....	254
Глава 10. Резервное копирование и репликация	255
Что такое журнал транзакций.....	255
Знакомство с журналом транзакций.....	256
Контрольные точки.....	257
Оптимизация журнала транзакций.....	257

Архивация и восстановление журнала транзакций.....	259
Настройка архивации	259
Конфигурирование файла pg_hba.conf	260
Создание базовой резервной копии	261
Воспроизведение журнала транзакций.....	263
Очистка архива журналов транзакций.....	267
Настройка асинхронной репликации	268
Базовая настройка.....	268
Остановка и возобновление репликации.....	270
Проверка состояния репликации для обеспечения доступности.....	271
Обработка отказов и линии времени	274
Управление конфликтами	275
Повышение надежности репликации.....	276
Переход на синхронную репликацию	277
Настройка долговечности.....	279
Слоты репликации.....	280
Работа с физическими слотами репликации	281
Работа с логическими слотами репликации	283
Использование команд CREATE PUBLICATION и CREATE SUBSCRIPTION	285
Резюме.....	287
Вопросы.....	287

Глава 11. Полезные расширения..... 289

Как работают расширения	289
Проверка доступных расширений	290
Использование модулей из подборки contrib	293
Модуль adminpack	293
Применение фильтра Блума	294
Установка btree_gist и btree_gin.....	296
Dblink – пора расстаться	297
Доступ к файлам с помощью file_fdw.....	298
Анализ хранилища с помощью pageinspect	299
Анализ кеша с помощью pg_buffercache.....	301
Шифрование данных с помощью pgcrypto.....	302
Прогрев кеша с помощью pg_prewarm.....	302
Анализ производительности с помощью pg_stat_statements	304
Анализ хранилища с помощью pgstattuple.....	304
Нечеткий поиск с помощью pg_trgm	305
Подключение к удаленному серверу с помощью postgres_fdw.....	305
Другие полезные расширения	309
Резюме.....	310

Глава 12. Поиск и устранение неполадок..... 311

Первоначальное изучение незнакомой базы данных	311
Анализ результатов pg_stat_activity.....	311
Опрос pg_stat_activity.....	312
Выявление медленных запросов.....	314

Анализ отдельных запросов	315
Углубленный анализ с помощью perf	316
Анализ журнала	317
Анализ наличия индексов.....	318
Анализ памяти и ввода-вывода.....	318
О конкретных ошибочных ситуациях.....	320
Повреждение clog.....	320
Что означают сообщения о контрольной точке.....	321
Что делать с поврежденными страницами данных.....	322
Беззаботное управление подключениями	323
Борьба с разбуханием таблиц.....	323
Резюме.....	324
Вопросы.....	324
Глава 13. Переход на PostgreSQL.....	325
Перенос команд SQL в PostgreSQL.....	325
Латеральные соединения	325
Наборы группирования	326
Фраза WITH – общие табличные выражения	327
Фраза WITH RECURSIVE.....	328
Фраза FILTER.....	328
Оконные функции.....	329
Упорядоченные наборы – фраза WITHIN GROUP	329
Фраза TABLESAMPLE.....	330
Ограничение выборки и смещение	331
Фраза OFFSET	331
Темпоральные таблицы.....	332
Сопоставление с образцом во временных рядах.....	332
Переход с Oracle на PostgreSQL.....	332
Использование расширения oracle_fdw для переноса данных	333
Использование ora2pg для перехода с Oracle.....	334
Распространенные подводные камни.....	336
ora_migrator – быстрая миграция Oracle в PostgreSQL	337
Переход из MySQL или MariaDB на PostgreSQL	338
Обработка данных в MySQL и MariaDB.....	339
Перенос данных и схемы	343
Резюме.....	345
Предметный указатель.....	346

Об авторе

Ганс-Юрген Шениг 18 лет работает с PostgreSQL. Он является генеральным директором компании Cybertec Schönig and Schönig GmbH, оказывающей поддержку и консультационные услуги пользователям PostgreSQL. Компания успешно обслуживает бесчисленных заказчиков по всему миру. Перед тем как основать Cybertec Schönig and Schönig GmbH в 2000 году, он трудился разработчиком баз данных в частной исследовательской компании, изучавшей австрийский рынок труда, где занимался в основном добычей данных и прогнозными моделями. Написал несколько книг о PostgreSQL.

