Карпова Екатерина Олеговна, магистрант
МГТУ им. Н. Э. Баумана

Кострицкий Александр Сергеевич,
старший преподаватель
МГТУ им. Н. Э. Баумана

СУЩЕСТВУЮЩИЕ РЕШЕНИЯ ПО УМЕНЬШЕНИЮ ПОТРЕБЛЕНИЯ ДИСКОВОГО ПРОСТРАНСТВА СИСТЕМОЙ УПРАВЛЕНИЯ БАЗАМИ ДАННЫХ В ПРИМЕНЕНИИ К POSTGRESQL

Аннотация. Описаны основные проблемы, связанные с потреблением дискового пространства системой управления базами данных PostgreSQL и проведен обзор существующих подходов, применяемых в реляционных системах управления базами данных для решения аналогичных проблем.

Ключевые слова: PostgreSQL, дисковое пространство, «мертвые» кортежи, «холодные» данные, сжатие, MVCC.

Введение. PostgreSQL является одной из самых популярных систем управления базами данных. Также, многие разработчики считают ее самой функциональной реляционной СУБД с открытым исходным кодом [1].

Любое приложение, исполняемое на физической или виртуальной машине, потребляет ее ресурсы. Наиболее общими группами ресурсов являются [2, 3] процессорное время, оперативная память, дисковое пространство и сеть. Каждый ресурс необходим для функционирования приложения, но персистентное хранение данных, как основное назначение баз данных, невозможно без достаточного дискового пространства. Администраторы баз данных так или иначе сталкиваются с тем, что таблицы и индексы в PostgreSQL значительно увеличиваются в размерах с течением времени [4], разрастается WAL [5], или объем хранимых «холодных» данных [6]. Чрезмерное потребление диска негативно сказывается на работе приложений и самой СУБД и может привести к остановке работы системы, потере данных, а следовательно повлечь финансовые потери компании-разработчика [7]. Поэтому оптимизации его потребления СУБД PostgreSQL являются одним из приоритетных направлений разработки многих производителей ПО, например: Yandex Cloud [8], PostgresPro [9], OrioleDB [10] и других.

Потребление дискового пространства СУБД PostgreSQL. PostgreSQL является свободно распространяемой реляционной СУБД [11, 12, 13]. PostgreSQL [11, 14, 15] поддерживает создание хранимых процедур, функций, триггеров, имеет расширяемую систему встроенных языков программирования, поддерживает композитные и сложные типы данных, обеспечивает ACID, предоставляет возможность использования как для OLTP, так и для OLAP профиля нагрузки, реализует как пессимистический, так и оптимистический способы управления параллелизмом, поодерживает все существующие платформы (Unix, Linux, Mac OS X, Windows). Последние версии PostgreSQL поддерживаются многими PaaS-провайдерами, например, сервисами SalesForce Heroku PostgreSQL, Engine Yard, Red Hat OpenShift и Amazon RDS для PostgreSQL. Дистрибутивы PostgreSQL входят в реестр ПО РФ [16].

PostgreSQL – одна из наиболее популярных современных СУБД. Она заняла первую строчку рейтинга StackOverflow от 2023 года по используемости: 49% из 60 369 опрошенных профессиональных разработчиков применяют в работе PostgreSQL [17].

На дисках кластера PostgreSQL хранятся [18] временные файлы, файлы данных, журналы транзакций (WAL), слоты логической репликации, файлы логов СУБД и системные файлы, необходимые для функционирования PostgreSQL.

Накопление временных файлов, файлов WAL и слотов логической репликации определяется поведением пользователя: характером запросов к базе данных [18], настройкой репликации [5], то есть не зависит от особенностей реализации PostgreSQL как программного продукта. Системные файлы необходимы PostgreSQL для функционирования и не являются основными потребителями дискового пространства. Разрастание файлов данных нередко связано с реализацией процесса MVCC в исходном коде PostgreSQL [18]. Кроме этого, в PostgreSQL отсутствует постраничное сжатие данных, что также не позволяет экономить на потреблении диска пользовательскими файлами и файлами логов системы управления базами данных [9].

Проблема накопления «мертвых» кортежей. Для обеспечения возможности параллельного выполнения транзакций PostgreSQL использует мультиверсионный подход [19]. Это означает, что при запросе базы данных каждая транзакция видит состояние данных в некоторый момент времени в прошлом, что защищает транзакцию от просмотра несогласованных данных, которые могли появиться из-за параллельных действий других транзакций.

В реализации MVCC PostgreSQL [20] чтение никогда не блокирует запись, а запись никогда не блокирует чтение, единицей мультиверсионности служат строки таблиц, индексные блоки не содержат никакой информации о версионности: записи в индексах ссылаются на каждую из версий строк.

