**Министерство науки и высшего образования**

**Российской Федерации**

**ФЕДЕРАЛЬНОЕ Государственное Автономное**

**образовательное учреждение высшего образования**

**«новосибирский национальный исследовательский государственный университет»**

**Факультет информационных технологий**

Кафедра компьютерных систем

Направление подготовки 09.03.01 Информатика и вычислительная техника

Направленность (профиль) Программная инженерия и компьютерные науки

**ОТЧЕТ**

**о прохождении учебной практики, эксплуатационной практики**

(указывается наименование практики)

**Обучающегося Винтер Алёны Викторовны группы №21201 курса 3**

(Ф.И.О. полностью)

**Тема задания**: **Исследование работы журнала предзаписи и поиск альтернативных методов резервного копирования данных в СУБД PostgreSQL.**

**Место прохождения практики:** Общество с ограниченнной ответственностью «Постгрес Профессиональный» (ООО «ППГ»), 630090, Новосибирская область, г. Новосибирск, пр. Лаврентьева 2/2, офис 419

(полное наименование организации и структурного подразделения, индекс, адрес)

**Сроки прохождения практики:** с 05.02.2024 г. по 24.05.2024 г.

**Руководитель практики   
от профильной организации** Рутман Михаил Валерьевич, \_\_\_\_\_\_\_\_\_\_\_\_\_\_

ведущий разработчик (подпись)

программного обеспечения

Департамента разработки

продуктов (Обособленное

подразделение г. Новосибирск)

(Ф.И.О. полностью, должность)

**Руководитель практики от НГУ** Пищик Борис Николаевич, \_\_\_\_\_\_\_\_\_\_\_\_\_\_

КафКС ФИТ НГУ, доцент (подпись)

(Ф.И.О. полностью, должность)

**Руководитель ВКР** Пищик Борис Николаевич КафКС ФИТ НГУ, доцент

(Ф.И.О. полностью) (должность)

**Оценка по итогам защиты отчета:**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(неудовлетворительно, удовлетворительно, хорошо, отлично)

**Отчет заслушан на заседании кафедры** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(наименование кафедры)

**протокол \_\_\_\_\_\_\_\_\_от** «\_\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_\_\_\_г.

Новосибирск 2024

**Содержание**

Введение..............................................................................................................................................3

1.Журнал предзаписи.........................................................................................................................4

2.Архитектура журнала предзаписи в PostgreSQL.........................................................................7

3.Преимущества и недостатки использования журнала предзаписи..........................................10

4.Восстановление базы данных после сбоя с помощью журнала предзаписи в PostgreSQL....11

5.Альтернативные методы резервного копирования в PostgreSQL.............................................12

Заключение........................................................................................................................................13

Список литературы...........................................................................................................................14

Приложение А...................................................................................................................................15

Приложение Б...................................................................................................................................16

Приложение В...................................................................................................................................17

Приложение Г...................................................................................................................................18

Приложение Д...................................................................................................................................19

**Введение**

В современной информационной среде надежность и целостность данных являются ключевыми аспектами для успешного функционирования баз данных. PostgreSQL, как одна из самых популярных систем управления базами данных, предоставляет различные механизмы для обеспечения этих требований. Одним из таких механизмов является журнал предзаписи (Write-Ahead Logging, WAL). Журнал предзаписи в PostgreSQL играет критически важную роль в обеспечении надежности данных, позволяя системе восстанавливать состояние базы данных после сбоев и обеспечивать атомарность транзакций. Этот механизм реализует концепцию, согласно которой все изменения, производимые в базе данных, сначала записываются в журнал, прежде чем они применяются к основной базе данных. Такая стратегия позволяет минимизировать риск потери данных и обеспечивает возможность отката к консистентному состоянию в случае возникновения критических ошибок или сбоев.

**Цель:** Исследование работы журнала предзаписи и поиск альтернативных методов резервного копирования данных в СУБД PostgreSQL.

**Задачи:**

1.Изучить работу журнала предзаписи;

2.Рассмотреть архитектуру журнала предзаписи в PostgreSQL;

3.Обзор преимуществ и недостатков использования журнала предзаписи;

4.Описание методов восстановления данных с использованием журнала предзаписи;

5.Анализ производительности и надежности системы при использовании журнала предзаписи;

6.Поиск различных методов резервного копирования в PostgreSQL;

7.Оценка преимуществ и недостатков альтернативных методов.

