Оглавление

О книге 17
Глава 1. Введение 23

Часть I. Изоляция и многоверсионность **45**
Глава 2. Изоляция 47
Глава 3. Страницы и версии строк 75
Глава 4. Снимки данных 97
Глава 5. Внутристраничная очистка и hot-обновления 111
Глава 6. Очистка и автоочистка 124
Глава 7. Заморозка 149
Глава 8. Перестроение таблиц и индексов 163

Часть II. Буферный кеш и журнал **175**
Глава 9. Буферный кеш 177
Глава 10. Журнал предзаписи 198
Глава 11. Режимы журнала 220

Часть III. Блокировки **239**
Глава 12. Блокировки отношений 241
Глава 13. Блокировки строк 254
Глава 14. Блокировки разных объектов 279
Глава 15. Блокировки в памяти 291

Часть IV. Выполнение запросов **301**
Глава 16. Этапы выполнения запросов 303
Глава 17. Статистика 327
Глава 18. Табличные методы доступа 352
Глава 19. Индексные методы доступа 375
Глава 20. Индексное сканирование 395
Глава 21. Вложенный цикл 420

Глава 22. Хеширование 441
Глава 23. Сортировка и слияние 466

Часть V. Типы индексов **493**
Глава 24. Хеш-индекс 495
Глава 25. B-дерево 507
Глава 26. Индекс GiST 534
Глава 27. Индекс SP-GiST 568
Глава 28. Индекс GIN 592
Глава 29. Индекс BRIN 622

Заключение 650
Предметный указатель 651

Содержание

О книге 17

Глава 1. Введение 23
1.1. Организация данных . . . 23
Базы данных . . . 23
Системный каталог . . . 24
Схемы . . . 25
Табличные пространства . . . 26
Отношения . . . 27
Слои и файлы . . . 28
Страницы . . . 33
TOAST . . . 33
1.2. Процессы и память . . . 39
1.3. Клиенты и клиент-серверный протокол . . . 41

Часть I. Изоляция и многоверсионность 45

Глава 2. Изоляция 47
2.1. Согласованность . . . 47
2.2. Уровни изоляции и аномалии в стандарте SQL . . . 49
Потерянное обновление . . . 50
Грязное чтение и Read Uncommitted . . . 50
Неповторяющееся чтение и Read Committed . . . 51
Фантомное чтение и Repeatable Read . . . 51
Отсутствие аномалий и Serializable . . . 52
Почему именно эти аномалии? . . . 52
2.3. Уровни изоляции в PostgreSQL . . . 54
Read Committed . . . 55
Repeatable Read . . . 63
Serializable . . . 70
2.4. Какой уровень изоляции использовать? . . . 73

Глава 3. Страницы и версии строк **75**
3.1. Структура страниц . . . 75
Заголовок страницы . . . 75
Специальная область . . . 76
Версии строк . . . 76
Указатели на версии строк . . . 77
Свободное место . . . 78
3.2. Структура версий строк . . . 78
3.3. Выполнение операций над версиями строк . . . 80
Вставка . . . 81
Фиксация . . . 85
Удаление . . . 87
Отмена . . . 88
Обновление . . . 88
3.4. Индексы . . . 89
3.5. TOAST . . . 90
3.6. Виртуальные транзакции . . . 91
3.7. Вложенные транзакции . . . 92
Точки сохранения . . . 92
Ошибки и атомарность операций . . . 94

Глава 4. Снимки данных **97**
4.1. Что такое снимок данных . . . 97
4.2. Видимость версий строк в снимке . . . 98
4.3. Из чего состоит снимок . . . 99
4.4. Видимость собственных изменений . . . 104
4.5. Горизонт транзакции . . . 105
4.6. Снимок данных для системного каталога . . . 108
4.7. Экспорт снимка данных . . . 109

Глава 5. Внутристраничная очистка и hot-обновления **111**
5.1. Внутристраничная очистка . . . 111
5.2. Hot-обновления . . . 115
5.3. Внутристраничная очистка при hot-обновлениях . . . 119
5.4. Разрыв hot-цепочки . . . 120
5.5. Внутристраничная очистка индексов . . . 122

Глава 6. Очистка и автоочистка **124**
6.1. Очистка вручную 124
6.2. Еще раз о горизонте базы данных 127
6.3. Этапы выполнения очистки 130
Сканирование таблицы 130
Очистка индексов 130
Очистка таблицы 132
Усечение таблицы 132
6.4. Анализ 133
6.5. Автоматическая очистка и анализ 133
Устройство автоочистки 134
Какие таблицы требуют очистки 135
Какие таблицы требуют анализа 137
Автоочистка в действии 138
6.6. Регулирование нагрузки 142
Управление интенсивностью обычной очистки 143
Управление интенсивностью автоочистки 143
6.7. Мониторинг очистки 144
Отслеживание выполнения ручной очистки 145
Отслеживание выполнения автоочистки 147

Глава 7. Заморозка **149**
7.1. Переполнение счетчика транзакций 149
7.2. Заморозка версий и правила видимости 150
7.3. Управление заморозкой 153
Минимальный возраст для заморозки 154
Возраст для агрессивной заморозки 156
Возраст для аварийного срабатывания автоочистки 158
Возраст для приоритетного режима заморозки 160
7.4. Заморозка вручную 160
Очистка с заморозкой 161
Заморозка при загрузке 161

Глава 8. Перестроение таблиц и индексов **163**
8.1. Полная очистка 163
Необходимость 163
Оценка плотности информации 164

Заморозка 168
8.2. Другие способы перестроения 169
Аналоги полной очистки 169
Перестроение без долгих блокировок 170
8.3. Профилактика 171
Читающие запросы 171
Обновление данных 172

Часть II. Буферный кеш и журнал 175

Глава 9. Буферный кеш 177
9.1. Кеширование 177
9.2. Устройство буферного кеша 178
9.3. Попадание в кеш 180
9.4. Промах кеша 185
Поиск буфера и вытеснение 186
9.5. Массовое вытеснение 188
9.6. Настройка размера 191
9.7. Прогрев кеша 194
9.8. Локальный кеш 196

Глава 10. Журнал предзаписи 198
10.1. Журналирование 198
10.2. Устройство журнала 200
Логическая структура 200
Физическая структура 203
10.3. Контрольная точка 205
10.4. Восстановление 210
10.5. Фоновая запись 213
10.6. Настройка 214
Настройка контрольной точки 214
Настройка фоновой записи 217
Мониторинг 217

Глава 11. Режимы журнала 220
11.1. Производительность 220

11.2. Надежность 224
Кеширование 225
Повреждение данных 226
Неатомарность записи 228
11.3. Уровни журнала 232
Minimal 233
Replica 235
Logical 237

Часть III. Блокировки **239**

Глава 12. Блокировки отношений **241**
12.1. Общие сведения о блокировках 241
12.2. Тяжелые блокировки 244
12.3. Блокировки номеров транзакций 246
12.4. Блокировки отношений 247
12.5. Очередь ожидания 250

Глава 13. Блокировки строк **254**
13.1. Устройство 254
13.2. Режимы блокировки строки 255
Исключительные режимы 255
Разделяемые режимы 257
13.3. Мультитранзакции 258
13.4. Очередь ожидания 260
Исключительные режимы 260
Разделяемые режимы 267
13.5. Блокировка без ожидания 270
13.6. Взаимоблокировки 272
Взаимоблокировка при обновлении строк 274
Взаимоблокировка двух команд UPDATE 275

Глава 14. Блокировки разных объектов **279**
14.1. Блокировки не-отношений 279
14.2. Блокировки расширения отношения 281
14.3. Блокировки страниц 282

