1 Почему важен PostgreSQL — и почему нужно говорить об ошибках

В этой главе

- ✓ Каковы основы PostgreSQL
- ✓ Как выявлять ошибки PostgreSQL и почему важно их обсуждать
- ✓ Какие бывают ошибки PostgreSQL
- ✓ Как использовать эту книгу для обучения

Добро пожаловать на страницы книги *«Антипаттерны PostgreSQL и как их избежать»*! Наверняка вы приобрели эту книгу, чтобы узнать больше о PostgreSQL, и речь пойдет действительно о нем. Однако это не учебник в традиционном смысле и не руководство по администрированию — уже есть хорошие книги с таким ракурсом. Эта книга подчеркивает, что в реальности ошибки случаются, но на них хорошо учиться. В этой главе мы увидим, почему СУБД PostgreSQL актуальна именно сейчас, как эта книга рассматривает ошибки, связанные с PostgreSQL, и почему стоит обращать на них внимание.

1.1. Почему важно изучать PostgreSQL

Как мы уже упоминали ранее, PostgreSQL (также известный как Postgres) — это инновационная технология, которая начала смещать крупных поставщиков баз

данных с их позиций. Это надежная и стабильная база данных с богатым функционалом, способным значительно расширяться, и с каждым новым выпуском она приобретает все больше возможностей продукта уровня Enterprise. Эта технология получила широкое признание не только в сообществе специалистов по СУБД, но и в коммерческих кругах: сотни вендоров теперь предлагают помощь в разработке СУБД, услуги техподдержки, версии с расширенными функциями и даже хостинг СУБД, включая предоставление облачных или DBaaS-решений (database-as-a-service) на основе Postgres.

Важно отметить, что проект PostgreSQL — яркий пример свободного ПО, который управляется и развивается сообществом, что выражено в лицензии PostgreSQL. Эта лицензия особенно значима на фоне того, что большинство других поставщиков СУБД придерживаются стратегии проприетарности ПО. Кроме того, стремясь увеличить собственную прибыль и ограничить заработок конкурентов, многие новые и давно работающие проекты в области СУБД переходят на лицензии, несовместимые с принципами ПО на базе открытого исходного кода. Вы же можете свободно извлекать прибыль из PostgreSQL, продавая поддержку, хостинг, обучение или услуги; даже создать на его основе собственный продукт с открытым или закрытым исходным кодом без риска неприятных юридических сюрпризов. Отсутствие привязки к конкретному поставщику в сочетании с возможностью полностью построить свой бизнес на Postgres без лицензионных ограничений — это мощный фактор, ярко выделяющий PostgreSQL на фоне других конкурирующих СУБД.

Такой подход к лицензированию меняться не будет. Джонатан Кац (Jonathan Katz) из основной команды разработчиков отметил в марте 2024 года: «Проект PostgreSQL начинался как совместная инициатива с открытым исходным кодом и создан так, чтобы никто не смог его полностью контролировать. Этот принцип сохраняется все 30 лет и даже закреплен во всех политиках проекта» (http://jkatz05.com/post/postgres/postgres-license-2024/).

Мощность и расширяемость PostgreSQL позволили ему найти применение в самом широком спектре отраслей и приложений. В частности, крупные банковские, кредитные и розничные системы применяют ее в OLTP-сценариях (Online Transaction Processing, оперативная обработка транзакций) и пользуются его расширенными возможностями составлять запросы и формировать отчеты для масштабной бизнес-аналитики, включая OLAP-процессы (Online Analytical Processing, процессы оперативной аналитики). Такие расширения, как PostGIS для географических информационных систем (ГИС, GIS), колоночные СУБД Citus, TimescaleDB для работы с временными рядами и pgvector для векторного поиска, добавили множество специализированных функций, что еще больше расширило пользовательскую базу. Сетевые отели, киностудии, транспортные компании, страховые фирмы, государственные организации, медицинские учреждения и космические агентства используют ту или иную форму Postgres,

и этот список можно продолжать. Для этих задач чрезвычайно важны надежность, масштабируемость и отказоустойчивость. Разумеется, экосистему дополняют множество крупных и мелких вендоров, консультантов и подрядчиков, которые обеспечивают работу всех этих приложений.