Реализация механизма журнала предзаписи рассмотрена для PostgreSQL версии 16.

Актуальность темы заключается в необходимости обеспечения высокой надежности и производительности СУБД PostgreSQL в условиях растущих объемов данных и требований к их сохранности, что требует глубокого понимания работы журнала предзаписи (WAL) и поиска альтернативных методов резервного копирования.

Ожидаемые результаты включают подробное понимание работы и оптимизации журнала предзаписи (WAL) в PostgreSQL, сравнительный анализ альтернативных методов резервного копирования, а также рекомендации по их выбору и настройке для повышения надежности и производительности системы.

Практика проводилась в обществе с ограниченной ответственностью «Постгрес Профессиональный» (ООО «ППГ»), 630090, Новосибирская область, г. Новосибирск, пр. Лаврентьева 2/2, офис 419.

Postgres Professional — российский разработчик системы управления базами данных Postgres Pro на основе PostgreSQL. СУБД Postgres Pro входит в Единый реестр российского программного обеспечения. Компания дополняет Postgres новой функциональностью (включая разработку ядра), обеспечивает безопасность.

**1.Журнал предзаписи**

Журнал предзаписи (Write-Ahead Logging, WAL) — это метод обеспечения надежности и целостности данных в системах управления базами данных, при котором все изменения сначала записываются в журнал (специальный лог-файл) перед их применением к основной базе данных. Этот механизм позволяет восстановить базу данных до консистентного состояния в случае сбоя, так как все необходимые операции для восстановления хранятся в журнале.

Запись об изменении страницы в обязательном порядке попадает на диск перед тем, как туда попадет измененная страница. Это требование дает возможность в случае сбоя прочитать с диска журнал и повторить те операции, которые были выполнены до сбоя, но результат которых не успел дойти до диска из оперативной памяти и пропал. Журналировать нужно все операции, при выполнении которых есть риск получить несогласованность на диске в случае сбоя.

Журнал представляет собой строго последовательный поток записей, т. е. новые записи всегда добавляются в конец журнала. Каждая запись снабжается уникальным идентификатором LSN (log sequence number), который однозначно ее идентифицирует. Важным требованием к LSN, которое необходимо для корректной работы алгоритма восстановления и поэтому соблюдается во всех системах, является строгая упорядоченность: записи журнала, созданные позже, должны иметь большие значения LSN, чтобы, сравнивая номера LSN двух записей, всегда можно было установить, какая из них появилась раньше.