14.4. Рекомендательные блокировки 282
14.5. Предикатные блокировки 284

Глава 15. Блокировки в памяти **291**
15.1. Спин-блокировки 291
15.2. Легкие блокировки 292
15.3. Примеры 292
Буферный кеш 292
Буферы журнала предзаписи 294
15.4. Мониторинг ожиданий 295
15.5. Семплирование 297

Часть IV. Выполнение запросов **301**

Глава 16. Этапы выполнения запросов **303**
16.1. Демонстрационная база данных 303
16.2. Протокол простых запросов 306
Разбор 306
Трансформация 308
Планирование 310
Исполнение 319
16.3. Протокол расширенных запросов 321
Подготовка 321
Привязка параметров 322
Планирование и исполнение 323
Получение результатов 326

Глава 17. Статистика **327**
17.1. Базовая статистика 327
17.2. Неопределенные значения 331
17.3. Уникальные значения 332
17.4. Наиболее частые значения 334
17.5. Гистограмма 337
17.6. Статистика для нескалярных типов данных 341
17.7. Средний размер поля 342
17.8. Корреляция 342

17.9. Статистика по выражению 343
Расширенная статистика по выражению 344
Статистика для индекса по выражению 345
17.10. Многовариантная статистика 346
Функциональные зависимости между столбцами 346
Многовариантное число различных значений 348
Многовариантные списки частых значений 350

Глава 18. Табличные методы доступа **352**
18.1. Подключаемые движки хранения 352
18.2. Последовательное сканирование 354
Оценка стоимости 355
18.3. Параллельные планы выполнения 359
18.4. Параллельное последовательное сканирование 360
Оценка стоимости 361
18.5. Ограничения параллельного выполнения 365
Количество рабочих процессов 365
Нераспараллеливаемые запросы 369
Ограниченно распараллеливаемые запросы 370

Глава 19. Индексные методы доступа **375**
19.1. Индексы и расширяемость 375
19.2. Классы и семейства операторов 378
Класс операторов 378
Семейство операторов 383
19.3. Интерфейс механизма индексирования 385
Свойства метода доступа 386
Свойства индекса 390
Свойства столбцов 391

Глава 20. Индексное сканирование **395**
20.1. Простое индексное сканирование 395
Оценка стоимости 396
Хороший случай: высокая корреляция 397
Плохой случай: низкая корреляция 400
20.2. Сканирование только индекса 403
Include-индексы 406

20.3. Сканирование по битовой карте 408
Точность карты 409
Действия с битовыми картами 411
Оценка стоимости 412
20.4. Параллельные версии индексного сканирования 416
20.5. Сравнение методов доступа 418

Глава 21. Вложенный цикл **420**
21.1. Виды и способы соединений 420
21.2. Соединение вложенным циклом 422
Декартово произведение 422
Параметризованное соединение 426
Кеширование (мемоизация) строк 430
Внешние соединения 434
Анти- и полусоединения 436
Не эквисоединения 438
Параллельный режим 439

Глава 22. Хеширование **441**
22.1. Соединение хешированием 441
Однопроходное соединение хешированием 441
Двухпроходное соединение хешированием 447
Динамические корректировки плана 450
Соединение хешированием в параллельных планах 454
Параллельное однопроходное хеш-соединение 455
Параллельное двухпроходное хеш-соединение 457
Модификации 460
22.2. Группировка и уникальные значения 463

Глава 23. Сортировка и слияние **466**
23.1. Соединение слиянием 466
Слияние отсортированных наборов 466
Параллельный режим 470
Модификации 471
23.2. Сортировка 472
Быстрая сортировка 474
Частичная пирамидальная сортировка 475

Внешняя сортировка 477
Инкрементальная сортировка 481
Параллельный режим 483
23.3. Группировка и уникальные значения 485
23.4. Сравнение способов соединения 488

Часть V. Типы индексов **493**

Глава 24. Хеш-индекс **495**
24.1. Общий принцип 495
24.2. Страничная организация 496
24.3. Класс операторов 503
24.4. Свойства 504
Свойства метода доступа 504
Свойства индекса 505
Свойства столбцов 506

Глава 25. B-дерево **507**
25.1. Общий принцип 507
25.2. Поиск и вставка 508
Поиск по равенству 508
Поиск по неравенству 510
Поиск по диапазону 511
Вставка 511
25.3. Страничная организация 513
Компактное хранение дубликатов 517
Компактное хранение внутренних индексных записей 519
25.4. Класс операторов 520
Семантика сравнения 520
Сортировка и составные индексы 526
25.5. Свойства 531
Свойства метода доступа 531
Свойства индекса 532
Свойства столбцов 532

Глава 26. Индекс GiST **534**
26.1. Общий принцип 534
26.2. R-дерево для точек 536
Страничная организация 539
Класс операторов 540
Поиск вхождения в область 542
Поиск ближайших соседей 544
Вставка 549
Ограничение исключения 550
Свойства 553
26.3. RD-дерево для полнотекстового поиска 556
Про полнотекстовый поиск 556
Индексация tsvector 557
Свойства 565
26.4. Другие типы данных 565

Глава 27. Индекс SP-GiST **568**
27.1. Общий принцип 568
27.2. Дерево квадрантов для точек 570
Класс операторов 571
Страничная организация 575
Поиск 576
Вставка 577
Свойства 580
27.3. K-мерные деревья для точек 582
27.4. Префиксное дерево для строк 584
Класс операторов 585
Поиск 586
Вставка 587
Свойства 589
27.5. Другие типы данных 590

Глава 28. Индекс GIN **592**
28.1. Общий принцип 592
28.2. Индекс для полнотекстового поиска 593
Страничная организация 595
Класс операторов 597

Поиск 599
Частые и редкие лексемы 600
Вставка 604
Ограничение выборки 606
Свойства 607
Ограничения GIN и RUM-индекс 609
28.3. Индекс для триграмм 610
28.4. Индекс для массивов 612
28.5. Индекс для JSON 616
Класс операторов jsonb_ops 616
Класс операторов jsonb_path_ops 619
28.6. Другие типы данных 621

Глава 29. Индекс BRIN **622**
29.1. Общий принцип 622
29.2. Пример 623
29.3. Страничная организация 625
29.4. Поиск 627
29.5. Обновление сводной информации 628
Вставка значений 628
Обобщение зоны 629
29.6. Диапазоны значений (minmax) 630
Выбор столбцов для индексирования 631
Размер зоны и эффективность поиска 632
Свойства 636
29.7. Мультидиапазоны значений (minmax-multi) 639
29.8. Охватывающие значения (inclusion) 642
29.9. Фильтры Блума (bloom) 645

Заключение **650**

Предметный указатель **651**

О книге

> — До чего же это все просто! — воскликнул Шпунтик. — А я где-то читал, что писателю нужен какой-то вымысел, замысел...
>
> — Э, замысел! — нетерпеливо перебил его Смекайло. — Это только в книгах так пишется, что нужен замысел, а попробуй задумай что-нибудь, когда все уже и без тебя задумано! Что ни возьми — все уже было.
>
> Николай Носов, *Приключения Незнайки и его друзей*

Для кого эта книга

Эта книга для тех, кого не устраивает работа с базой данных как с черным ящиком. Если вы любознательны, не довольствуетесь авторитетными советами и хотите во всем разобраться сами — нам по пути.

Я ориентируюсь на читателей, имеющих определенный опыт использования PostgreSQL и хотя бы в общих чертах представляющих себе, что к чему. Для совсем новичков текст будет тяжеловат. Например, я ни слова не скажу о том, как устанавливать сервер, вводить команды в psql или изменять конфигурационные параметры.