1.2. Почему важно говорить об ошибках в PostgreSQL

Популярность PostgreSQL растет, ежедневно увеличивается число вариантов его применения, и мы наблюдаем множество пользователей, которые пытаются применять практики, усвоенные при работе с другими СУБД, или неправильно понимают особенности Postgres. Эти практики могут выражаться в том, что люди повторяют что-то, что работало в другой СУБД, или используют функциональность, не понимая полностью ее документации. Например, PostgreSQL поддерживает такие сложные типы данных, как массивы, тогда как в других СУБД пришлось бы вставлять несколько строк с повторяющимися данными. Иногда это просто случай использования неподходящего инструмента для задачи — скажем, неправильной функции или типа данных. Также может быть, что выбранное решение работает сейчас, но откажется работать в будущем по различным причинам, таким как чрезмерное увеличение размера таблицы.

Эти и другие факторы часто приводят к дорогостоящим ошибкам, если они происходят в производственной системе. Поэтому лучше знать об этих ошибках заранее и вовремя их выявлять, поскольку исправление также может отнимать много времени и быть связано с высокой степенью сложности или риска, например, если изменения вносятся в работающую систему.

Важно распространять информацию о потенциальных ошибках и подводных камнях, чтобы:

- сэкономить человеко-часы, которые будут затрачены на устранение последствий;
- безопасно хранить данные;
- гарантировать долгосрочную жизнеспособность проектов баз данных;
- устранить антипаттерны, вредные для проектирования и производительности;
- применять стандартные решения, а не изобретать велосипед;
- повысить осведомленность о лучших практиках работы с базами данных в вашей команде или организации;
- максимально эффективно использовать возможности PostgreSQL.

Также важно защищать репутацию вашей любимой СУБД! Слишком часто мы видим сообщения в блогах или статьи, критикующие PostgreSQL, авторы которых не поняли технологию. Пользователи пытаются сделать что-то нестандартным способом, терпят неудачу и приходят к ошибочному выводу, что PostgreSQL плох.

1.3. Что вы узнаете

Книга «*Антипаттерны PostgreSQL и как их избежать*», надеюсь, поможет вам:

- узнать о потенциальных подводных камнях PostgreSQL, относящихся к различным аспектам конфигурации, администрирования и эксплуатации СУБД;
- научиться избегать упомянутых проблем до того, как проявятся последствия, или применять обходные решения, чтобы их исправить;
- приобрести более широкое понимание фундаментальных принципов работы баз данных, чтобы применять лучшие практики;
- глубже понять особенности и нюансы PostgreSQL, которые отличают ее от других баз данных;
- проследить, как лучшие практики влияют на использование PostgreSQL и его производительность.

Эти знания позволят вам в повседневной работе:

- экономить время и усилия, применяя лучшие практики и связанные с ними шаблоны использования;
- действовать на опережение и решать потенциальные проблемы до того, как они станут разрушительными;
- принимать профилактические меры и защищать себя от неправильного использования базы данных другими (случайно или умышленно);
- обучать коллег, делясь этими знаниями, и сделать опыт работы с PostgreSQL более продуктивным и приятным для всех.

1.4. Типичные виды ошибок PostgreSQL

Давайте посмотрим, как можно случайно сделать ошибку в PostgreSQL.

1.4.1. Ожидания от других баз данных

Многие новички в Postgres чувствуют себя сбитыми с толку и разочарованными, делая первые шаги в работе с этой базой данных. Одна из возможных причин заключается в том, что другие базы данных часто требуют длительного

и сложного мастера установки, где вам приходится принимать важные (и часто необратимые) решения, прежде чем вы сможете приступить к использованию СУБД. В случае с Postgres многие пользователи поражаются тому, что в большинстве случаев, как только вы ее установили, вы можете сразу же начать работать с сервером базы данных. Все просто: подключите клиент базы данных к серверу, и можно начать исследовать возможности. Однако это сопряжено с риском: Postgres устанавливается с некоторыми разумными значениями по умолчанию (sensible defaults). Насколько они приемлемы, зависит от того, для чего вы планируете применять базу данных. Конфигурация по умолчанию почти наверняка не подходит для большинства сценариев использования в условиях продакшен-среды.

Еще один возможный источник путаницы — то, что большинство СУБД имеют собственные подходы к выполнению операций, часто отклоняющиеся от стандарта SQL (международного стандарта языка структурированных запросов; в настоящее время ISO/IEC 9075:2023) по историческим, коммерческим или техническим причинам реализации. У PostgreSQL также есть особенности работы, но при этом он общепризнан как наиболее соответствующая стандарту SQL база данных из доступных сейчас. Как следствие, люди иногда ожидают увидеть то, к чему привыкли при работе с другими СУБД. Например, пользователи могут ожидать, что создание пользователя базы данных автоматически создаст схему с тем же именем[1] и неявно назначит этому пользователю право владения всеми объектами в ней. Кроме того, хотя некоторые базы данных автоматически и без предупреждений преобразуют типы данных (например, позволяя вставлять значения 1 и 0 как целые числа в столбец типа `BOOLEAN`), PostgreSQL сообщит об ошибке в такой ситуации. Вполне закономерно, что пользователи удивляются, обнаружив, что ни одна из этих ожидаемых возможностей в PostgreSQL не работает.