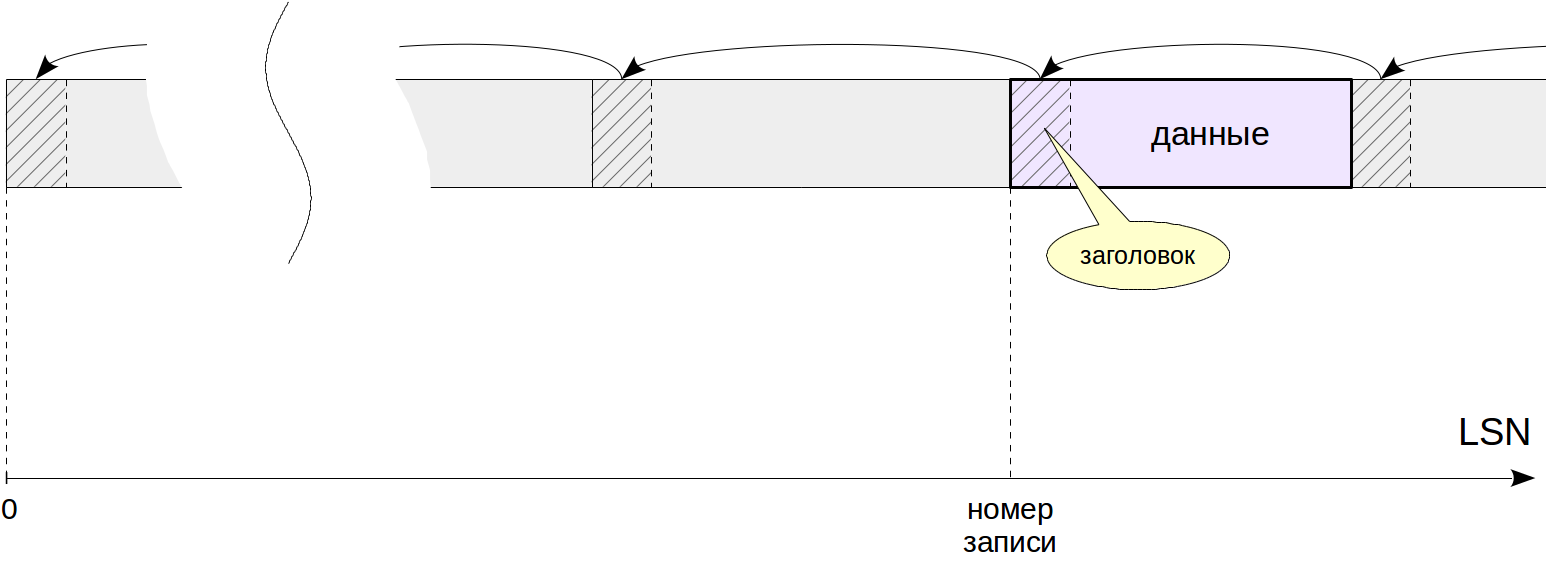


Рис. 1

Ведение журнала предзаписи подчиняется следующим правилам:

1. записи, регистрирующие любые изменения в базе данных, должны быть занесены в журнал и вытолкнуты на диск, до того как сами эти изменения (в базе данных) попадут на диск;
2. информация о фиксации транзакций в журнале должна быть вытолкнута на диск раньше, чем завершится операция фиксации, и раньше, чем приложение получит информацию о том, что фиксация выполнена успешно.

Существует несколько стратегий записи в журнал, гарантирующих возможность восстановления согласованного состояния базы данных. Эти стратегии обеспечивают выполнение требований атомарности и долговечности транзакций: завершенные транзакции не могут быть потеряны, а оборванные (или не завершенные до отказа) транзакции не должны оставлять изменений в базе данных. Для выполнения этих требований, необходимо чтобы информация об изменениях, которые еще не записаны на диск базы данных, обязательно сохранялась в журнале и при этом попадала на энергонезависимый носитель данных.

Выбор стратегии ведения журнала определяется следующими свойствами:

**FORCE / NO FORCE.**

Свойство FORCE означает, что все изменения, выполненные транзакцией, заносятся в базу данных и выталкиваются на диск, до того как завершается выполнение операции COMMIT. В этом случае нет необходимости в записях REDO[[1]](#footnote-1).

Свойство NO FORCE, наоборот, предполагает, что записи REDO выталкиваются из журнала на диск до завершения фиксации, но не требуют записи изменений в базу данных.

**STEAL / NO STEAL.**

Свойство STEAL означает, что страницы (блоки) базы данных, содержащие изменения незафиксированных транзакций, могут выталкиваться на диск. В этом случае необходимы журнальные записи UNDO[[2]](#footnote-2), которые должны выталкиваться на диск раньше, чем страницы базы данных, содержащие эти изменения.

Свойство NO STEAL запрещает выталкивание на диск страниц, содержащих изменения незавершенных транзакций. В этому случае записи UNDO не требуются, потому что на диск попадают только изменения зафиксированных транзакций.

Комбинация этих свойств дает четыре возможных стратегии ведения журнала.

**FORCE + NO STEAL.**

Эта стратегия требует, чтобы все изменения записывались в базу данных в момент фиксации. Для ее реализации используется метод теневых страниц, состоящий в том, что измененные транзакцией страницы записываются на новое место, а при фиксации происходит обновление только одной страницы, содержащей указатели на актуальные версии данных. Для восстановления не требуются записи ни REDO, ни UNDO, поэтому при восстановлении после системных отказов журнал не нужен. Эта стратегия оказывается менее эффективной при нормальной работе (т. е. создает большую дополнительную нагрузку, чем остальные) и в высокопроизводительных системах не применяется.

**FORCE + STEAL.**

Поскольку все изменения должны быть записаны на диск базы данных до фиксации, могут возникать задержки при фиксации транзакций, выполнивших относительно большое количество изменений. Для этой стратегии не требуются записи REDO.