Надеюсь, что книга будет полезной и тем, кто хорошо знаком с устройством другой СУБД, но переходит на PostgreSQL и хочет разобраться в отличиях. Несколько лет назад такая книга сэкономила бы мне много времени. Именно поэтому я ее в конце концов и написал.

Чего нет в книге

Эта книга — не сборник рецептов. На все случаи жизни готовых решений не напасешься, а понимание внутренней механики сложной системы дает

возможность критически переосмысливать чужой опыт и делать свои собственные выводы. Поэтому я и объясняю такие подробности устройства, знание которых на первый взгляд не имеет практического смысла.

Но эта книга и не учебник. Она углубляется в одни области (более интересные мне самому) и обходит стороной другие. Если вы изучаете SQL, обратите внимание на учебник Евгения Моргунова *PostgreSQL. Основы языка SQL*[1], а необходимый теоретический фундамент даст книга Бориса Новикова *Основы технологий баз данных*[2].

Называться справочником эта книга тоже не претендует. Я старался быть точным, но у меня не было цели заменить книгой документацию, поэтому я легко опускал непринципиальные на мой взгляд подробности. В любой непонятной ситуации читайте документацию.

Еще эта книга не учит разрабатывать ядро PostgreSQL. Я не предполагаю у читателя знания языка C и ориентируюсь на администраторов и прикладных разработчиков. Хотя и ссылаюсь постоянно на исходный код, из которого можно узнать столько подробностей, сколько душе угодно, и даже больше.

Что в книге есть

Во вводной главе без особых деталей я даю основные понятия, на которые опирается все дальнейшее повествование. Я предполагаю, что вы не почерпнете из этой главы практически ничего нового, но все-таки включаю ее для полноты картины. К тому же она может пригодиться тем, кто переходит с других СУБД.

Первая часть книги посвящена вопросам согласованности и изоляции, которые я сперва рассматриваю с позиции пользователя (какие уровни изоляции существуют и чем это грозит), а затем с точки зрения внутреннего устройства. Для этого мне приходится погрузиться в детали реализации многоверсионности и изоляции на основе снимков данных. Особенно много внимания требует процедура очистки неактуальных версий строк.

[1] postgrespro.ru/education/books/sqlprimer.
[2] postgrespro.ru/education/books/dbtech.

Во второй части я рассматриваю буферный кеш и механизм, позволяющий восстанавливать согласованность после сбоев, — журнал предзаписи.

В третьей части детально разбирается устройство и использование блокировок разных уровней: легких блокировок для оперативной памяти, тяжелых блокировок для отношений, блокировок табличных строк.

Четвертая часть объясняет, как сервер планирует и выполняет SQL-запросы. Я рассказываю, какие есть способы доступа к данным, какие применяются методы соединения и как используется статистическая информация.

В пятой части обсуждение индексов, сводившееся ранее к B-деревьям, добирается и до остальных методов доступа. Сначала я рассматриваю общие принципы расширяемости, устанавливающие границы между ядром системы индексирования, индексными методами доступа и типами данных (что требует введения понятия классов операторов), а затем подробно останавливаюсь на особенностях каждого из имеющихся методов.

В состав PostgreSQL входит масса «интроспективных» расширений, которые не нужны для обычной работы, но дают возможность заглянуть во внутреннюю жизнь сервера. В книге используются многие из них. Кроме того, что эти расширения позволяют лучше изучить устройство сервера, они могут облегчить диагностику в сложных случаях.

Обозначения

Я пытался писать книгу так, чтобы ее можно было читать последовательно, страница за страницей. Но всю правду не получается раскрыть сразу, и к одной и той же теме приходится возвращаться несколько раз. Если бы я каждый раз писал «это будет рассмотрено позже», книга сильно увеличилась бы в размере, поэтому в таких случаях я ставлю на полях номер страницы, на которой тема развивается дальше. Такой же номер, ведущий назад, отсылает к месту в книге, где уже что-то говорилось о предмете обсуждения.

с. 19

Текст книги и все примеры актуальны для PostgreSQL 15. Некоторые абзацы имеют на полях отметку о номере версии. Это означает, что сказанное справедливо для версий PostgreSQL, начиная с указанной, а более ранние

v. 15

версии либо вовсе не имели описанной возможности, либо были устроены как-то иначе. Такие пометки могут оказаться полезными для тех, кто еще не обновил систему до последнего выпуска.

Также на полях указываются значения по умолчанию для обсуждаемых параметров. Сами параметры (как обычные, так и параметры хранения) выде-
4MB лены курсивом: *work_mem*.

В сносках я постоянно ссылаюсь на первоисточники. Их несколько, и на первом месте стоит кладезь полезной информации — документация[1]. Являясь органичной частью проекта, она всегда поддерживается в актуальном состоянии самими разработчиками. Но главный первоисточник — безусловно, исходный код[2]. Удивительно, на какое количество вопросов можно найти ответы просто в комментариях и файлах README, даже не владея языком C. Реже я ссылаюсь на записи коммитфеста[3]: в переписках pgsql-hackers всегда можно проследить историю изменений и понять логику принятых разработчиками решений, но ценой чтения огромного массива обсуждений.

> Лирические отступления и замечания, которые могут увести в сторону от основной мысли, но которые я не удержался и вставил в книгу, выделены так, чтобы их можно было пропустить.

Конечно, в книге много фрагментов кода, в основном на языке SQL. Код показан с приглашением =>; если необходимо, то следом за ним приведен и ответ сервера:

```
=> SELECT now();
              now
-------------------------------
 2022-12-08 22:25:30.718801+03
(1 row)
```

Если аккуратно повторять все приведенные команды в PostgreSQL 15, должен получиться такой же результат (конечно, с точностью до номеров транзакций и прочих несущественных деталей). Во всяком случае, весь код в книге — результат выполнения скрипта, содержащего ровно эти команды.

[1] postgrespro.ru/docs/postgresql/15/index.
[2] git.postgresql.org/gitweb/?p=postgresql.git;a=summary.
[3] commitfest.postgresql.org.

Когда требуется показать одновременную работу нескольких транзакций, код, выполняющийся в другом сеансе, выделен отступом и отчеркиванием:

```
=> SHOW server_version;
 server_version
----------------
 15.1
(1 row)
```

Чтобы повторить такие команды (а это полезно для самообразования, как и любые эксперименты), удобно открыть два терминала с psql.

Отдельные команды и названия различных объектов базы данных (таких как таблицы и столбцы, функции, расширения) выделены в тексте моноширинным шрифтом: `UPDATE`, `pg_class`.

Вызовы утилит из операционной системы показаны с приглашением, оканчивающимся на `$`:

```
postgres$ whoami
postgres
```

Я использую Linux, но без какой-либо специфики; достаточно будет самого базового понимания.

Благодарности

Книгу невозможно написать в одиночку, и это отличный повод сказать спасибо хорошим людям.

Я благодарен Павлу Лузанову, который в нужный момент предложил мне заняться чем-то действительно стоящим.

Я признателен компании Postgres Professional за возможность работать над этой книгой не только в свободное время. Но компания — это люди, и я хочу сказать отдельное спасибо Олегу Бартунову за неиссякаемую энергию и идеи и Ивану Панченко за всестороннюю поддержку и LaTeX.

Спасибо моим товарищам по образовательному отделу за творческую атмосферу и дискуссии, в ходе которых формировался материал учебных курсов и способ его подачи, что нашло свое отражение и в книге. Спасибо Павлу Толмачеву за внимательную вычитку черновиков.