1.4.2. Непонимание особенностей PostgreSQL

PostgreSQL — это СУБД общего назначения. Тем не менее это не означает, что она подходит для абсолютно любого мыслимого применения. Автору доводилось наблюдать, как люди пытались использовать ее:

- в качестве встраиваемой (embedded) базы данных;
- в качестве распределенной базы данных с собственной репликацией;
- в качестве сервера журналирования (лог-сервера);
- для хранения объемных видеофайлов (например, фильмов);

[1] Стандартное поведение Oracle. — *Примеч. науч. ред.*

- для сценариев работы в памяти (in-memory DB), для которых лучше подходит Redis;
- в качестве графовой базы данных (каюсь, грешен).

Да, в определенной степени она способна выполнять все эти задачи. Однако всегда следует осознавать, какие решения доступны для конкретной ниши, которую вы пытаетесь занять, и взвешивать все «за» и «против», когда вы выбираете между универсальным или узкоспециализированным решением для баз данных.

1.4.3. Непонимание документации

PostgreSQL обладает очень подробной документацией, что закреплено в управлении проектом. Есть четко установленное правило, что ни один патч не принимается без сопроводительной документации: если вы что-то создаете или изменяете, вы должны это задокументировать. Весь документационный материал доступен по адресу https://postgresql.org/docs/.

Зачастую эта официальная документация представлена в очень техническом или академическом стиле (что соответствует происхождению проекта PostgreSQL). Для неподготовленных пользователей, привыкших к обучающим руководствам или заметкам с более практическими примерами, документация может быть трудной для восприятия. Как следствие, возможно неверно понять суть той или иной функции или способ ее применения.

Также можно упустить важные примечания или не иметь того контекста, который более опытные пользователи приобрели на практике. Иногда части документации кажутся неполными, потому что необходимые для понимания сведения, термины или детали приходится искать в других разделах документации. Наконец, следует убедиться, что вы читаете документацию, специфичную для вашей версии PostgreSQL, чтобы избежать путаницы и недопонимания.

1.4.4. Использование реликтов из стандарта SQL

Тот факт, что что-то присутствует в официальном стандарте SQL, не означает, что вы обязаны это использовать. Многие пережитки сохранились в стандарте потому, что либо относятся к раннему периоду существования стандарта, либо когда-то казались целесообразными, либо были всеми забыты. Часть из них плохо определены с точки зрения реализации и работают по-разному в каждой СУБД. Некоторые функции SQL были сохранены для обратной совместимости, и причины, по которым их не следует предпочитать или использовать вообще, широко понятны. Хорошим примером этого служит тип `TIME WITH TIME ZONE`, действие которого объясняется в главе 3.

1.4.5. Несоблюдение лучших практик

За десятилетия наблюдений и на основе накопленного отраслью практического опыта сформировались лучшие практики проектирования, разработки, администрирования и сопровождения ИТ-систем. Во многих системах есть общие черты, которые предполагают схожие лучшие практики, которые, в свою очередь, имеет смысл применять к базам данных. Базы данных — это сложные системы со множеством аспектов, связанных с точностью, производительностью и безопасностью, и PostgreSQL не исключение из этого правила.

Несоблюдение лучших практик при проектировании схемы базы данных приложения, планировании сценариев использования или уровня параллелизма, внедрении решений по высокой доступности и аварийному восстановлению, а также при формировании политик безопасности может привести к серьезным проблемам. Это верный способ создать себе трудности в будущем.

1.5. Как устроена эта книга

Как я уже упоминал, большинство ошибок и идей в этой книге (если не все) были собраны в ходе наблюдений за тем, как в реальных условиях работают внедренные решения на базе PostgreSQL и как ведут себя их пользователи. Второй их источник — личный опыт автора и истории, которыми поделились другие специалисты отрасли и участники сообщества.