PostgreSQL использует последовательные номера транзакций для определения порядка событий [21]. Табличный блок содержит набор версий строк, для каждой из которых хранятся номера двух транзакций: начальной и конечной. При вставке строки номер транзакции записывается в нее как начальный, при удалении – как конечный. Обновление строки работает как удаление и вставка новой. Таким образом, внутри одного блока могут находиться разные версии одной и той же строки [20, 22].

Благодаря такой реализации, фиксация и отмена транзакций выполняются с одинаковой скоростью, но хранение дополнительных версий строк требует большого объема дискового пространства [22]. Чтобы освободить место, необходимо периодически очищать блоки от тех версий, которые больше не видны ни одной транзакции – «мертвых» кортежей.

Решения проблемы накопления «мертвых» кортежей. В качестве существующих решений здесь и в дальнейшем будут рассматриваться подходы, применяемые в других реляционных СУБД, так как рассматриваемая СУБД PostgreSQL является реляционной, и парадигмы реляционного и нереляционного подходов к хранению данных значительно разнятся.

Без периодической очистки «мертвых» кортежей таблицы и индексы будут разрастаться, а поиск по ним – замедляться. Для того, чтобы минимизировать количество «мертвых» кортежей, в самом PostgreSQL существует специальный регулярный процесс – VACUUM. Он высвобождает пространство, занимаемое «мёртвыми» кортежами [23] и делает его доступным для повторного использования PostgreSQL для этой же таблицы, но не возвращает операционной системе. Этот процесс может работать параллельно с обычными операциями чтения и записи таблицы, так как не требует исключительной блокировки. Другая версия этого процесса, VACUUM FULL, переписывает всё содержимое таблицы в новый файл на диске, что позволяет возвратить неиспользованное пространство операционной системе. Если корректно настроить VACUUM FULL, то он решает описанную проблему, но работает намного медленнее и запрашивает исключительную блокировку для каждой обрабатываемой таблицы, что может замедлять или ограничивать работу приложения, требующего доступа к данным. Кроме того, VACUUM не помогает избавиться от фрагментации, усиливающейся в процессе обновления данных [57], так как не перестраивает таблицы, в отличие от VACUUM FULL.

Реализация MVCC в CockroachDB во многом схожа с представленной в PostgreSQL [24]. CockroachDB так же хранит версии строк, но, так как она является распределенной, то реализует дополнительные алгоритмы с временными метками, позволяющие регулировать

конкурентный доступ. Эти временные метки, присваиваемые версиям строк, так же необходимы для работы сборщика мусора, обеспечивающего функционал схожий с VACUUM-процессом в PostgreSQL [25]. В CockroachDB данные хранятся в формате ключ-значение и разделены на диапазоны по ключам. Версия MVCC ключа считается устаревшей, если ее время жизни превышает установленный GC TTL для диапазона ключей, и есть более новая версия того же ключа. Такие версии строк становятся недоступны для чтения и удаляются сборщиком мусора.

В реляционной СУБД Oracle [20] за упорядоченность отвечает SCN – счетчик, увеличивающийся при каждой фиксации или откате изменений, а не номера транзакций, как в PostgreSQL, единицей многоверсионности служит блок6 любое изменение данных отображается в журнале отката.

Блок всегда содержит актуальный на некоторый момент набор строк. При любом изменении данных в блоке в журнал отката записывается минимальная информация, необходимая для отмены этих изменений. При вставке строки это указание о ее удалении, при изменении – старое значение изменившихся полей, и только при удалении – вся строка полностью. Номер транзакции представляет собой ссылку на цепочку записей в журнале отката. Аналогично с блоками индексов.

Каждый блок, табличный или индексный, содержит ITL – список транзакций, изменяющих этот блок, с указанием их статуса, момента начала и окончания. Эта информация определяет SCN блока – момент, на который данные в блоке актуальны, номером SCN определяется снимок данных.

Чтобы получить в блоке согласованные данные, сначала откатываются изменения незафиксированных транзакций, затем одно за другим откатываются изменения уже зафиксированных транзакций, пока SCN блока не достигнет заданного снимком значения. Фиксация изменений выполняется простой сменой статуса транзакции в журнале отката. Отмена транзакции может занять столько же времени, сколько уже было потрачено на ее выполнение, в отличие от PostgreSQL, где и фиксация, и откат транзакции – одинаковые по трудозатратам операции.

В СУБД MySQL реализован подход с журналом отката, подобный используемому в СУБД Oracle. Периодически запускается purge-процесс, удаляющий записи журнала отката, которые больше не используются транзакциями [26].