Многие главы книги впервые были опубликованы в виде статей на Хабре[1], и я благодарен читателям за замечания и отклики. Они показали нужность этой работы, позволили разглядеть белые пятна в моих знаниях и сделать текст лучше.

Спасибо Людмиле Мантровой, проделавшей огромную работу над языком книги. Если вы не спотыкаетесь на каждом втором предложении, это ее заслуга.

В книге я не называю имен, но за каждой функцией и возможностью, про которые я пишу, стоит многолетний труд вполне конкретных людей. Я восхищаюсь разработчиками PostgreSQL, и мне особенно приятно, что многих из них я имею честь называть коллегами.

[1] habr.com/ru/company/postgrespro/blog.

1

Введение

1.1. Организация данных

Базы данных

PostgreSQL — программа, которая относится к классу систем управления базами данных. Когда эта программа выполняется, мы называем ее *сервером* PostgreSQL, или *экземпляром* сервера.

Данные, которыми управляет PostgreSQL, хранятся в базах данных[1]. Один экземпляр PostgreSQL одновременно работает с несколькими базами, которые вместе называются *кластером баз данных*.

Чтобы кластер можно было использовать, его необходимо *инициализировать*[2] (создать). Каталог, в котором размещаются все файлы, относящиеся к кластеру, обычно называют словом PGDATA, по имени переменной окружения, указывающей на этот каталог.

> Пакетные установки могут вводить свои «слои абстракции» над штатным механизмом PostgreSQL, явно прописывая все необходимые утилитам параметры. Сервер баз данных представляется в таком случае в виде службы операционной системы, и непосредственно с переменной PGDATA можно никогда не встретиться. Но сам термин устоялся, и я буду им пользоваться.

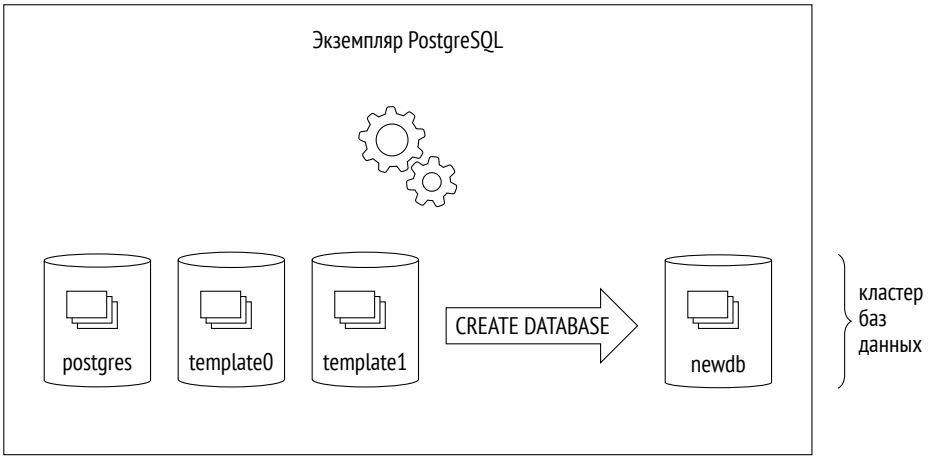
[1] postgrespro.ru/docs/postgresql/15/managing-databases.
[2] postgrespro.ru/docs/postgresql/15/app-initdb.

При инициализации в PGDATA создаются три одинаковые базы данных:

template0 используется, например, для восстановления из логической ре-
с. 611 зервной копии или для создания базы в другой кодировке и никогда не должна меняться;

template1 служит шаблоном для всех остальных баз данных, которые может создать пользователь в этом кластере;

postgres представляет собой обычную базу данных, которую можно использовать по своему усмотрению.



Системный каталог

Метаинформация обо всех объектах кластера (таких как таблицы, индексы, типы данных или функции) хранится в таблицах, относящихся к *системному каталогу*[1]. В каждой базе данных имеется собственный набор таблиц (и представлений), описывающих объекты этой конкретной базы. Существует также несколько таблиц системного каталога, общих для всего кластера, которые не принадлежат какой-либо определенной базе данных (формально используется фиктивная база с нулевым идентификатором), но доступны в любой из них.

[1] postgrespro.ru/docs/postgresql/15/catalogs.

К системному каталогу можно обращаться с помощью обычных запросов SQL, а изменения в нем происходят при выполнении команд DDL. Клиент psql располагает целым рядом специальных команд для просмотра системного каталога.

Все имена таблиц системного каталога имеют префикс `pg_`, например `pg_database`. Имена столбцов начинаются с трехбуквенного кода, который обычно соответствует имени таблицы, например `datname`.

Во всех таблицах системного каталога столбец с первичным ключом называется `oid` и имеет одноименный тип `oid` (object identifier) — целое 32-битное число.

> По сути, механизм идентификаторов объектов `oid` — аналог последовательностей, появившийся в PostgreSQL задолго до самих последовательностей. Его особенность в том, что уникальные номера используются в разных таблицах системного каталога, хотя выдаются с помощью единого счетчика. Когда общее число выданных номеров выходит за диапазон значений, счетчик обнуляется. Уникальность значений в конкретной таблице гарантируется тем, что очередное значение `oid` проверяется с помощью уникального индекса и, если оно уже используется в этой таблице, счетчик увеличивается, а проверка повторяется[1].

Схемы

Схемы[2] представляют собой пространства имен для всех объектов, хранящихся в базе данных. Кроме пользовательских схем, имеется несколько специальных служебных:

public используется по умолчанию для пользовательских объектов, если не выполнены иные настройки;

pg_catalog используется для таблиц системного каталога;

information_schema дает альтернативное представление системного каталога, регламентируемое стандартом SQL;

pg_toast используется для объектов, относящихся к TOAST; *с. 33*

[1] backend/catalog/catalog.c, функция GetNewOidWithIndex.
[2] postgrespro.ru/docs/postgresql/15/ddl-schemas.

pg_temp объединяет временные таблицы (хотя временные таблицы разных пользователей создаются в разных схемах `pg_temp_N`, каждый обращается к своим объектам, используя имя `pg_temp`).

Схемы существуют внутри базы данных, и все объекты базы принадлежат каким-либо схемам.

Если при обращении к объекту схема не указана явно, выбирается первая подходящая схема из перечисленных в *пути поиска*. Путь формируется на основе значения параметра *search_path*, к которому, в частности, неявно добавляются схемы `pg_catalog` и (при необходимости) `pg_temp`. Это позволяет иметь в разных схемах объекты с одинаковыми именами.

Табличные пространства

В отличие от логического распределения объектов по базам данных и схемам, *табличные пространства* определяют физическое расположение данных. Фактически табличное пространство — это каталог файловой системы. Например, табличные пространства можно использовать, чтобы разместить архивные данные на медленных носителях, а данные, с которыми идет активная работа, — на быстрых.

Одно и то же табличное пространство может использоваться разными базами данных, а одна база данных может хранить данные в нескольких табличных пространствах. То есть логическая и физическая структуры не зависят друг от друга.

У каждой базы данных есть так называемое *табличное пространство по умолчанию*, в котором создаются все объекты, если явно не указать иное. В этом же табличном пространстве хранятся и объекты системного каталога.