История, или нарратив, — это ключ для понимания обстановки и контекста теоретической или практической проблемы. Она задает основу для образа мышления того, кто пытается ее решить, и проливает свет на его процесс принятия решений. Отсюда мы можем перейти к общему варианту использования и к конкретной проблеме. Мы увидим то, как готовилась попытка решения и какая цепочка событий привела к ошибке. Затем мы глубже погрузимся в то, почему это ошибка, каковы ее причины и потенциальные последствия. Наконец, мы обсудим правильное решение и то, как его реализовать. Все это обсуждение при необходимости дополняется реальными образцами схемы базы данных и самих данных, а также кодом на SQL или PL/pgSQL, как для ошибочных решений, так и для правильного способа действий. Для каждого случая также показан ожидаемый результат.

СОВЕТ PL/pgSQL — это простой процедурный язык для PostgreSQL, который можно использовать для создания функций, процедур и триггеров. Он добавляет управляющие структуры к языку SQL; может применять все пользовательские типы, функции и операторы; способен выполнять сложные вычисления.

На момент написания книги актуальным релизом является PostgreSQL 17, поэтому именно эту версию мы будем использовать в книге. Разумеется, большая часть книги будет применима и к более старым, но все еще актуальным версиям PostgreSQL. В случаях, где есть различия, они будут указаны.

1.5.1. Ментальные модели

Полезно всегда помнить, что PostgreSQL — это клиент-серверная СУБД с многопроцессной архитектурой. Соответственно, любое клиентское приложение подключается к серверу базы данных (серверу PostgreSQL), работающему на одном узле, для выполнения запросов и получения данных. Это подключение обслуживается одним или несколькими фоновыми процессами PostgreSQL (backend-процессами postgres), работающими на стороне сервера. PostgreSQL включает и другие внутренние процессы, которые могут не взаимодействовать напрямую с клиентом. Многопроцессный дизайн позволяет выполнять задачи параллельно и задействовать все доступные CPU, но в некоторых аспектах ведет себя иначе, чем многопоточные системы баз данных. На рис. 1.1 показано, как выглядит эта модель.

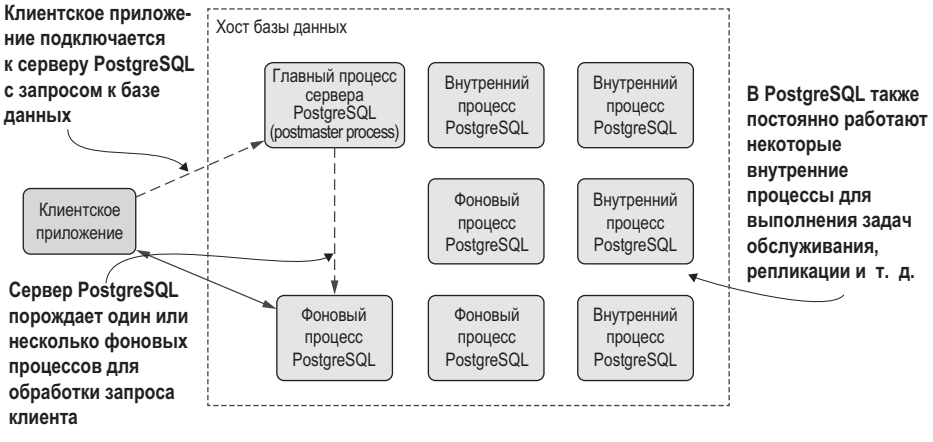

Рис. 1.1. Как работает клиент-серверная архитектура PostgreSQL

Эта книга построена следующим образом. Мы по этапам (по шагам) начинаем с описаний, которые задают контекст для нашего примера использования. Мы изучаем проблему и попытку ее решения, которая приводит к ошибке. Затем мы определяем, почему этот подход неверен, каковы его последствия и возможные обходные пути. Наконец, мы приходим к правильному решению и рассматриваем,

как его реализовать. На рис. 1.2 показан этот процесс, состоящий из этапов разбора проблемы и поиска решений, в тексте они обозначены как «Кейсы».