О рецензенте

Шелдон Штраух – ветеран с 23-летним опытом консультирования таких компаний, как IBM, Sears, Ernst & Young, Kraft Foods. Имеет степень бакалавра по организации управления, применяет свои знания, чтобы помочь компаниям самоопределиться. В сферу его интересов входят сбор, управление и глубокий анализ данных, карты и их построение, бизнес-аналитика и применение анализа данных в целях непрерывного улучшения. В настоящее время занимается разработкой сквозного управления данными и добычей данных в компании Enova International, оказывающей финансовые услуги и расположенной в Чикаго. В свободное время увлекается искусством, особенно музыкой, и путешествует со своей женой Мэрилин.

Предисловие

Второе издание этой книги призвано помочь вам в создании динамических решений на основе базы данных для корпоративных приложений, где в качестве базы используется последняя версия PostgreSQL, позволяющая аналитикам без труда проектировать физические и технические аспекты системной архитектуры.

Книга начинается введением в последние возможности PostgreSQL 11, которые дают возможность строить эффективные и отказоустойчивые приложения. Мы подробно рассмотрим передовые аспекты PostgreSQL, включая логическую репликацию, кластеры баз данных, оптимизацию производительности, мониторинг и управление пользователями. Мы также покажем, как пользоваться оптимизатором PostgreSQL, настраивать базу, так чтобы она работала максимально быстро, и как перейти с Oracle на PostgreSQL. В процессе чтения книги мы познакомимся с транзакциями, блокировкой, индексами и оптимизацией запросов.

Кроме того, вы научитесь настраивать сетевую безопасность и использовать резервные копии и репликацию. Мы также расскажем о полезных расширениях PostgreSQL, позволяющих увеличить производительность при работе с большими базами данных.

Прочитав эту книгу до конца, вы сможете выжать из своей базы все до капли благодаря высокотехнологичной реализации административных задач.

НА КОГО РАССЧИТАНА ЭТА КНИГА

В этой книге рассмотрены средства, включенные в PostgreSQL 11, и показано, как создавать более качественные приложения PostgreSQL и эффективно администрировать базу данных PostgreSQL. Вы освоите передовые возможности PostgreSQL и приобретете навыки, необходимые для создания эффективных решений.

КРАТКОЕ СОДЕРЖАНИЕ КНИГИ

Глава 1 «Обзор PostgreSQL» содержит введение в PostgreSQL и сведения о новых средствах, появившихся в PostgreSQL 11.

В главе 2 «Транзакции и блокировка» показано, как наиболее эффективно использовать транзакции в PostgreSQL.

В главе 3 «Использование индексов» обсуждаются индексы – их типы, сценарии использования и реализация собственной стратегии индексирования.

Глава 4 «Передовые средства SQL» посвящена современному SQL и его возможностям. Мы рассмотрим различные типы множеств и вопрос о написании собственных агрегатов.

В главе 5 «Журналы и статистика системы» объясняется, как извлечь полезный смысл из статистики базы данных.

В главе 6 «Оптимизация запросов для достижения максимальной производительности» показано, как писать более быстрые запросы. Мы также поговорим о том, что делает запрос плохим.

В главе 7 «Написание хранимых процедур» мы рассмотрим различия между процедурами и функциями. Обсуждаются также хранимые процедуры, использование расширений и некоторые продвинутые возможности PL/pgSQL.

В главе 8 «Безопасность в PostgreSQL» описаны типичные проблемы безопасности, с которыми сталкивается разработчик или администратор базы данных PostgreSQL.