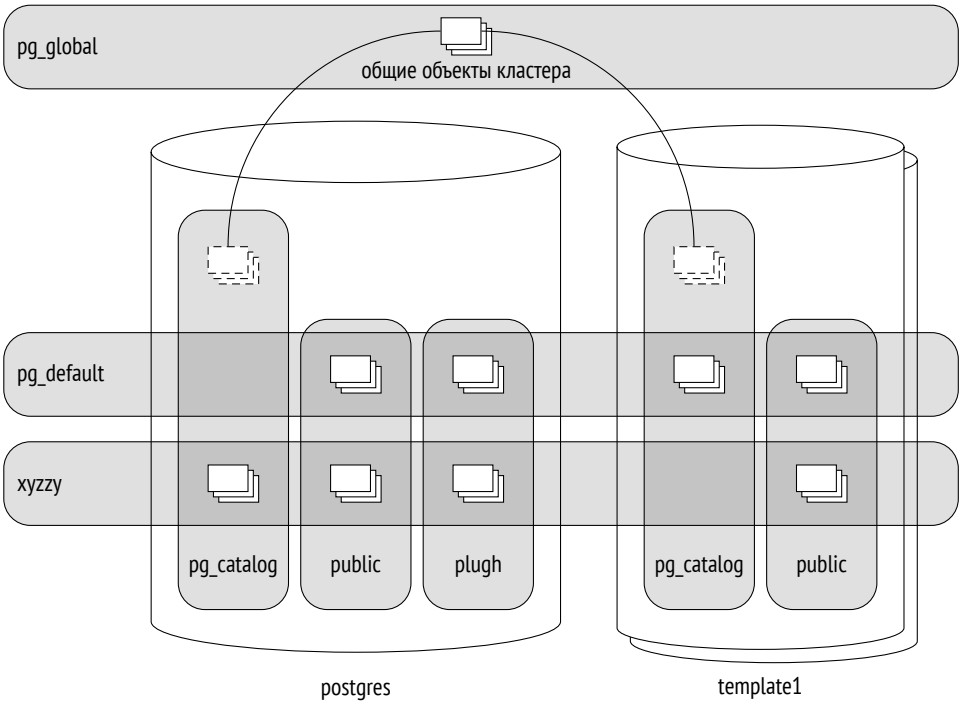
При инициализации кластера создаются два табличных пространства:

pg_default располагается в каталоге PGDATA/base и используется как табличное пространство по умолчанию, если явно не выбрать для этой цели другое пространство;

pg_global располагается в каталоге PGDATA/global и хранит общие для всего кластера объекты системного каталога.

При создании пользовательского табличного пространства указывается произвольный каталог; PostgreSQL создает на него символьную ссылку в каталоге PGDATA/pg_tblspc. Вообще, все пути PostgreSQL относительные и отсчитываются от каталога данных PGDATA. Благодаря этому можно переносить PGDATA на другое место (конечно, предварительно остановив сервер).

Рисунок сводит воедино базы данных, схемы и табличные пространства. Здесь база данных `postgres` использует табличное пространство по умолчанию `xyzzy`, а база `template1` — `pg_default`. На пересечении табличных пространств и схем находятся различные объекты базы данных:



Отношения

Самые важные объекты базы данных — *таблицы* и *индексы* — при всех своих различиях схожи в том, что состоят из строк. Для таблиц это очевидно, но и

с. 507 в B-деревьях узлы состоят из строк, содержащих индексированные значения и ссылки на другие узлы или на табличные строки.

Есть еще некоторое количество объектов, устроенных похожим образом: *последовательности* (по сути однострочные таблицы), *материализованные представления* (по сути таблицы, которые помнят запрос). А еще есть обычные *представления*, которые сами по себе не хранят данные, но во всех остальных смыслах похожи на таблицы.

Все эти объекты в PostgreSQL называются общим словом *отношение* (*relation*).

> Слово, на мой взгляд, выбрано неудачно, поскольку смешивает таблицы базы данных и (настоящие) отношения реляционной теории. Дает о себе знать университетское происхождение проекта и склонность его основателя, Майкла Стоунбрейкера, во всем видеть отношения. В одной из работ он даже ввел понятие «упорядоченного отношения» (ordered relation) для таблицы, в которой порядок строк задается индексом.
>
> Таблица системного каталога для отношений изначально называлась `pg_relation`, но довольно скоро, на волне увлечения объектной ориентированностью, ее переименовали в привычный нам сейчас `pg_class`. Однако столбцы этой таблицы по-прежнему имеют префикс `rel`.

Слои и файлы

Информация, связанная с отношением, организована в несколько *слоев*[1] (*forks*) разных типов, каждый из них содержит определенный вид данных.

Изначально слой представлен единственным *файлом*. Имя файла состоит из числового идентификатора (`oid`), к которому может быть добавлен суффикс, соответствующий типу слоя.

Файл постепенно растет, и когда его размер достигает 1 Гбайта, создается следующий файл этого же слоя (такие файлы иногда называют *сегментами*). Порядковый номер сегмента добавляется в конец имени файла.

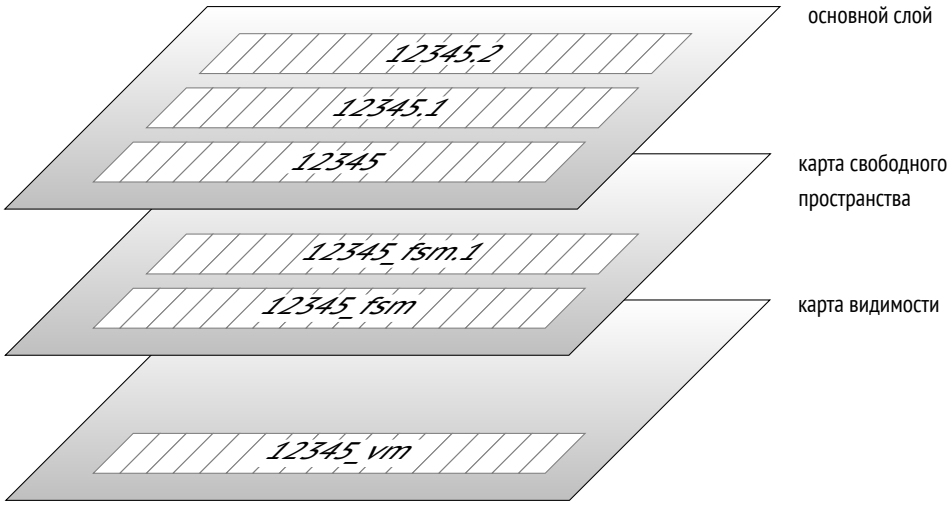
Ограничение размера файла в 1 Гбайт возникло исторически для поддержки различных файловых систем, некоторые из которых не умеют работать

[1] postgrespro.ru/docs/postgresql/15/storage-file-layout.

с файлами большого размера. Ограничение можно изменить при сборке PostgreSQL (`./configure --with-segsize`).

Таким образом, одному отношению на диске может соответствовать несколько файлов. Даже для небольшой таблицы без индексов их будет минимум три, по числу обязательных слоев.

Внутри каталога каждого табличного пространства (кроме `pg_global`) создаются подкаталоги для отдельных баз данных. Все файлы объектов, принадлежащих одному табличному пространству и одной базе данных, будут помещены в один подкаталог. Это необходимо учитывать, потому что файловые системы обычно не очень хорошо работают с большим количеством файлов в каталоге.



Имеется несколько стандартных типов слоев.

Основной слой (main fork) — это собственно данные: те самые табличные или индексные строки. Основной слой существует для любых отношений (кроме представлений, которые не содержат данных).

Имена файлов основного слоя состоят только из числового идентификатора (равного значению столбца `relfilenode` таблицы `pg_class`).