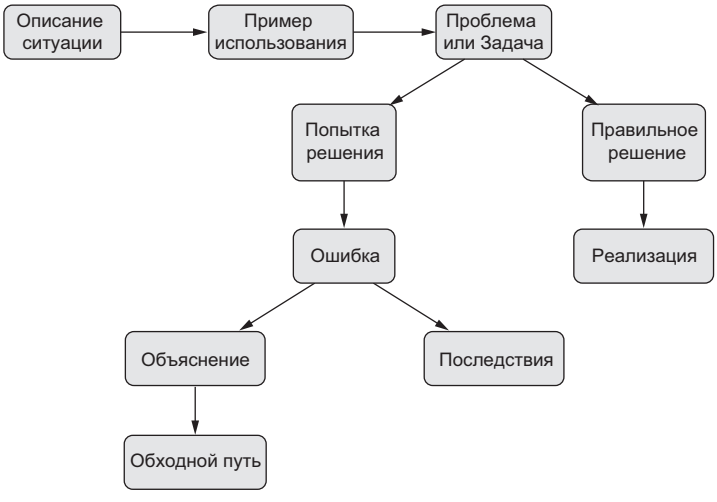

Рис. 1.2. Как устроена эта книга

1.5.2. Пример ошибки

Давайте теперь рассмотрим возможную ошибку, чтобы кратко проиллюстрировать, как эта книга проводит читателя через описание ситуации, проблему и шаги решения. Предположим, у вас есть таблица с сотнями тысяч записей тикетов службы поддержки, например такая (чрезмерно упрощенная) модель:

```
CREATE SCHEMA support
       CREATE TABLE tickets (id int, content text, status smallint);
```

> **ПРИМЕЧАНИЕ** Этот пример не особенно впечатляющий, и он применим не только к PostgreSQL, но из него видно, как эта книга подходит к решению проблем в соответствии с ментальной моделью, представленной на рис. 1.2.

Для нашего примера мы предположим, что `status = 10` означает «открыт» (open), а `status = 20` означает «закрыт» (closed). Давайте вставим несколько сотен тысяч строк с закрытыми тикетами:

```
INSERT INTO support.tickets
SELECT id, 'case description text', 20
FROM generate_series(1, 499750) AS id;
```

А теперь вставим несколько сотен строк недавних, все еще открытых тикетов:

```
INSERT INTO support.tickets
SELECT id, 'case description text', 10
FROM generate_series(499751, 500000) AS id;
```

Ради простоты мы предположим, что параллельное выполнение запросов невозможно, поэтому отключим его с помощью:

```
SET max_parallel_workers_per_gather = 0;
```

Мы будем отслеживать продолжительность выполнения запроса, включив тайминг в `psql`:

```
\timing
Timing is on.
```

Теперь предположим, что для приложения поддержки клиентов актуальны только открытые тикеты. Соответственно, вас интересуют только они, и вы хотите их подсчитать. Попробуем следующий подход:

Кейс 1.1. Попытка решить задачу

```
SELECT count(*)
FROM support.tickets
WHERE status = 10;
```

Действительно, такой подсчет открытых тикетов возвращает правильный результат. Но он выполняется медленно:

```
count
-------
   250
(1 row)

Time: 110.036 ms
```

Посмотрим, почему он медленный. Для этого выполним `EXPLAIN`, который покажет нам, как PostgreSQL будет выполнять запрос (*план запроса*):

```
EXPLAIN SELECT count(*)
FROM support.tickets
WHERE status = 10;
                        QUERY PLAN
-------------------------------------------------------------
Aggregate(cost=9927.28..9927.29 rows=1 width=8)
  ->  Seq Scan on tickets (cost=0.00..9927.28 rows=1 width=0)
        Filter: (status = 10)
(3 rows)
```

Здесь говорится о том, что для выполнения агрегации `count()` PostgreSQL планирует использовать последовательное сканирование (`Seq Scan`) таблицы `tickets` с последующей фильтрацией результатов по условию `status = 10`. Последовательные сканирования (также известные как полные сканирования таблиц) работают медленно. Поэтому вы решаете создать индекс. Ведь индексы ускоряют все, верно?

```
CREATE INDEX ON support.tickets(status);
CREATE INDEX
Time: 732.403 ms
```

Теперь, когда индекс создан, пробуем снова:

```
SELECT count(*)
FROM support.tickets
WHERE status = 10;
 count
-------
   250
(1 row)

Time: 3.715 ms
```

Этот результат значительно лучше. `EXPLAIN` поясняет, почему теперь все работает гораздо быстрее: используется сканирование только по индексу (`Index Only Scan`):

```
                                QUERY PLAN
---------------------------------------------------------------------------
Aggregate (cost=4.44..4.45 rows=1 width=8)
  -> Index Only Scan using tickets_status_idx on tickets
  ➥(cost=0.42..4.44 rows=1 width=0)
        Index Cond: (status = 10)
(3 rows)
```

Однако этот индекс довольно большой относительно полезных данных.

Кейс 1.2. Почему решение не оптимально?

```
\x
\di+ support.tickets*
List of relations
-[ RECORD 1 ]-+-------------------
Schema        | support
Name          | tickets_status_idx
Type          | index
Owner         | frogge
Table         | tickets
Persistence   | permanent
Access method | btree
Size          | 3408 kB
Description   |
```