В СУБД MSSQL Server реализован подход с хранением исторических версий строк, подобный используемому в СУБД PostgreSQL. Периодически запускается shrink-процесс, удаляющий версии строк, которые больше не используются транзакциями [27]. В момент запуска данного процесса некоторые транзакции могут оказаться блокированными или отмененными, если они требуют чтения удаляемых версий.

Разработчиками Postgres Pro была предпринята попытка реализовать журнал отката наподобие существующего в СУБД Oracle и MySQL [28]. Последние изменения вносились в проект 4 года назад, и он менее популярен среди пользователей, чем иные проекты Postgres Pro. Это может свидетельствовать о том, что данная реализация, по мнению сообщества, не является предпочтительной в рамках использования PostgreSQL.

Оба верхних примера реализуют оптимистичный подход к параллелизму транзакций. Кроме того, существует и пессимистичный вариант, связанный с блокировками. Например, в YugabyteDB реализовано 2 подвида такого подхода [29]. Подход fail-on-clonflict подразумевает, что каждая транзакция наделяется неким приоритетом, и, если две транзакции параллельно работают с одними и теми же данными, то транзакция с низшим приоритетом завершается, и начинает выполняться транзакция, у которой приоритет выше. В подходе wait-on-conflict у транзакций нет приоритетов: новая транзакция ждет завершения транзакций, уже работающих с данными, и начинается только после их фиксации или отмены.

В СУБД SQLite также реализован пессимистичный подход [30].

В целом среди пессимистичных подходов можно выделить несколько групп [31]: стандартный протокол блокировок, протокол блокировок с предварительным запросом (при

котором транзакция сразу запрашивает все необходимые блокировки и начинается только после успешного захвата всех блокировок из списка) и протокол двухфазных блокировок (при котором транзакция выполняется в несколько этапов – сначала она захватывает блокировки постепенно, по одной, затем выполняет необходимые действия с данными, после освобождает блокировки поочередно в обратном порядке).

В PostgreSQL используется протокол двухфазных блокировок [31].

Проблема накопления «холодных» данных. В процессе эксплуатация базы данных объем информации в ней может существенно возрастать. Тогда хранение всех данных базы на диске может стать не лучшим решением из-за возрастающей стоимости на дисковое хранение [6]. Зачастую не весь объем данных находится в активном использовании: рабочие нагрузки OLTP, на который в первую очередь ориентирован PostgreSQL, обычно демонстрируют искаженные шаблоны доступа, когда некоторые записи являются горячими, то есть часто используемыми, а другие – холодными, редко используемыми или никогда не используемыми.

Такие данные могут занимать большую часть дискового пространства, но при этом не быть нужными для работы приложения долгое время.

Решения проблемы накопления «холодных» данных. Более экономично хранить самые холодные записи во вторичном хранилище, таком как флэш-память, либо же в объектных хранилищах [6]. Например, Yandex Cloud предлагает услугу по переносу «холодных» данных в Yandex Object Storage, объектное хранилище. Там они будут храниться в служебном бакете в сжатом и зашифрованном виде. Такое хранение в большинстве облачных провайдеров обходится дешевле [32].

Также, существует расширение pg_tier для PostgreSQL с открытым исходным кодом, предоставляющее решение для управления «холодными» данными [33]. Оно гарантирует, что данные будут храниться экономически эффективно, оставаясь доступными при необходимости. В данной реализации предусмотрено хранение данных в объектном хранилище S3 [34].

Проблема отсутствия постраничного сжатия данных. В базах данных обычно хранятся большие объёмы текста и повторяющейся информации. Поэтому сжатие для большинства баз данных может быть довольно эффективным и позволяет сократить объём хранимых данных в 3–5 раз [35]. Сжатие может быть полезно не только для табличных страниц, но и для страниц индексов по текстовым ключам или для индексов с большим количеством повторяющихся значений.

Для значений отдельных столбцов в PostgreSQL может применяться TOAST-механизм, который подразумевает сжатие и/или разбиение больших значений, то есть занимающих более двух килобайт, на несколько физических строк [36]. Это связано с тем, что PostgreSQL использует фиксированный размер страницы, и не позволяет кортежам занимать несколько страниц. Поэтому непосредственно хранить очень большие значения полей невозможно. Чтобы поддерживать TOAST, тип данных должен представлять значение переменной длины, в котором первое четырехбайтовое слово любого хранящегося значения содержит общую длину значения в байтах. Метод, который будет применяться для сжатия данных при внутреннем и внешнем хранении, можно выбрать для каждого отдельного столбца, задав параметр. Если какие-либо столбцы таблицы хранятся в формате TOAST, у таблицы будет связанная с ней таблица TOAST.