Посмотрим для примера путь к файлу таблицы, созданной в табличном пространстве `pg_default`:

```
=> CREATE UNLOGGED TABLE t(
  a integer,
  b numeric,
  c text,
  d json
);
=> INSERT INTO t VALUES (1, 2.0, 'foo', '{}');
=> SELECT pg_relation_filepath('t');
 pg_relation_filepath
----------------------
 base/16384/16385
(1 row)
```

Каталог base соответствует табличному пространству pg_default, следующий подкаталог — базе данных, и уже в нем находится интересующий нас файл:

```
=> SELECT oid FROM pg_database WHERE datname = 'internals';
  oid
-------
 16384
(1 row)
=> SELECT relfilenode FROM pg_class WHERE relname = 't';
 relfilenode
-------------
       16385
(1 row)
```

Вот соответствующий файл в файловой системе:

```
=> SELECT size
FROM pg_stat_file('/usr/local/pgsql/data/base/16384/16385');
 size
------
 8192
(1 row)
```

Слой инициализации (init fork[1]) существует только для нежурналируемых таблиц (созданных с указанием UNLOGGED) и их индексов. Такие объекты

[1] postgrespro.ru/docs/postgresql/15/storage-init.

ничем не отличаются от обычных, кроме того, что действия с ними не записываются в журнал предзаписи. За счет этого работа с ними происходит быстрее, но в случае сбоя невозможно восстановить данные в согласованном состоянии. Поэтому при восстановлении PostgreSQL просто удаляет все слои таких объектов и записывает слой инициализации на место основного слоя. В результате получается «пустышка». *с. 198*

Таблица `t` создана нежурналируемой, поэтому у нее есть слой инициализации. Он имеет такое же имя, как и основной слой, но с суффиксом _init:

```
=> SELECT size
FROM pg_stat_file('/usr/local/pgsql/data/base/16384/16385_init');
 size
------
    0
(1 row)
```

Карта свободного пространства (free space map[1]) — слой, в котором отслеживается примерный объем свободного места внутри страниц. Этот объем постоянно меняется: при добавлении новых версий строк уменьшается, при очистке — увеличивается. Карта свободного пространства используется при вставке новых версий строк, чтобы быстро найти подходящую страницу, на которую поместятся добавляемые данные.

Файлы, относящиеся к карте свободного пространства, имеют суффикс _fsm. Но появляются они не сразу, а только при необходимости. Самый простой способ добиться этого — выполнить очистку таблицы: *с. 132*

```
=> VACUUM t;
=> SELECT size
FROM pg_stat_file('/usr/local/pgsql/data/base/16384/16385_fsm');
 size
-------
 24576
(1 row)
```

[1] postgrespro.ru/docs/postgresql/15/storage-fsm; backend/storage/freespace/README.

Для ускорения поиска карта свободного пространства организована как дерево и занимает минимум три страницы (отсюда и размер файла слоя для почти пустой таблицы).

Карта свободного пространства существует не только для таблиц, но и для индексов. Но поскольку индексную строку нельзя добавить на произвольную страницу (например, для B-дерева место вставки определяется порядком сортировки), отслеживаются только те страницы, которые были полностью очищены и могут быть снова задействованы в индексной структуре.

Карта видимости (visibility map[1]) — слой, который позволяет быстро определить, требует ли страница очистки или заморозки. Для этого на каждую табличную страницу в этом слое отведено два бита.

Одним битом отмечены страницы, которые содержат только актуальные версии строк. Процесс очистки пропускает такие страницы, поскольку в них нечего очищать. Кроме того, когда транзакция пытается прочитать строку из такой страницы, можно не проверять ее видимость, а это позволяет использовать сканирование только индекса.

с. 130, *с. 403*

Второй бит отмечает страницы, на которых все версии строк заморожены. Эту часть слоя я буду называть *картой заморозки*.

v. 9.6, *с. 149*

Файлы карты видимости имеют суффикс _vm. Обычно они самые небольшие по размеру:

```
=> SELECT size
FROM pg_stat_file('/usr/local/pgsql/data/base/16384/16385_vm');
 size
------
 8192
(1 row)
```

Карта видимости существует для таблиц, но не для индексов.

с. 89

[1] postgrespro.ru/docs/postgresql/15/storage-vm.

Страницы

Для удобства организации ввода-вывода файлы логически поделены на *страницы* (или *блоки*) — это минимальный объем данных, который считывается или записывается. Соответственно, и многие внутренние алгоритмы PostgreSQL ориентированы на работу со страницами. *с. 75*

Обычно страница имеет размер 8 Кбайт. Его можно поменять в некоторых пределах (до 32 Кбайт), но только при сборке (`./configure --with-blocksize`), и, как правило, никто этим не занимается. Собранный и запущенный экземпляр может работать со страницами только одного размера — создать табличные пространства с разноразмерными страницами нельзя.

Независимо от того, к какому слою принадлежат файлы, они используются сервером примерно одинаково. Страницы сначала помещаются в буферный кеш (где их могут читать и изменять процессы), а затем при необходимости вытесняются обратно на диск. *с. 177*

TOAST

Каждая строка должна помещаться целиком на одну страницу: нет способа «продолжить» строку на следующей странице. Для длинных строк используется технология, названная TOAST[1] (The Oversized Attributes Storage Technique).

TOAST подразумевает несколько стратегий. Длинные значения атрибутов можно отправить в отдельную служебную таблицу, предварительно нарезав на небольшие фрагменты-тосты. Другой вариант — сжать длинное значение так, чтобы строка все-таки поместилась на одну страницу. А можно и то, и другое: сначала сжать, а уже потом нарезать и отправить.

Если основная таблица содержит потенциально длинные атрибуты, для нее сразу же создается отдельная toast-таблица (одна для всех атрибутов).

[1] postgrespro.ru/docs/postgresql/15/storage-toast; include/access/heaptoast.h.

Например, если в таблице есть столбец типа `numeric` или `text`, toast-таблица будет создана, даже если в таком столбце никогда не будут храниться длинные значения.

Для индексов технология TOAST предлагает только сжатие; вынесение атрибутов в отдельную таблицу не поддерживается. Это накладывает ограничение на размер индексируемых ключей (как это ограничение реализуется,
с. 378 зависит от класса операторов).

Изначально стратегии определяются типами столбцов. Посмотреть стратегии проще всего командой `\d+` в psql, но для сокращения вывода я использую запрос к системному каталогу:

```
=> SELECT attname, atttypid::regtype,
  CASE attstorage
    WHEN 'p' THEN 'plain'
    WHEN 'e' THEN 'external'
    WHEN 'm' THEN 'main'
    WHEN 'x' THEN 'extended'
  END AS storage
FROM pg_attribute
WHERE attrelid = 't'::regclass AND attnum > 0;
 attname | atttypid | storage
---------+----------+----------
 a       | integer  | plain
 b       | numeric  | main
 c       | text     | extended
 d       | json     | extended
(4 rows)
```

Стратегии состоят в следующем:

plain — TOAST не используется (стратегия применяется для заведомо «коротких» типов данных, как integer);

extended — допускается как сжатие, так и хранение в отдельной toast-таблице;

external — длинные значения хранятся в toast-таблице несжатыми;

main — длинные значения в первую очередь сжимаются, а в toast-таблицу попадают, только если сжатие не помогло.

В общих чертах алгоритм выглядит следующим образом[1]. PostgreSQL стремится к тому, чтобы на странице помещалось хотя бы четыре строки. Поэтому если размер строки превышает четвертую часть страницы за вычетом заголовка (для страницы стандартного размера это около 2000 байт), к части значений необходимо применить TOAST. Действуем в порядке, описанном ниже, и прекращаем, как только длина строки перестает превышать пороговое значение:

1. Сначала перебираем атрибуты со стратегиями external и extended, двигаясь от самых длинных к более коротким. Extended-атрибуты сжимаются, и если после этого значение (само по себе, без учета других атрибутов) превосходит четверть страницы, оно сразу же отправляется в toast-таблицу. External-атрибуты обрабатываются так же, но не сжимаются.