**NO FORCE + NO STEAL.**

Изменения записываются на диск базы данных только после фиксации транзакций. При этом некоторые страницы могут слишком долго оставаться в оперативной памяти (например, страницы, часто обновляемые различными транзакциями). В этом случае записи UNDO не требуются.

**NO FORCE + STEAL.**

Эта стратегия дает возможности для полностью асинхронной записи изменений базы данных на диск, никак не связанной с фиксацией транзакций. Обычно именно эта стратегия используется высокопроизводительными системами, в том числе PostgreSQL.

**2.Архитектура журнала предзаписи в PostgreSQL**

В PostgreSQL для всего кластера баз данных используется один общий журнал предзаписи, состоящий из последовательности файлов, называемых WAL-сегментами. Каждый сегмент имеет фиксированный размер, задающимся параметром *wal\_segment\_size* (по умолчанию, 16 МБ). Эти файлы хранятся в каталоге *PGDATA/pg\_wal*. Сегменты содержат строго последовательные записи различной длины. Местонахождение записи в WAL — это совокупность имени сегмента и смещения от начала файла.

Каждая запись содержит стандартный заголовок и данные об операции. Сами данные имеют разный формат и смысл. Для интерпретации и воспроизведения каждой записи используется соответствующий *менеджер ресурсов*.

В приложении А приведён код заголовка журнальной записи. XLogRecord содержит основные характеристики записи, такие как её длина, номер менеджера ресурсов, флаги, идентификатор транзакции, контрольную сумму и указатель на предыдущую запись в журнале. Эти данные являются ключевыми для чтения и понимания содержания записей.

После заголовка может быть 0 или более заголовков блоков данных — XlogRecordBlockHeader. Код представлен в приложении Б. Заголовки блоков содержат некоторую информацию о данных в блоках: номер блока, его длина, флаги.

Если в заголовке блока флаг содержит значение BKPBLOCK\_HAS\_IMAGE, то после следует XlogRecordBlockImageHeader. Код приведён в приложении В. Дополнительный информационный заголовок, для полностраничного изображения. Такие записи хранят не только лишь измененную строку, но всю страницу целиком.

Если в заголовке изображения флаг содержит значения BKPIMAGE\_HAS\_HOLE и BKPIMAGE\_COMPRESSED, то после следует XlogRecordBlockCompressHeader. Код в приложении Г. Информация, используемая для сжатого полностраничного изображения.

Если в заголовке блока флаг не содержит значение BKPBLOCK\_SAME\_REL, то после следует RelFileLocator. Код приведён в приложении Д. Эта структура хранит всю необходимую информацию о физическом расположении таблицы/строки: идетификатор табличного пространтсва, идентификатор базы данных и номер отношения.

После каждого заголовка блока следует BlockNumber. Каждый файл данных (куча или индекс) в PostgreSQL поделены на блоки диска (которые можно рассматривать какединичу ввода-вывода), которые пронумерованные последовательно.

Далее идут блоки данных в соответствие с каждым XlogRecordBlockHeader. Вид блоков определяет тип журнальной записи и её resource manager.

Завершает запись часть данных под названием *main data*, содержащая основополагающие данные для каждого формата журнальных записей.

Для повышения производительности WAL-записи сначала помещаются в специальные WAL-буферы, выделенные в общей памяти сервера, работающие преимущественно в режиме кольцевого буфера: записи добавляются в «голову», а сохраняются на диск с «хвоста». Позиция уже сохраненных записей («хвост» буфера) и позиция вставки («голова» буфера) в ненагруженной системе почти всегда будут совпадать.

Чтобы сослаться на определенную запись, используется тип данных *pg\_lsn* (log sequence number, LSN), представляющий 64-битное смещение в байтах до журнальной записи относительно начала журнала. LSN выводится как два 32-битных числа в шестнадцатеричной системе через косую черту.

Когда буфер заполняется или когда происходит транзакция COMMIT, содержимое буфера записывается на диск.

Для восстановления согласованности после сбоя необходимо последовательно читать журнал и применять к страницам те записи, которые относятся к пропавшим изменениям. Чтобы их обнаружить, надо сравнить LSN страницы на диске с LSN журнальной записи. Однако остается непонятным, с какой именно записи следует начинать восстановление. Если начать слишком поздно, то к страницам, записанным на диск раньше этого момента, будут применены не все изменения, а это приведет к необратимому повреждению данных. Начинать с самого начала невозможно из-за огромного объема данных, который невозможно хранить, и из-за длительного времени, которое требуется для восстановления из такого объема данных.