PostgreSQL позволяет включить сжатие WAL [37]. Тогда образ полной страницы, записываемый в WAL, будет сжиматься, и сжатый образ страницы будет распакован при воспроизведении WAL, которое необходимо при восстановлении данных. Это позволяет без дополнительных рисков повреждения данных уменьшить объём WAL, ценой дополнительной нагрузки на процессор, связанной со сжатием данных при записи в WAL и разворачиванием их при воспроизведении WAL.

PostgreSQL поддерживает разные виды индексов, которые отличаются в первую очередь внутренней реализацией. Например, в GIN-индексе записи дедуплицируются: если индексированные ключи для разных строк таблицы идентичны, индекс GIN сохранит это как

одну запись индекса. Указатели на несколько строк таблицы хранятся в posting-списке этой записи [38]. Однако, B-tree индексы хранят записи индекса для каждой указанной строки таблицы. С 13 версии введена дедупликация записей для B-tree индексов. PostgreSQL дедуплицирует записи B-tree только тогда, когда в противном случае ему пришлось бы разделить страницу индекса. Эта дополнительная работа компенсируется уменьшением необходимости разбиения страниц и, как следствие, уменьшением размера индекса. Это экономит дисковое пространство, раздувание индекса уменьшается [39].

В документации PostgreSQL изложены вышеперечисленные идеи, но нет упоминания реализации постраничного сжатия данных, которое могло бы значительно оптимизировать потребление дискового пространства табличными данными. В подкасте про сжатие данных в PostgreSQL, в котором принимали участие Майкл Христофидес, основатель pgMustard [40], и Николай Самохвалов, основатель Postgres.AI [41], также было упомянуто отсутствие подобной оптимизации и высказано предположение о ее положительном влиянии на потребление дискового пространства [42]. Авторы подкаста ссылались на подобное мнение Андрея Бородина, инженера, разрабатывающего открытые СУБД в Яндексе, кандидата технических наук [43].

В своем докладе для CitusCon 2023 [44] Андрей Бородин также упомянул идею сжатия временных файлов, которые могут быть созданы, например, в процессе формирования индекса или операции хеш-джойна, при помощи механизмов RAC, при котором можно получать произвольный доступ к изначальным данным, и SSC, который подразумевает разбиение сжимаемых данных на несколько последовательных сжатых файлов. Он подчеркнул, что спользование RAC оказалось затруднительным в связи с необходимостью дефрагемнтации данных. На данный момент пробный патч, реализующий такое сжатие временных файлов, не реализован.

Решения по постраничному сжатию данных. Для того, чтобы в PostgreSQL работало сжатие всех данных, хранимых на диске, можно использовать файловую систему ZFS на примонтированном диске [45]. ZFS поддерживает автоматическое сжатие данных при помощи разных алгоритмов: LZ4, GZIP, ZSTD и других. В примере из статьи на сайте EnterpriseDB вышло достичь коэффициента сжатия равного 7 для данных PostgreSQL в результате определенных настроек размера блока файловой системы [46].

Postgres Pro разработали собственную файловую систему CFS, позволяющую реализовать сжатие на уровне страниц [47]. Postgres Pro Enterprise хранит отношение в наборе файлов, ограничивая размер каждого 2 гигабайтами. CFS создаёт для каждого файла отдельную карту отображения страниц и производит сборку мусора в нескольких фоновых процессах. Обрабатывается каждый файл независимо. Эти файлы могут блокироваться в момент сборки мусора, но всё отношение при этом не блокируется. Сжатие можно включить для отдельных табличных пространств. При этом системные отношения не сжимаются в любом случае.

CFS сохраняет логический размер блока в памяти, но на диск пишет их сжатыми непосредственно друг за другом без выравнивания. При этом, если блок обновляется, то он не переписывается по месту, а дописывается в конец файла. Если в файле было обновлено много блоков, в нём появляется много неиспользуемого места. Чтобы вернуть неиспользуемое место файловой системе, используется процесс компактификации: «живые» блоки копируются в новый файл, файлы данных подменяются, файл отображения, содержащий указатели на актуальные позиции блоков, корректируется.

В СУБД MySQL InnoDB реализовано постраничное сжатие данных [48]. Все пользовательские данные в таблицах InnoDB и индексах хранятся в формате B-tree индексов, для которых реализовано сжатие по страницам. При записи страницы она сжимается с использованием указанного в конфигурации алгоритма сжатия. Сжатые данные записываются на диск, где встроенный механизм освобождает пустые блоки с конца страницы. Если сжатие не удается, данные записываются без сжатия.

В СУБД CockroachDB используется движок Pebble для хранения данных в формате ключ-значение [49]. В нем реализовано сжатие ключей по префиксу и от RocksDB, на которой он основан, унаследовано постраничное сжатие данных [50].