2. Если после первого прохода строка все еще не помещается, по одному отправляем в toast-таблицу оставшиеся атрибуты со стратегиями external и extended.

3. Если и это не помогло, пытаемся сжать атрибуты со стратегией main, оставляя их при этом в табличной странице.

4. Если строка все равно недостаточно коротка, отправляем в toast-таблицу main-атрибуты.

Пороговое значение составляет все те же 2000 байт, но может переопределяться параметром хранения *toast_tuple_target* на уровне таблицы. v. 11

Иногда может оказаться полезным изменить стратегию для некоторых столбцов. Если заранее известно, что данные в столбце не сжимаются (например, в столбце хранятся JPEG-изображения), можно установить для него стратегию external — это позволит сэкономить на бесполезных попытках сжатия. Изменить стратегию можно следующим образом:

```
=> ALTER TABLE t ALTER COLUMN d SET STORAGE external;
```

Повторив запрос, получим:

[1] backend/access/heap/heaptoast.c.

```
 attname | atttypid | storage
---------+----------+----------
 a       | integer  | plain
 b       | numeric  | main
 c       | text     | extended
 d       | json     | external
(4 rows)
```

v. 15 Для сжатия доступны два алгоритма: традиционный PGLZ и более современный LZ4 (оба являются разновидностями алгоритма словарного сжатия Лемпеля-Зива). По умолчанию сжатие выполняется алгоритмом, указанным
pglz в параметре *default_toast_compression*; значение можно установить для отдельных столбцов в предложении COMPRESSION. Эталонные тесты показывают, что при сходном с PGLZ уровне сжатия LZ4 тратит меньше ресурсов процессора.

Toast-таблицы располагаются в отдельной схеме pg_toast, не входящей в путь поиска, и поэтому обычно не видны. Для временных таблиц используется схема pg_toast_temp_*N* аналогично обычной pg_temp_*N*.

Конечно, при желании всегда можно подглядеть за внутренней механикой процесса. Скажем, в таблице t есть три потенциально длинных атрибута, поэтому toast-таблица обязана быть. Вот она:

```
=> SELECT relnamespace::regnamespace, relname
FROM pg_class WHERE oid = (
  SELECT reltoastrelid FROM pg_class WHERE relname = 't'
);
 relnamespace |    relname
--------------+----------------
 pg_toast     | pg_toast_16385
(1 row)

=> \d+ pg_toast.pg_toast_16385
TOAST table "pg_toast.pg_toast_16385"
   Column   |  Type   | Storage
------------+---------+---------
 chunk_id   | oid     | plain
 chunk_seq  | integer | plain
 chunk_data | bytea   | plain
Owning table: "public.t"
```

```
Indexes:
    "pg_toast_16385_index" PRIMARY KEY, btree (chunk_id, chunk_seq)
Access method: heap
```

Логично, что для «тостов», на которые нарезается строка, применяется стратегия plain: TOAST второго уровня не существует.

Вместе с toast-таблицей в той же схеме создается и индекс, который *всегда* используется для доступа к фрагментам значений. Имя индекса видно в выводе команды, но его можно найти и запросом:

```
=> SELECT indexrelid::regclass FROM pg_index
WHERE indrelid = (
  SELECT oid FROM pg_class WHERE relname = 'pg_toast_16385'
);
          indexrelid
-------------------------------
 pg_toast.pg_toast_16385_index
(1 row)

=> \d pg_toast.pg_toast_16385_index
Unlogged index "pg_toast.pg_toast_16385_index"
  Column   |  Type   | Key? | Definition
-----------+---------+------+------------
 chunk_id  | oid     | yes  | chunk_id
 chunk_seq | integer | yes  | chunk_seq
primary key, btree, for table "pg_toast.pg_toast_16385"
```

Таким образом, toast-таблица увеличивает минимальное число файлов, «обслуживающих» таблицу, до восьми: три слоя основной таблицы, три слоя toast-таблицы и два слоя toast-индекса.

Столбец c использует стратегию extended, значения в нем будут сжиматься:

```
=> UPDATE t SET c = repeat('A',5000);
=> SELECT * FROM pg_toast.pg_toast_16385;
 chunk_id | chunk_seq | chunk_data
----------+-----------+------------
(0 rows)
```

В toast-таблице ничего нет: повторяющиеся символы сжались алгоритмом LZ, и после этого значение поместилось в обычной табличной странице.

А теперь составим значение из случайных символов:

```
=> UPDATE t SET c = (
  SELECT string_agg( chr(trunc(65+random()*26)::integer), '')
  FROM generate_series(1,5000)
)
RETURNING left(c,10) || '...' || right(c,10);
        ?column?
-------------------------
 BZFWEEEUJF...VNRDRIBMHR
(1 row)
UPDATE 1
```

Такую последовательность сжать не получается, и она попадает в toast-таблицу:

```
=> SELECT chunk_id,
  chunk_seq,
  length(chunk_data),
  left(encode(chunk_data,'escape')::text, 10) || '...' ||
  right(encode(chunk_data,'escape')::text, 10)
FROM pg_toast.pg_toast_16385;
 chunk_id | chunk_seq | length |        ?column?
----------+-----------+--------+-------------------------
    16390 |         0 |   1996 | BZFWEEEUJF...GLDYGAJJSS
    16390 |         1 |   1996 | ETTSPZBNFE...LJWWSFDKPH
    16390 |         2 |   1008 | CXCPSTBNSK...VNRDRIBMHR
(3 rows)
```

Видно, что данные нарезаны на фрагменты. Размер фрагментов выбирается так, чтобы на странице toast-таблицы помещалось четыре строки. От версии к версии это значение может немного меняться в зависимости от размера заголовка страницы.

При обращении к «длинному» атрибуту PostgreSQL автоматически, прозрачно для приложения, восстанавливает исходное значение и возвращает его клиенту. Если же такие атрибуты не участвуют в запросе, то toast-таблица не читается. Это одна из причин не использовать «звездочку» в производственном коде.

v. 13 Если клиент запрашивает начальную часть длинного значения, то будут прочитаны только необходимые фрагменты, в том числе и в случае, когда значение хранится в сжатом виде.

Тем не менее и на сжатие с нарезкой, и на последующее восстановление тратится довольно много ресурсов. Поэтому хранить объемные данные в PostgreSQL — не лучшая идея, особенно если они активно используются и при этом для них не требуется транзакционная логика (как пример: отсканированные оригиналы бухгалтерских документов). Потенциально более выгодная альтернатива — хранить такие данные в файловой системе, а в базе данных держать только имена соответствующих файлов. Правда, тогда СУБД не сможет обеспечивать согласованность данных.

1.2. Процессы и память

Экземпляр сервера PostgreSQL состоит из нескольких взаимодействующих процессов.

В первую очередь при старте сервера запускается процесс postgres, традиционно называемый postmaster. Postmaster запускает все остальные процессы (в Unix-подобных системах для этого используется системный вызов `fork`) и «присматривает» за ними — если какой-нибудь процесс завершится аварийно, postmaster перезапустит его (или весь сервер, если сочтет, что процесс мог повредить общие данные).