В PostgreSQL постепенно продвигающаяся вперед *контрольная точка* позволяет безопасно начинать восстановление; предшествующие ей журнальные записи, соответственно, можно удалять. На практике контрольная точка растягивается во времени и фактически превращается в отрезок.

Журнальная запись контрольной точки CHECKPOINT содержит список активных транзакций и список страниц, состояние которых в оперативной памяти отличается от состояния на постоянном носителе. Фоновый процесс записи копирует все изменения из этого списка на диск. При этом нормальная работа системы продолжается. Изменения на страницах, не включенных в список, будут учтены в следующей контрольной точке.

Когда копирование страниц, включенных в контрольную точку, заканчивается, в журнал заносится запись о завершении контрольной точки. После этого может быть создана новая контрольная точка и работа системы продолжается. Наличие записей контрольной точки позволяет при рестарте сервера на фазе повторного выполнения начать просмотр журнала не с самого начала, а с предпоследней контрольной точки. Заметим, что запись начала контрольной точки избыточна и нужна только для упрощения процесса анализа при восстановлении. В PostgreSQL начало контрольных точек в журнале не отмечается.

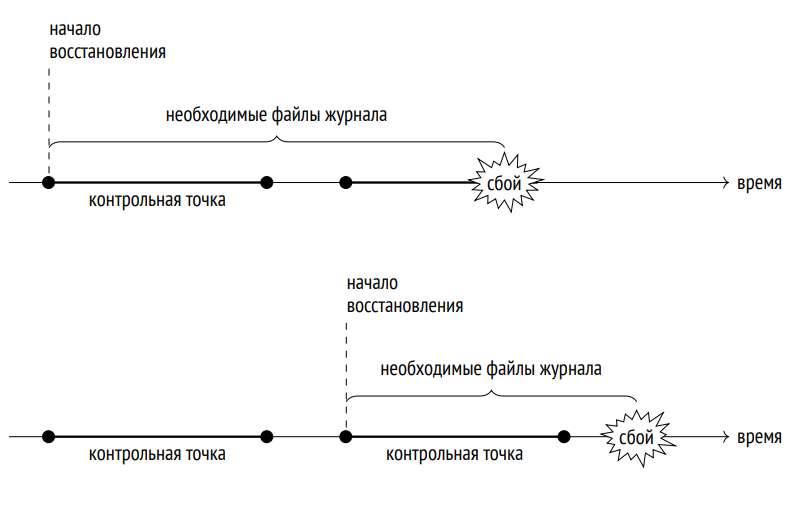


Рис. 2

Таким образом, механизм контрольной точки позволяет определить точку во времени, с которой следует начать восстановление, а также сокращает объём WAL-записей, необходимых серверу. Изменяя параметры *checkpoint\_timeout[[3]](#footnote-3)* и *max\_wal\_size[[4]](#footnote-4),* можно регулировать объём генерируемых журнальных файлов.