В СУБД YugabyteDB также используется RocksDB, поддерживающая постраничное сжатие данных [51]. СУБД MSSQL Server, Oracle тоже поддерживают постраничное сжатие данных [52, 53].

Сравнение существующих решений. В рамках работы было рассмотрено несколько проблем, влияющих на потребление дискового пространства СУБД PostgreSQL.

В таблице 1 представлено сравнение проблем потребления дискового пространства СУБД PostgreSQL в соответствии с выбранными критериями.

Таблица 1

Сравнение проблем, связанных с потреблением дискового пространства

Проблема	Критерий сравнения		
	Независимость от специфики данных	Влияние на производительность	Распространенность проблемы
Проблема накопления «мертвых» кортежей	Да	Да	Да
Проблема накопления «холодных» данных	Нет	Нет	Да
Проблема отсутствия постраничного сжатия данных	Да	Нет	Нет

Оценка распространенности основывалась на следующих фактах. В большинстве рассмотренных в рамках работы источников из перечисленных проблем упоминается только проблема накопления мёртвых кортежей, в частности, это касается литературных изданий про PostgreSQL, например, «Конфигурирование PostgreSQL» Б. Шейка [22], «Освоение PostgreSQL 13» Х. Шенига [54], «Поваренная книга по администрированию PostgreSQL 9.5/9» С. Риггса, Г. Чиолли, Г. Бартолини [55] и других. Крупные облачные провайдеры, например, Yandex Cloud, разработали собственный инструмент, поддерживающий миграции данных в «холодные» хранилища [32]. В Postgres Pro предпринималась попытка имплементировать замену существующего алгоритма MVCC, чтобы избежать накопления «мертвых» кортежей [28]. В подкасте Майкла Христофидеса и Николая Самохвалова было упомянуто, что проблеме отсутствия постраничного сжатия в сообществе PostgreSQL уделяется мало внимания [42].

По результатам сравнения наиболее актуальной является проблема накопления «мертвых» кортежей.

В таблице 2 представлено сравнение существующих решений проблемы накопления «мертвых кортежей» в соответствии с выбранными критериями.

Таблица 2

Сравнение существующих методов решения проблемы накопления «мертвых» кортежей

Метод решения	Критерий сравнения		
	Влияние на производительность	Доступность для PostgreSQL	Блокировка таблиц
VACUUM FULL	Создает нагрузку на процессор, требуется дополнительное место на диске	Да	Да
Сборщик мусора для мертвых кортежей	Создает нагрузку на процессор	Нет	Нет

Журнал отката	Откат транзакции медленнее, чем при стандартном MVCC	Нет	Нет
Блокировки	Ограничивают пропускную способность	Да	Да

Реализованный в PostgreSQL механизм VACUUM FULL требует значительное количество дополнительных ресурсов системы и может влиять на доступность данных для приложения, использующего базу данных, накладывая ограничения на степень оптимизации его работы. Реализованный в CockroachDB сборщик мусора оптимизирован под работу с key-value хранением данных, что не подходит для PostgreSQL. Он не следит за востребованностью данных транзакциями, ориентируясь исключительно на временные отметки, что требует отдельной настройки администратором порядка очистки данных при помощи механизма PTS [56]. Реализация журнала отката транзакций подразумевает замедление процесса отката транзакций, так как он требует воспроизведения набора действий в определенном порядке и дополнительного кеширования восстановленных версий для точечной оптимизации. Блокировки изначально подразумевают снижение пропускной способности приложения, так как не позволяют транзакциям выполняться параллельно.

Таким образом, ни одно из рассмотренных решений не соответствует требованиям сообщества в полной мере.

В таблице 3 представлено сравнение существующих решений проблемы накопления «холодных» данных в соответствии с выбранными критериями.

Таблица 3

Сравнение существующих методов решения проблемы накопления «холодных» данных

Метод решения	Критерий сравнения			
	Проприетарность	Недступность для польз. кластера PostgreSQL	Блокировка таблиц	Автоматиз. на уровне СУБД
Yandex Object Storage + Yandex Data Transfer	Да	Да	Нет	Нет
pg_tier	Нет	Нет	Нет	Нет

Существующее решение с открытым исходным кодом решает обозначенную проблему, но его использование требует контроля и не подразумевает автоматическое обнаружение «холодных» данных. Таким образом, процесс переноса становится трудоемким, требующим периодического ручного выполнения, что ведет к риску повторного накопления «холодных» данных и ошибок, связанных с человеческим фактором.

В таблице 4 представлено сравнение существующих решений проблемы отсутствия постраничного сжатия в соответствии с выбранными критериями.