Глава 9 «Резервное копирование и восстановление» посвящена восстановлению базы данных из резервной копии, а также вопросу о частичной выгрузке данных.

В главе 10 «Резервное копирование и репликация» мы рассмотрим журнал транзакций в PostgreSQL и объясним, как работать с ним, чтобы повысить качество и безопасность СУБД.

В главе 11 «Полезные расширения» обсуждаются некоторые широко распространенные расширения PostgreSQL.

Темой главы 12 «Поиск и устранение неполадок» являются анализ неизвестной базы данных, выявление узких мест, решение проблем, возникающих при повреждении системы хранения, и инспекция неработающих реплик.

Глава 13 «Переход на PostgreSQL» посвящена переходу с других СУБД на PostgreSQL.

КАК ВЫЖАТЬ МАКСИМУМ ИЗ ЭТОЙ КНИГИ

Книга написана для широкой аудитории. Для проработки приведенных примеров хорошо бы иметь некоторый опыт работы с SQL вообще и с PostgreSQL в частности (хотя это требование необязательно). Не мешает также знакомство с командной строкой UNIX.

ГРАФИЧЕСКИЕ ВЫДЕЛЕНИЯ



В этой книге применяется ряд соглашений о графическом выделении.

Код в тексте: зарезервированные слова, имена таблиц базы данных, имена папок и файлов, URL-адреса, данные, которые вводит пользователь, и адреса в Твиттере, например: «Я добавлю в таблицу одну строку с помощью простой команды INSERT».

Команды и их результаты набраны таким шрифтом:

```
test=# CREATE TABLE t_test (id int);
CREATE TABLE
test=# INSERT INTO t_test VALUES (0);
INSERT 0 1
```

Новые термины, важные фрагменты, а также элементы графического интерфейса в меню или диалоговых окнах набраны **полужирным шрифтом**, например «Выберите пункт **System info** на панели **Administration**».

-  Так будут оформляться предупреждения и важные примечания.
-  Так будут оформляться советы или рекомендации.

ОТЗЫВЫ И ПОЖЕЛАНИЯ

Мы всегда рады отзывам наших читателей. Расскажите нам, что вы думаете об этой книге – что понравилось или, может быть, не понравилось. Отзывы важны для нас, чтобы выпускать книги, которые будут для вас максимально полезны.

Вы можете написать отзыв прямо на нашем сайте www.dmkpress.com, зайдя на страницу книги и оставив комментарий в разделе «Отзывы и рецензии». Также можно послать письмо главному редактору по адресу dmkpress@gmail.com, при этом напишите название книги в теме письма.

Если есть тема, в которой вы квалифицированы, и вы заинтересованы в написании новой книги, заполните форму на нашем сайте по адресу http://dmkpress.com/authors/publish_book/ или напишите в издательство по адресу dmkpress@gmail.com.

СПИСОК ОПЕЧАТОК

Хотя мы приняли все возможные меры, для того чтобы удостовериться в качестве наших текстов, ошибки все равно случаются. Если вы найдете ошибку в одной из наших книг – возможно, ошибку в тексте или в коде, – мы будем очень благодарны, если вы сообщите нам о ней. Сделав это, вы избавите других читателей от расстройств и поможете нам улучшить последующие версии данной книги.

Если вы найдете какие-либо ошибки в коде, пожалуйста, сообщите о них главному редактору по адресу dmkpress@gmail.com, и мы исправим это в следующих тиражах.

НАРУШЕНИЕ АВТОРСКИХ ПРАВ

Пиратство в интернете по-прежнему остается насущной проблемой. Издательства «ДМК Пресс» и Packt очень серьезно относятся к вопросам защиты авторских прав и лицензирования. Если вы столкнетесь в интернете с незаконно выполненной копией любой нашей книги, пожалуйста, сообщите нам адрес копии или веб-сайта, чтобы мы могли принять меры.