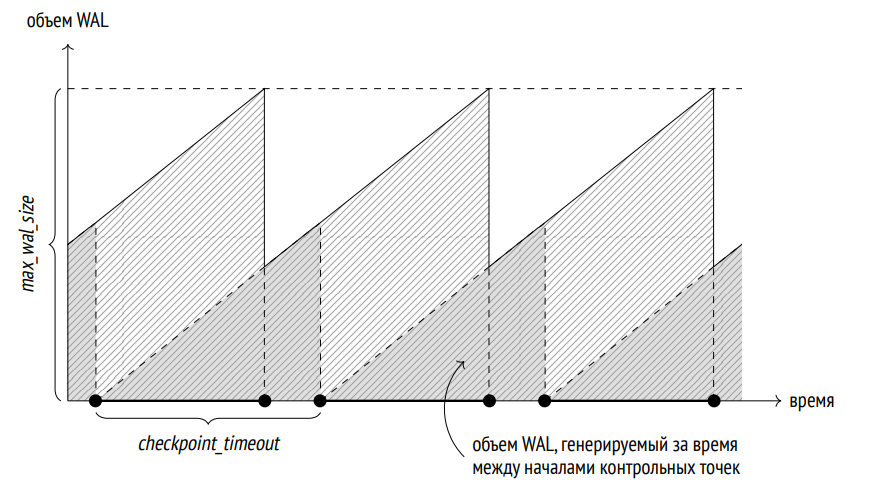


Рис. 3

Основная задача журнала предзаписи — обеспечить возможность восстановления после сбоя. Но журнал можно использовать и для решения других задач, расширив состав входящих в него записей. Есть несколько уровней журналирования: *minimal*, *replica* и *logical*. Каждый следующий уровень включает в себя все, что попадает в журнал предыдущего уровня, и добавляет некоторую новую информацию. Используемый уровень задается параметром *wal\_level* (по умолчанию, replica); его изменение требует перезапуска сервера.

**Minimal**

Гарантирует только восстановление после сбоя. Для экономии места операции с отношениями, созданными или опустошенными в текущей транзакции, не записываются в журнал, если они связаны со вставкой большого объема данных. Вместо журналирования необходимые данные сразу же сбрасываются на диск, а изменения в системном каталоге становятся видимыми при фиксации транзакции.

**Replica**

Гарантируется возможность восстановить данные из резервной копии и использовать физическую репликацию. В журнал попадают все операции, необходимые для полного восстановления.

**Logical**

Обеспечивает возможность логического декодирования и логической репликации. Он должен быть включен на публикующем сервере. С точки зрения журнальных записей этот уровень практически не отличается от уровня replica.

**3.Преимущества и недостатки использования журнала предзаписи**

Основное преимущество журнала предзаписи — это надежность и целостность данных. Процесс журналирования устроен так, что запись об измении сохраняется на диск раньше, чем это изменение, что гарантирует возможность восстановления базы данных до консистентного состояния.

WAL используется для асинхронной репликации и архивации, что позволяет создать резервные копии базы данных на других серверах для обеспечения высокой доступности и распределенной нагрузки.

Главный недостаток - запись всех изменений в журнал увеличивает использование дискового пространства, что может стать проблемой при больших объемах данных и высокой нагрузке. Постоянная запись в журнал и необходимость синхронизации данных могут создавать накладные расходы, влияющие на общую производительность системы.

Недостатком стратегии копирования и восстановления на основе копий в последовательных файлах является большое время восстановления. Очевидно, что большое время восстановления отрицательно влияет на характеристику доступности, которая считается крайне важной для некоторых классов приложений.

Хоть PostgreSQL и сбрасывает журнал на диск раньше изменений, нет гарантии, что журнальные записи будут сохранены, так как системный вызов записи на диск приводит лишь к тому, что операционная система переносит данные в свой кеш (также находящийся в оперативной памяти и подверженный сбоям). Если не предпринимать специальных мер, момент надежного сохранения данных остается неизвестным.

Для достижения оптимальной производительности и надежности требуется тонкая настройка параметров WAL, что может быть сложным и требовать опыта и знаний.

Контрольные точки, необходимые для управления размером журнала и сокращения времени восстановления, могут быть ресурсоемкими и времязатратными, особенно при высоких нагрузках.

Оборудование несовершенно, и данные могут повредиться в памяти или на носителе, измениться при передаче по интерфейсным кабелям. Повреждение WAL-файлов может затруднить восстановление базы данных, хотя PostgreSQL включает механизмы для минимизации таких рисков, например, чтобы вовремя обнаружить возникшую проблему, журнальные записи всегда снабжаются контрольными суммами. Страницы данных также можно защитить контрольными суммами.

На низком уровне запись происходит блоками, поэтому при сбое страница данных может записаться частично. При восстановлении бессмысленно применять к такой поврежденной странице обычные журнальные записи. Для защиты от частичной записи PostgreSQL сохраняет в журнале полный образ страницы при первом ее изменении после начала контрольной точки. Этим поведением управляет параметр *full\_page\_writes*, но его отключение может привести к неустранимому повреждению данных.