Таблица 4

Сравнение существующих методов решения проблемы отсутствия постраничного сжатия

Метод решения	Критерий сравнения		
	Проприетарность	Неприменимость в парадигме PostgreSQL	Реализация на уровне СУБД
ZFS	Нет	Нет	Нет
CFS	Да	Нет	Нет
Индексное хранение	Нет	Да	Да
Сжатие при модели хранения ключ-значение	Нет	Да	Да

Индексное хранение и сжатие при модели хранения ключ-значение требуют внесения значительных изменений в исходный код системы хранения данных PostgreSQL, поэтому не являются предпочтительными. Решения, связанные с использованием файловых систем, подходят для большинства случаев возникновения проблемы и не требуют внесения изменений в хранимые данные. Однако, потребность в автоматизированном решении на уровне СУБД сохраняется, о чем свидетельствуют мнения Майкла Христофидеса и Николая Самохвалова, разработчиков, тесно взаимодействующих с PostgreSQL в рамках профессиональной деятельности [41].

Заключение. Описаны основные проблемы, связанные с потреблением дискового пространства системой управления базами данных PostgreSQL: накопление «мертвых» кортежей, накопление «холодных» данных, отсутствие постраничного сжатия данных, наиболее актуальной среди которых является накопление «мертвых» кортежей. Ни один из приведенных подходов не удовлетворяет существующим требованиям пользователей в полной мере, и потребность в альтернативном способе уменьшения потребления дискового пространства PostgreSQL сохраняется. В соответствии с мнениями из упомянутых источников, решение должно минимизировать дополнительную нагрузку на процессор, учитывать востребованность данных, не блокировать параллельные транзакции и позволять за приемлемое время, то есть схожее со временем, затрачиваемым на подобные действия в текущей реализации PostgreSQL, восстанавливать исторические данные. Расширение pg_tier для PostgreSQL с открытым исходным кодом решает обозначенную проблему накопления «холодных» данных, но не подразумевает автоматическое обнаружение таких данных, что влечет за собой риск их повторного накопления и ошибки, связанные с человеческим фактором. Существующие подходы к реализации постраничного сжатия, связанные с использованием файловых систем с настраиваемым автоматическим сжатием, подходят для многих случаев и не требуют внесения изменений в хранимые данные и исходный код PostgreSQL. Однако, потребность в автоматизированном решении на уровне СУБД сохраняется.

Список литературы:

1. Worsley, John and Drake, Joshua D. Practical PostgreSQL. O'Reilly Media, Inc., 2002.
2. Matsunaga, Andrea and Fortes, Jose AB. On the use of machine learning to predict the time and resources consumed by applications, 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing, 2010, с. 495–504.
3. Sun, Xiang and Ansari, Nirwan and Wang, Ruopeng. Optimizing resource utilization of a data center. IEEE Communications Surveys & Tutorials., т. 18, № 4, с. 2822–2846.
4. Официальный сайт PGConf.Russia 2023 [Электронный ресурс]. Режим доступа: https://pgconf.ru/talk/1589479 (дата обращения: 25.11.2024).
5. Официальный сайт Yandex Cloud: устранение последствий переполнения хранилища кластера WAL-журналами [Электронный ресурс]. Режим доступа: https://yandex.cloud/ru/docs/troubleshooting/managed-postgresql/known-issues/cluster-in-readonly-due-to-wal-overflow (дата обращения: 26.11.2024).
6. Levandoski, Justin J and Larson, Perke and Stoica, Radu. Identifying hot and cold data in main-memory databases. 2013 IEEE 29th International Conference on Data Engineering (ICDE), 2013, с. 26–37.
7. Eun, Park Ju and Jang, Minyoung. A Study on Optimizing Disk Utilization of Software-Defined Storage. KIPS Transactions on Computer and Communication Systems, 2023, т. 12, № 4, с. 135–142.
8. Официальный сайт Yandex Cloud: миграция базы данных из Managed Service for PostgreSQL в Object Storage [Электронный ресурс]. Режим доступа: https://yandex.cloud/ru/docs/tutorials/dataplatform/mpg-to-objstorage (дата обращения: 25.11.2024).