Пожалуйста, свяжитесь с нами по адресу электронной почты dmkpress@gmail.com со ссылкой на подозрительные материалы.

Мы высоко ценим любую помощь по защите наших авторов, помогающую нам предоставлять вам качественные материалы.

Глава 1

Обзор PostgreSQL

Прошло уже немало времени с тех пор, как я написал последнюю книгу о PostgreSQL. Я прошел долгий путь и рад рассказать обо всем, что узнал, в этом издании «Осваиваем PostgreSQL», где освещены крутые новшества, вошедшие в PostgreSQL 11.

PostgreSQL – одна из самых передовых систем баз данных с открытым исходным кодом, в ней много возможностей, широко используемых как разработчиками, так и системными администраторами. А в версию PostgreSQL 11 добавлен ряд вещей, способных сделать этот исключительный продукт еще успешнее.

В данной книге мы подробно рассмотрим и обсудим многие из них.

Эта глава представляет собой введение в PostgreSQL и появившиеся в версии PostgreSQL 11 новшества. Детально описывается часть новой функциональности. Учитывая огромное количество изменений и сам размер проекта PostgreSQL, представленный список, конечно, далеко не полон, я старался выделить наиболее важные моменты, интересные большинству пользователей.

Рассматриваемые в этой главе средства можно разбить на несколько категорий:

- что нового в PostgreSQL 11;
- средства, относящиеся к SQL и к разработке;
- резервное копирование, восстановление и репликация;
- производительность.

Что нового в PostgreSQL 11.0?

Версия PostgreSQL 11, вышедшая осенью 2018 г., предлагает пользователям современные средства, полезные равно профессионалам и начинающим. PostgreSQL 11 – вторая основная версия, нумеруемая в соответствии с новой схемой, принятой сообществом PostgreSQL. Следующая основная версия будет иметь номер 12. Сервисные версии будут называться **PostgreSQL 11.1**, **11.2**, **11.3** и т. д. Это существенное изменение, по сравнению со схемой нумерации, действовавшей до выхода версии 10, поэтому на него следует обратить внимание.

Какую версию использовать? Рекомендуется работать с самой последней. Нет никакого смысла начинать работу, скажем, с версии PostgreSQL 9.6. Если вы только приступаете к использованию PostgreSQL, берите версию 11. В PostgreSQL не бывает дефектов – сообщество всегда дает вам работающий код, поэтому бояться PostgreSQL 10 или PostgreSQL 11 не надо. Они работают – и все тут.

Новые средства администрирования базы данных

В PostgreSQL 11 много новых средств, помогающих снизить нагрузку на администратора и сделать систему более надежной и стабильной.

Одно из таких средств – возможность задавать размер **сегмента WAL**.

Задание размера сегментов WAL

Первая версия PostgreSQL вышла 20 лет назад, и с тех пор размер одного файла предзаписи (файла WAL) был равен 16 МБ. В самом начале это ограничение даже было зашито в код, но впоследствии стало параметром времени компиляции. Начиная с версии PostgreSQL 11 размер сегментов WAL можно задавать в момент создания экземпляра, что дает администратору дополнительный рычаг конфигурирования и оптимизации PostgreSQL. В следующем примере показано, как задать размер сегмента WAL в момент запуска `initdb`:

```
initdb -D /pgdata --wal-segsize=32
```

Команда `initdb` создает экземпляр базы данных. Обычно этот вызов скрыт в каком-то скрипте операционной системы: вашего любимого дистрибутива Linux, Windows или иной. Теперь у `initdb` появился параметр, позволяющий задать требуемый размер сегмента WAL.

Как уже было сказано, по умолчанию размер равен 16 МБ, но в большинстве случаев для повышения производительности имеет смысл его увеличить. Использовать меньший размер стоит, только если вы работаете с совсем крохотной базой данных во встраиваемой системе.