**4.Восстановление базы данных после сбоя с помощью журнала предзаписи в PostgreSQL**

При старте сервера первым делом запускается процесс *postmaster*. В свою очередь он запускает процесс *startup*, задача которого — обеспечить восстановление, если произошел сбой. Чтобы определить, требуется ли восстановление, startup читает управляющий файл *pg\_control* и проверяет статус кластера. Статус аккуратно остановленного сервера — «shut down». Статус «in production» у неработающего сервера свидетельствует о сбое. В этом случае процесс startup автоматически выполняет восстановление с позиции начала последней пройденной контрольной точки из того же файла *pg\_control*.

Процесс последовательно читает журнал, начиная с найденной позиции, и применяет журнальные записи к страницам данных, если LSN страницы меньше LSN записи. Если же LSN страницы оказался больше, то запись применять не нужно, а точнее говоря — нельзя, поскольку записи рассчитаны на строго последовательное применение.

Журнальные записи применяются к страницам в буферном кеше, как это происходит и при обычной работе. Аналогично журнальные записи применяются и к файлам: например, если запись говорит о том, что файл должен существовать, а его нет, — файл создается. В заключение выполняется контрольная точка, чтобы зафиксировать на диске восстановленное состояние. На этом работа процесса startup заканчивается.

Для того чтобы восстановление было возможно, необходимо периодически создавать резервную копию базы данных, а в промежутках между созданиями копии архивировать все сегменты журнала, порождаемые в результате нормальной работы системы.

Существует два метода создания резервной базы данных:

• Применение программ создания резервных копий, входящих в состав СУБД. Для создания такой копии не требуется останавливать нормальную работу сервера базы данных.

• Копирование файлов базы данных (включая журнал транзакций) средствами операционной системы. Такое копирование может в некоторых системах работать быстрее, но для некоторых СУБД может требоваться остановка сервера баз данных, что, конечно, влияет на доступность системы.

Поскольку процедура восстановления предусматривает повторное внесение изменений по журналу в том порядке, в котором эти изменения выполнялись, возможно восстановление не только последнего согласованного состояния баз данных до момента отказа, но и состояния на любой предшествующий момент времени (после завершения записи резервной копии). Такая возможность полезна в том случае, когда необходимость восстановления вызвана, например, некорректностью работы приложений или ошибками персонала.

В системе PostgreSQL предусмотрено завершение процесса восстановления по одному из следующих критериев:

• достигнуто согласованное состояние базы данных;

• достигнут явно указанный момент времени;

• выполнена транзакция с указанным идентификатором;

• достигнута заранее созданная именованная точка восстановления.

**5.Альтернативные методы резервного копирования в PostgreSQL**

В PostgreSQL существует несколько методов резервного копирования, каждый из которых имеет свои особенности, преимущества и недостатки. Выбор метода зависит от конкретных требований и условий эксплуатации, включая размер базы данных, требования к времени восстановления, доступные ресурсы и уровень необходимой надежности. В некоторых случаях может быть целесообразно использовать комбинацию различных методов для достижения оптимальных результатов.

Один из основных методов — физическое резервное копирование. Утилита *pg\_basebackup* используется для создания полного физического резервного копирования базы данных. Она проста в использовании и позволяет создавать горячие резервные копии (без остановки базы данных). Как правило создание полной копии может занять много времени и потребовать значительного дискового пространства.

Другим способом физического копирования могут служить возможности файловой системы для создания снимков состояния данных (Filesystem Snapshots). Снимки очень быстро созданиются снимков и оказывают минимальное воздействие на производительность, но требуют поддержки со стороны файловой системы, может потребоваться дополнительное ПО для управления.

Логическое резервирование — это набор команд SQL, восстанавливающая кластер (или базу данных, или отдельный объект) с нуля. Утилита *pg\_dumpall* создает логическую резервную копию всех баз данных в кластере PostgreSQL. *Pg\_dumpall* позволяет выполнить полное резервное копирование всего кластера, включая роли и конфигурационные параметры. Такой метод копирования медленнее по сравнению с физическим резервным копированием, повышается сложность восстановления больших объемов данных.

Основные способы резервного копирования — WAL-архивирование и потоковая репликация. Архивирование позволяет сохранить историю всех изменений в базе данных, гарантирует высокую надежность и позволяет восстановление до любого момента времени. Но требует большого количества дискового пространства, а также надёжность физического хранилища.

Репликация — это передача изменений в реальном времени на рабочую копию основного сервера, что обеспечивает высокую доступность данных, а также переключение работы на реплику в случае сбоя основного сервера. Требует дополнительных серверов для реплик и настройки и администрирования.