9. Доклады PGConf.Russia 2022 [Электронный ресурс]. Режим доступа: https://pgconf.ru/talk/1589011 (дата обращения: 25.11.2024).
10. Официальный сайт OrioleDB [Электронный ресурс]. Режим доступа: https://www.orioledb.com (дата обращения: 25.11.2024).
11. Доклады PGCONF.RUSSIA 2023 [Электронный ресурс]. Режим доступа: https://pgconf.ru/talk/1589083 (дата обращения: 26.11.2024)
12. Проскурин А. И., Ефремов М. А. Современные СУБД. Современные информационные технологии и информационная безопасность, 2023, с. 88–91.
13. Чередникова О. Ю., Щедрин С. В., Исаков А. Ю., Ногтев Е. А. Исследование технологии шардинга для повышения эффективности обработки данных программного комплекса приемной комиссии ДонНТУ. Информатика и кибернетика, Государственное образовательное учреждение высшего профессионального…, 2020, № 4, с. 40–46.
14. Obe, Regina O and Hsu, Leo S. PostgreSQL: up and running: a practical guide to the advanced open source database. O'Reilly Media, Inc., 2017.
15. Juba, Salahaldin and Vannahme, Achim and Volkov, Andrey. Learning PostgreSQL. Packt Publishing Ltd., 2015.
16. Реестр ПО РФ [Электронный ресурс]. Режим доступа: https://reestr.digital.gov.ru/search/?q=Postgres (дата обращения: 27.11.2024).
17. Результаты опроса разработчиков, проведённого в 2023 году StackOverflow [Электронный ресурс]. Режим доступа: https://survey.stackoverflow.co/2023/#section-most-popular-technologies-databases (дата обращения: 27.11.2024).
18. Официальный сайт Selectel [Электронный ресурс]. Режим доступа: https://docs.selectel.ru/cloud/managed-databases/postgresql/use-disk-space (дата обращения: 25.11.2024).
19. Официальный сайт PostgreSQL [Электронный ресурс]. Режим доступа: https://www.postgresql.org/docs/current/mvcc-intro.html (дата обращения: 25.11.2024).
20. Официальный сайт Postgres Pro: MVCC [Электронный ресурс]. Режим доступа: https://postgrespro.ru/blog/pgsql/17758 (дата обращения: 25.11.2024).
21. Momjian, Bruce. MVCC Unmasked 2010 [Электронный ресурс]. Режим доступа: https://momjian.us/main/writings/pgsql/mvcc.pdf (дата обращения: 25.11.2024).
22. Shaik, Baji. PostgreSQL Configuration. Best Practices for Performance and Security. Apress, Berkeley, CA, Springer, 2020.
23. Официальный сайт Postgres Pro: VACUUM [Электронный ресурс]. Режим доступа: https://postgrespro.ru/docs/postgrespro/9.5/sql-vacuum (дата обращения: 25.11.2024).
24. Официальный сайт CockroachDB: transactions [Электронный ресурс]. Режим доступа: https://www.cockroachlabs.com/docs/stable/architecture/transaction-layer (дата обращения: 25.11.2024).
25. Официальный сайт CockroachDB: mvcc and garbage collection [Электронный ресурс]. Режим доступа: https://www.cockroachlabs.com/docs/v24.3/architecture/storage-layer.html (дата обращения: 25.11.2024).
26. Официальный сайт MySQL: Purge configuration [Электронный ресурс]. Режим доступа: https://dev.mysql.com/doc/refman/8.4/en/innodb-purge-configuration.html (дата обращения: 25.11.2024).
27. Официальный сайт MSSQL Server: Transaction locking [Электронный ресурс]. Режим доступа: https://learn.microsoft.com/en-us/sql/relational-databases/sql-server-transaction-locking-and-row-versioning-guide?view=sql-server-ver16#managing-concurrent-data-access (дата обращения: 25.11.2024).
28. Официальный репозиторий Postgres Pro: undam [Электронный ресурс]. Режим доступа: https://github.com/postgrespro/undam (дата обращения: 25.11.2024).
29. Официальный сайт YugabyteDB [Электронный ресурс]. Режим доступа: https://docs.yugabyte.com/preview/architecture/transactions/concurrency-control (дата обращения: 25.11.2024).

30. Официальный сайт SQLite: File Locking And Concurrency In SQLite Version 3 [Электронный ресурс]. Режим доступа: https://www.sqlite.org/lockingv3.html (дата обращения: 25.11.2024).

31. Stonebraker, Michael. The design of the Postgres storage system. University of California, Berkeley, College of Engineering, Electronics…, 1987.

32. Официальный сайт Yandex Cloud: миграция данных в холодное хранилище [Электронный ресурс]. Режим доступа: https://yandex.cloud/ru/docs/tutorials/dataplatform/greenplum-yezzey (дата обращения: 25.11.2024).

33. Официальный репозиторий pg_tier [Электронный ресурс]. Режим доступа: https://github.com/tembo-io/pg_tier (дата обращения: 25.11.2024).

34. Официальный сайт Amazon S3 [Электронный ресурс]. Режим доступа: https://aws.amazon.com/ru/s3 (дата обращения: 25.11.2024).

35. Официальный сайт Postgres Pro: CFS compression [Электронный ресурс]. Режим доступа: https://postgrespro.ru/docs/enterprise/9.6/cfs-overview (дата обращения: 25.11.2024).