Какого влияния на производительность следует ожидать? Как всегда, все зависит от того, что именно вы делаете. Если 99% операций, выполняемых базой данных, – чтение, то эффект увеличения размера сегментов WAL будет нулевым. Да-да, вы не ослышались – **НУЛЕВЫМ**. Если операции записи выполняются, когда система простаивает 95% времени и не подвергается серьезной нагрузке, то эффект будет нулевым или близким к тому. Выигрыш имеет место, только если в составе рабочей нагрузки преобладают операции записи. Лишь в этом случае размер стоит изменять. А если речь идет о парочке онлайн-форм, заполняемых случайным посетителем сайта, то к чему огород городить? Эта новая возможность заиграет во всю силу, только когда в базе производится много изменений и, стало быть, в WAL пишется много данных.

Увеличенный `queryid` в `pg_stat_statements`

Если вы хотите серьезно заняться производительностью PostgreSQL, то стоит приглядеться к представлению `pg_stat_statements`. Лично я считаю его бесцен-

ным подспорьем для всякого, кто хочет знать, что происходит в системе. Модуль `pg_stat_statements` загружается в момент запуска PostgreSQL, если прописан в параметре `shared_preload_libraries`, и собирает статистические сведения о запросах, выполняемых сервером. Представление сразу покажет, если что-то пошло не так.

В представлении `pg_stat_statements` имеется поле `queryid`, которое до сих пор содержало 32-разрядный идентификатор (внутренний хеш-код). Иногда это приводило к проблемам из-за коллизии ключей. Магнус Хагандер (Magnus Hagander) в одной из своих статей подсчитал, что после выполнения 3 млрд запросов следует ожидать примерно 50 000 коллизий. После перехода на 64-разрядный `queryid` это количество уменьшается до примерно 0.25 конфликта на 3 млрд запросов, что следует считать значительным улучшением.

Имейте в виду, что если вы используете в своих скриптах представление `pg_stat_statements` для диагностики проблем с производительностью, то после перехода на PostgreSQL 11 эти скрипты нужно будет обновить.

Усовершенствования в индексировании и оптимизации

PostgreSQL 11 предлагает целый спектр улучшенных функций для администрирования. Не остались в стороне и индексы. К ним как раз относятся одни из самых важных усовершенствований.

Статистика индекса по выражению

Для оптимизации простого запроса PostgreSQL анализирует внутреннюю статистику. Рассмотрим пример:

```
SELECT * FROM person WHERE gender = 'female';
```

В этом случае PostgreSQL смотрит на внутреннюю статистику и оценивает количество женщин в таблице `person`. Если оно мало, то PostgreSQL будет использовать индекс, а если большая часть записей относится к женщинам, то PostgreSQL предпочтет последовательный просмотр. Статистика собирается по каждому столбцу. Кроме того, в PostgreSQL 10 добавлена возможность собирать межстолбцовую статистику (посмотрите справку по команде `CREATE STATISTICS`). Хорошая новость состоит в том, что PostgreSQL может также собирать статистику для функциональных индексов:

```
CREATE INDEX idx_cos ON t_data (cos(data));
```

Но до сих пор было невозможно использовать более сложную статистику по функциональным индексам.

Рассмотрим пример индекса по нескольким столбцам:

```
CREATE INDEX coord_idx ON measured (x, y, (z + t));
ALTER INDEX coord_idx ALTER COLUMN 3 SET STATISTICS 1000;
```

Здесь мы имеем индекс по двум физическим столбцам и еще одному виртуальному столбцу, представленному выражением. Новая возможность позволяет явно собрать больше статистики для третьего столбца, который в про-

тивном случае был бы покрыт неоптимально. В этом примере мы говорим PostgreSQL, что по третьему столбцу в системной статистике должно быть 1000 записей. Это позволит оптимизатору точнее вычислить оценку и, значит, создать более качественный план – весьма полезное прибавление эффективности в некоторых специализированных приложениях.