> Процессная модель применяется в PostgreSQL с самого начала проекта из-за своей простоты, и все это время не прекращаются дискуссии о переходе к использованию потоков.
>
> У текущей модели есть ряд недостатков: статически выделяемая общая память не позволяет на лету менять размер таких структур, как буферный кеш; параллельные алгоритмы сложны в реализации и менее эффективны, чем могли бы быть; сеансы жестко привязаны к процессам. Использование потоков выглядит многообещающе, хотя и чревато сложностями с изолированностью, совместимостью с операционными системами, управлением ресурсами. Не говоря уже о том, что переход потребует радикальных изменений в коде и годы работы. Пока побеждает консервативный взгляд, и в ближайшее время никаких изменений не предвидится.

Работу сервера обеспечивает ряд служебных процессов. Основные из них:

startup восстанавливает систему после сбоев; *с. 210*

с. 133 **autovacuum** очищает таблицы и индексы от неактуальных данных;

с. 222 **wal writer** записывает на диск журнальные записи;

с. 206 **checkpointer** выполняет контрольную точку;

с. 213 **writer** записывает грязные страницы на диск;

wal sender передает журнальные записи на реплику;

wal receiver принимает журнальные записи на реплике.

Некоторые из этих процессов завершаются после выполнения своей задачи, другие работают постоянно в фоновом режиме, а какие-то могут быть отключены.

> Каждый процесс управляется конфигурационными параметрами, иногда десятками параметров. Чтобы осмысленно выполнять настройку сервера, нужно хорошо представлять его внутреннее устройство. Но из общих соображений можно выбрать лишь начальные адекватные значения параметров, которые затем потребуется уточнять, получая обратную связь от мониторинга.

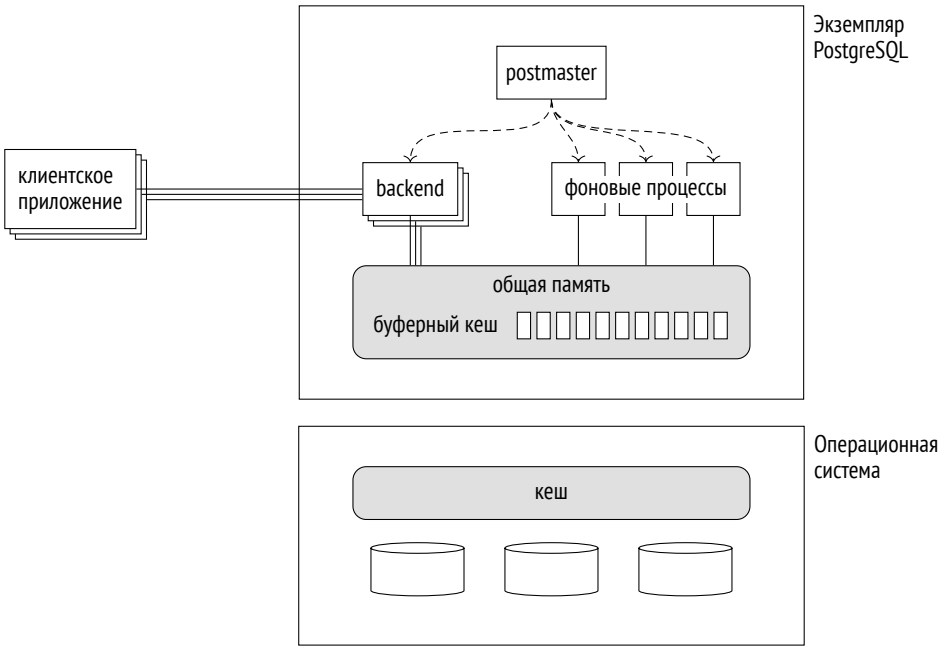
Чтобы процессы могли обмениваться информацией, postmaster выделяет *общую память*, которая доступна всем процессам.

Из-за того, что диски (особенно HDD, но и SSD тоже) работают значитель-
с. 177 но медленнее, чем оперативная память, применяется кеширование: в общей области оперативной памяти отводится место под недавно прочитанные страницы в надежде, что они еще не раз понадобятся и можно будет сэкономить на повторном обращении к диску. Измененные данные также записываются на диск не сразу, а через некоторое время.

Буферный кеш занимает бóльшую часть общей памяти. В ней же располагаются и другие буферы, которые используются сервером для ускорения работы с диском.

Свой кеш имеется и у операционной системы. PostgreSQL не использует (почти) прямой ввод-вывод в обход механизмов операционной системы, поэтому кеширование получается двойным.

При сбое (например, при аварийном отключении питания или крахе операционной системы) содержимое оперативной памяти, включая буферный



кеш, пропадает. На диске остаются файлы, страницы которых записаны в разные моменты времени. Чтобы иметь возможность восстановить согласованность данных, в процессе работы PostgreSQL ведет *журнал предзаписи* (WAL), позволяющий при необходимости выполнить потерянные операции повторно. *с. 198*

1.3. Клиенты и клиент-серверный протокол

Еще одна задача процесса postmaster — слушать входящие соединения. При появлении нового клиента postmaster порождает для него *обслуживающий процесс*[1] (backend). Клиент устанавливает соединение и начинает *сеанс* общения со своим серверным процессом. Сеанс продолжается до отключения клиента или разрыва связи.

[1] backend/tcop/postgres.c, функция PostgresMain.

Для каждого клиента на сервере порождается собственный обслуживающий процесс. Большое количество подключений может вызывать проблемы:

- Каждому процессу требуется оперативная память для хранения кеша системного каталога, подготовленных запросов, промежуточных результатов при выполнении запросов и других данных. Чем больше соединений открыто, тем больше должно быть доступной памяти.

с. 321
с. 320

- Если соединения выполняются часто, а сеансы при этом короткие (то есть клиент выполняет один небольшой запрос и отключается), непозволительно много ресурсов будет тратиться на установление соединения, порождение нового процесса и ненужное заполнение локальных кешей.

- Чем больше запущено процессов, тем больше времени требуется на просмотр их списка, а эта операция выполняется очень часто. В результате с увеличением числа клиентов производительность может падать.

с. 99

В таких случаях используют *пул соединений*, чтобы ограничить число обслуживающих процессов. PostgreSQL не имеет встроенного пула соединений, поэтому приходится применять сторонние решения: менеджеры пулов, встроенные в сервер приложений, или внешние программы (такие как PgBouncer[1] или Odyssey[2]). При этом, как правило, один процесс на сервере поочередно выполняет транзакции разных клиентов. Это накладывает определенные ограничения на разработку приложений, позволяя использовать средства, которые локализованы в пределах транзакции, но не сеанса.

Чтобы клиент и сервер понимали друг друга, они должны использовать один и тот же протокол взаимодействия[3]. Обычно для реализации протокола используют штатную библиотеку libpq, хотя встречаются и независимые реализации.

Говоря в самых общих чертах, протокол позволяет клиенту подключиться к серверу и выполнять SQL-запросы.

[1] pgbouncer.org.
[2] yandex.ru/dev/odyssey.
[3] postgrespro.ru/docs/postgresql/15/protocol.

Подключение всегда выполняется под определенной ролью (пользователем) и к конкретной базе данных. Несмотря на то что сервер работает с кластером баз данных, для использования в приложении сразу нескольких баз придется выполнить отдельное подключение к каждой из них. При подключении выполняется *аутентификация*: обслуживающий процесс удостоверяется, что пользователь является тем, за кого себя выдает (например, спросив пароль), и проверяет полномочия пользователя на подключение к серверу и к выбранной базе данных.

SQL-запросы передаются обслуживающему процессу в текстовом виде. Процесс разбирает текст, оптимизирует запрос, выполняет его и возвращает результат клиенту.