**Заключение**

В ходе проведения практики были успешно выполнены все поставленные задачи. Была изучена теоретическая информация о работе журнала предзаписи (WAL). Также журнал предзаписи был рассмотрен в контексте его применения в кластере СУБД PostgreSQL, а именно как WAL обеспечивает надежность и целостность данных, позволяет восстановить работу базы данных после сбоя. Были выявлены преимущества и недостатки журнала предзаписи. Рассмотрен процесс восстановления сервера с использованием WAL-записей. А также были найдены альтернативные методы резервного копирования данных.

В начале практики возникли трудности, связанные с освоением сложных архитектурных аспектов кластера PostgreSQL, однако благодаря упорному труду и поддержке наставников, эти трудности были преодолены.

Как итог моя самооценка в этой области заметно выросла. Я повысила уровень аналитических навыков, получила глубокое понимание механизма журнала предзаписи и научилась самостоятельному анализу архитектурных решений. Практика стимулировала дальнейшее стремление к профессиональному росту и развитию компетенций в области безопасности и сохранности данных.

**Список литературы**

1. Рогов, Е. В. (2024). PostgreSQL 16 изнутри. Москва: ДМК Пресс. - 664 с.

2. Новиков, Б. А., Горшкова, Е. А., Графеева, Н. Г. (2020). Основы технологий баз данных (2-

е изд.). Под ред. Е. В. Рогова. Москва: ДМК Пресс. - 582 с.

3. Goetz Graefe, Wey Guy, Caetano Sauer. (2016). Instant Recovery with Write-Ahead Logging. Page Repair, System Restart, Media Restore, and System Failover (2nd Edition). Springer Nature Switzerland. - 69 с.

4. Alex Petrov. (2019). Database Internals : a deep dive into how distributed data systems work. O'Reilly Media. - 280 с.

5. Joseph M. Hellerstein, Michael Stonebraker, James Hamilton. (2007). Architecture of a database system. [Электронный ресурс]. URL: <https://dsf.berkeley.edu/papers/fntdb07-architecture.pdf>

**Приложение А**

typedef struct XLogRecord

{

uint32 xl\_tot\_len; /\* total len of entire record \*/

TransactionId xl\_xid; /\* xact id \*/

XlogRecPtr xl\_prev; /\* ptr to previous record in log \*/

uint8 xl\_info; /\* flag bits \*/

RmgrId xl\_rmid; /\* resource manager for this record \*/

/\* 2 bytes of padding here, initialize to zero \*/

pg\_crc32 xl\_crc; /\* CRC for this record \*/

} XLogRecord;

**Приложение Б**

typedef struct XLogRecordBlockHeader

{

uint8 id; /\* block reference ID \*/

uint8 fork\_flags; /\* fork within the relation, and flags \*/

uint16 data\_length; /\* number of payload bytes (not including page image) \*/

} XLogRecordBlockHeader;

**Приложение В**

typedef struct XLogRecordBlockImageHeader

{

uint16 length; /\* number of image bytes \*/

uint16 hole\_offset; /\* number of bytes before «hole» \*/

uint8 bimg\_info; /\* flag bits \*/

} XLogRecordBlockImageHeader;

**Приложение Г**

typedef struct XLogRecordBlockCompressHeader

{

uint16 hole\_length; /\* number of bytes in «hole» \*/

} XlogRecordBlockCompressHeader;

**Приложение Д**

typedef struct RelFileLocator

{

Oid spcOid; /\* tablespace \*/

Oid dbOid; /\* database \*/

RelFileNumber relNumber; /\* relation \*/

} RelFileLocator;

1. REDO — roll-forward recovery. Восстановление работоспособности системы с повтором всех завершённых транзакций. [↑](#footnote-ref-1)
2. UNDO — roll-backward recovery. Восстановление работоспособности системы с удалением изменений всех незавершённых транзакций. [↑](#footnote-ref-2)
3. checkpoint\_timeout — интервал между контрольными точками (по умолчанию, 5 мин). [↑](#footnote-ref-3)
4. max\_wal\_size — объём журнальных записей, при превышении которого сервер инициирует внеплановую контрольную точку (по умолчанию, 1 ГБ). [↑](#footnote-ref-4)