Покрывающие индексы

Во многих других системах баз данных давно уже существуют **покрывающие индексы**¹. Что это значит? Рассмотрим следующий пример, где мы просто выбираем из таблицы два столбца:

```
SELECT id, name FROM person WHERE id = 10;
```

Предположим, что имеется индекс по столбцу `id`. В таком случае PostgreSQL найдет ключ в этом индексе, а остальные поля прочтает из таблицы. Это называется просмотром индекса (`index scan`) и включает обращение и к индексу, и к базовой таблице для формирования строки. Раньше для решения проблемы нужно было бы создать индекс по двум столбцам, что позволило бы PostgreSQL выполнить просмотр только индекса (`index-only scan`), а не просмотр индекса. Если индекс содержит все необходимые столбцы, то дополнительных обращений к таблице не нужно (как правило).



Выбирайте только столбцы, которые действительно необходимы, иначе все может закончиться бессмысленным обращением к таблице. Запросы типа показанного ниже в общем случае считаются плохими с точки зрения производительности: `SELECT * FROM person WHERE id = 10;`

Проблема возникает, если `id` должен быть первичным ключом, но при этом хочется, чтобы при чтении дополнительного столбца выполнялся просмотр только индекса. Здесь-то и приходит на помощь новая возможность:

```
CREATE UNIQUE INDEX some_name ON person USING btree (id) INCLUDE (name);
```

PostgreSQL гарантирует, что столбец `id` будет уникальным, но при этом в индексе будет храниться дополнительное поле, так что при запросе обоих столбцов будет произведен просмотр только индекса. Если рабочая нагрузка преимущественно типа OLTP, то производительность может резко возрасти. Разумеется, точные оценки привести трудно, потому что таблицы и запросы у всех разные.

Параллельное построение индексов

Традиционно для построения индекса в PostgreSQL сервер использовал только одно процессорное ядро. Но PostgreSQL используется во все более крупных системах, поэтому создание индексов становилось камнем преткновения. И тогда сообщество начало думать о том, как улучшить сортировку. Первый шаг – раз-

¹ Автор использует термин «покрывающие индексы» как синоним INCLUDE-индексов. В другом распространенном значении «покрывающим» называется любой индекс, содержащий все необходимые для запроса значения. – *Прим. ред.*

решить параллельное построение **В-деревьев** – уже реализован в PostgreSQL 11. В будущих версиях PostgreSQL параллельная сортировка будет разрешена и для обычных операций (в версии 11 это, к сожалению, еще не поддерживается).

Параллельное построение может существенно ускорить создание индексов, и мы с нетерпением ждем дальнейших прорывов в этой области (например, поддержки для индексов других типов).

Улучшенное управление кешем

PostgreSQL 11 предлагает дополнительные способы управления кешем ввода-вывода (разделяемыми буферами). Особенно стоит отметить команду `pg_prewarm`.

Усовершенствованная команда `pg_prewarm`

Команда `pg_prewarm` позволяет восстановить содержимое кеша ввода-вывода PostgreSQL после перезапуска. Она существует уже довольно давно и широко применяется пользователями PostgreSQL. В версии 11 `pg_prewarm` дополнена и теперь допускает автоматическую выгрузку списка буферов с регулярными интервалами.

Можно также автоматически загрузить старое содержимое кеша, чтобы наблюдаемая производительность не ухудшалась после перезапуска. От этих улучшений выиграют системы с большим объемом оперативной памяти.

Улучшенные оконные функции

Оконные функции и средства аналитики – краеугольный камень любой современной реализации SQL, поэтому они широко используются профессионалами. PostgreSQL уже довольно давно поддерживает оконные функции, но некоторые возможности, упомянутые в стандарте SQL, до сих пор отсутствовали. Теперь, в версии PostgreSQL 11, поддерживается все, что требует стандарт SQL: 2011 в этой области.