36. Официальный сайт Postgres: TOAST [Электронный ресурс]. Режим доступа: https://www.postgresql.org/docs/current/storage-toast.html (дата обращения: 25.11.2024).

37. Официальный сайт Postgres: WAL [Электронный ресурс]. Режим доступа: https://www.postgresql.org/docs/current/runtime-config-wal.html (дата обращения: 25.11.2024).

38. Официальный сайт Postgres: GIN-индексы [Электронный ресурс]. Режим доступа: https://www.postgresql.org/docs/current/gin.html (дата обращения: 25.11.2024).

39. Официальный сайт CYBERTEC: index deduplication [Электронный ресурс]. Режим доступа: https://www.cybertec-postgresql.com/en/b-tree-index-deduplication/ (дата обращения: 25.11.2024).

40. Официальный сайт pgMustard [Электронный ресурс]. Режим доступа: https://www.pgmustard.com (дата обращения: 25.11.2024).

41. Официальный сайт Postgres.AI [Электронный ресурс]. Режим доступа: https://postgres.ai (дата обращения: 25.11.2024).

42. Официальный сайт postgres.fm: сжатие данных [Электронный ресурс]. Режим доступа: https://postgres.fm/episodes/compression (дата обращения: 25.11.2024).

43. Официальный сайт Highload++: Андрей Бородин [Электронный ресурс]. Режим доступа: https://highload.ru/spring/2021/authors/2147 (дата обращения: 25.11.2024).

44. Официальный сайт CitusCon 2023: Андрей Бородин [Электронный ресурс]. Режим доступа: https://learn.microsoft.com/en-us/shows/cituscon-an-event-for-postgres-2023/on-compression-of-everything-in-postgres-citus-con-2023 (дата обращения: 25.11.2024).

45. Официальный сайт OpenZFS [Электронный ресурс] – Режим доступа: https://openzfs.github.io/openzfs-docs/Performance%20and%20Tuning/Workload%20Tuning.html\#compression (дата обращения: 25.11.2024).

46. Официальный сайт EnterpriseDB: ZFS compression [Электронный ресурс]. Режим доступа: https://www.enterprisedb.com/blog/pg-phriday-postgres-zfs (дата обращения: 25.11.2024).

47. Официальный сайт Postgres Pro Enterprise: CFS [Электронный ресурс]. Режим доступа: https://postgrespro.ru/docs/enterprise/9.6/cfs (дата обращения: 25.11.2024).

48. Официальный сайт MySQL: compression [Электронный ресурс]. Режим доступа: https://dev.mysql.com/doc/refman/8.4/en/innodb-compression-internals.html (дата обращения: 25.11.2024).

49. Официальный сайт CockroachDB: Storage Layer [Электронный ресурс]. Режим доступа: https://www.cockroachlabs.com/docs/stable/architecture/storage-layer (дата обращения: 25.11.2024).

50. Официальный репозиторий RocksDB [Электронный ресурс]. Режим доступа: https://github.com/facebook/rocksdb/wiki/Rocksdb-BlockBasedTable-Format (дата обращения: 25.11.2024).

51. Презентация с официального доклада YugabyteDB [Электронный ресурс]. Режим доступа: https://info.yugabyte.com/hubfs/YFTT%20Slide%20Decks/2023_01_13_YFTT_Raghavendra_TK_Network%20and%20On-Disk%20Compression%20in%20YugabyteDB.pdf (дата обращения: 25.11.2024).

52. Официальный сайт MSSQL Server: Data compression [Электронный ресурс]. Режим доступа: https://learn.microsoft.com/en-us/sql/relational-databases/data-compression/data-compression?view=sql-server-ver16 (дата обращения: 25.11.2024).

53. Официальный сайт Oracle: Database Concepts [Электронный ресурс]. Режим доступа: https://docs.oracle.com/en/database/oracle/oracle-database/19/cncpt/tables-and-table-clusters.html (дата обращения: 25.11.2024).

54. Schonig, Hans-Jurgen. Mastering PostgreSQL 13: Build, administer, and maintain database applications efficiently with PostgreSQL 13. Packt Publishing Ltd., 2020.

55. Riggs, Simon and Ciolli, Gianni and Bartolini, Gabriele. PostgreSQL Administration Cookbook, 9.5/9. Packt Publishing Ltd., 2017.

56. Официальный сайт CockroachDB: PTS [Электронный ресурс]. Режим доступа: https://www.cockroachlabs.com/blog/protected-timestamps-for-less-garbage/ (дата обращения: 25.11.2024).

57. Супрунов С. Полезные советы по PostgreSQL. Системный администратор №. 11, 2006.