Добавлены следующие возможности:

- RANGE BETWEEN:
 - раньше только ROWS;
 - теперь поддерживаются значения;
- исключение фрейма:
 - EXCLUDE CURRENT ROW;
 - EXCLUDE TIES.

Для демонстрации новых возможностей я включил пример. Приведенный ниже код содержит две оконные функции. В первой используется то, что уже было в PostgreSQL 10 и раньше. Во второй функции `array_agg` исключается текущая строка, это новая возможность, появившаяся в PostgreSQL 11.

```
test=# SELECT *,
      array_agg(x) OVER (ORDER BY x ROWS BETWEEN
                        1 PRECEDING AND 1 FOLLOWING),
      array_agg(x) OVER (ORDER BY x ROWS BETWEEN
```



```

1 PRECEDING AND 1 FOLLOWING EXCLUDE CURRENT ROW)
FROM generate_series(1, 5) AS x;
x | array_agg | array_agg
-----+-----+-----
1 | {1,2}     | {2}
2 | {1,2,3}   | {1,3}
3 | {2,3,4}   | {2,4}
4 | {3,4,5}   | {3,5}
5 | {4,5}     | {4}
(5 строк)

```

Исключение текущей строки – довольно обычное требование, так что это улучшение не следует недооценивать.

Добавление JIT-компиляции

Своевременная (just-in-time, JIT) компиляция – одна из жемчужин PostgreSQL 11. Добавлена обширная инфраструктура для поддержки дополнительных видов JIT-компиляции в будущем, а PostgreSQL 11 – первая версия, в которой эта современная техника используется на полную катушку. Но сначала разберемся, что, собственно, такое JIT-компиляция. При выполнении запроса многое становится известно только на этапе выполнения, а не на этапе компиляции запроса. Поэтому традиционный компилятор находится в невыгодном положении, т. к. не знает, что произойдет во время выполнения. А JIT-компилятор знает гораздо больше и может отреагировать соответственно.

JIT-компиляция реализована начиная с версии PostgreSQL 11 и особенно полезна для больших запросов. Детали будут рассмотрены в последующих главах.

Улучшенное секционирование

Первый вариант секционирования появился в PostgreSQL 10. Конечно, наследованием мы пользовались и раньше. Но PostgreSQL 10 стала первой версией, в которой это делается на современном уровне. В PostgreSQL 11 эта и так уже весьма мощная функциональность расширена, в частности появилась возможность создавать секцию по умолчанию, если ни одна из существующих не подходит.

Вот как это работает:

```
postgres=# CREATE TABLE default_part PARTITION OF some_table DEFAULT;
CREATE TABLE
```

В данном случае все строки, не попавшие ни в какую другую секцию, попадают в секцию default_part.

Но это еще не все. В PostgreSQL строку нельзя было (легко) переместить из одной секции в другую. Предположим, что мы завели по одной секции на страну. Если человек переехал, например, из Франции в Эстонию, то одной команды UPDATE раньше было недостаточно. Нужно было удалить старую строку и вставить новую. В PostgreSQL 11 эта проблема решена. Строки перемещаются из одной секции в другую совершенно прозрачно.

Раньше в PostgreSQL было много других недостатков. В прежних версиях секции приходилось индексировать по отдельности. Невозможно было создать один индекс для всех секций. В PostgreSQL 11 построение индекса над родительской таблицей автоматически гарантирует, что все дочерние таблицы тоже будут проиндексированы. И это замечательно, потому что теперь вряд ли удастся просто забыть про какой-то индекс. Кроме того, в PostgreSQL 11 можно добавить глобальный уникальный индекс¹, так что для секционированной таблицы стало возможно определить ограничение уникальности.

Вплоть до PostgreSQL 10 у нас было секционирование по диапазону и секционирование по списку. В PostgreSQL 11 добавилось секционирование по хеш-коду. Например:

```
test=# CREATE TABLE tab(i int, t text) PARTITION BY HASH (i);
CREATE TABLE
test=# CREATE TABLE tab_1 PARTITION OF tab FOR VALUES WITH (MODULUS 4, REMAINDER 0);
CREATE TABLE
```

Но добавлена не только новая функциональность, но и целый ряд улучшений производительности. Устранение секций теперь производится гораздо быстрее, а кроме того, появилась возможность соединений по секциям и агрегатов по секциям, а это именно то, что необходимо для аналитики и хранилищ данных.

Поддержка хранимых процедур

В PostgreSQL всегда были функции, которые часто назывались хранимыми процедурами. Однако между хранимой процедурой и функцией есть различие. Вплоть до PostgreSQL 10 у нас были только функции, а процедур не было.

Вся штука в том, что функция – это часть объемлющей структуры, транзакции. А процедура может содержать несколько транзакций, поэтому ее нельзя вызывать из объемлющей транзакции, т. е. она в каком-то смысле самостоятельна.

Приведем синтаксис команды CREATE PROCEDURE:

```
test=# \h CREATE PROCEDURE
Команда: CREATE PROCEDURE
Описание: создать процедуру
Синтаксис:
CREATE [ OR REPLACE ] PROCEDURE
    Имя ( [ [ режим_аргумента ] [ имя_аргумента ]
        тип_аргумента [ { DEFAULT | = } выражение_по_умолчанию ] [, ... ] )
{ LANGUAGE имя_языка
  | TRANSFORM { FOR TYPE имя_типа } [, ... ]
  | [ EXTERNAL ] SECURITY INVOKER | [ EXTERNAL ] SECURITY DEFINER
  | SET параметр_конфигурации { TO значение | = значение | FROM CURRENT }
  | AS 'определение'
  | AS 'объектный_файл', 'объектный_символ'
} ...
```

¹ Технически такой индекс не является глобальным: в каждой секции создаются собственные локальные индексы. – *Прим. ред.*

Ниже показано, как в одной процедуре можно выполнить две транзакции:

```
test=# CREATE PROCEDURE test_proc()
      LANGUAGE plpgsql
AS $$
BEGIN
  CREATE TABLE a (aid int);
  CREATE TABLE b (bid int);
  COMMIT;
  CREATE TABLE c (cid int);
  ROLLBACK;
END;
$$;
CREATE PROCEDURE
```

Отметим, что первые две команды зафиксированы, а вторая транзакция отменена. Чуть ниже мы увидим, к чему это приводит.

Для выполнения данной процедуры используется команда CALL:

```
test=# CALL test_proc();
CALL
```

Первые две таблицы созданы, а третья – нет, потому что внутри процедуры была команда ROLLBACK:

```
test=# \d
      Список отношений
  Схема | Имя | Тип | Владелец
-----+-----+-----+-----
 public | a   | table | hs
 public | b   | table | hs
(2 строки)
```

Процедуры – важный шаг на пути к созданию полнофункциональной системы баз данных.

Улучшение команды ALTER TABLE

Команда ALTER TABLE используется, чтобы изменить определение таблицы. В PostgreSQL 11 поведение команды ALTER TABLE ... ADD COLUMN улучшено. В примерах ниже показано, как можно добавить столбцы в таблицу и как PostgreSQL будет работать с этими столбцами:

```
ALTER TABLE x ADD COLUMN y int;
ALTER TABLE x ADD COLUMN z int DEFAULT 57;
```

Первая из этих двух команд всегда работала быстро, поскольку в PostgreSQL столбец по умолчанию имеет значение NULL. Поэтому PostgreSQL просто добавляет описание столбца в системный каталог, а данные таблицы при этом не изменяются. Столбец добавляется в конец таблицы, поэтому если хранящаяся на диске строка оказывается слишком короткой, мы знаем, что последнее поле в ней содержит NULL.

Конец ознакомительного фрагмента.

Приобрести книгу можно

в интернет-магазине

«Электронный универс»

e-Univers.ru