ГЛАВА 5

Обработка транзакций и восстановление

В этой книге мы начали изучать концепции СУБД «снизу вверх» и уже рассмотрели структуры хранения данных. Пора перейти к изучению компонентов более высокого уровня, отвечающих за управление буфером, управление блокировками и восстановление. Понимание их внутреннего устройства является необходимым условием для понимания транзакций баз данных.

Транзакция — это неделимая логическая единица работы в СУБД, позволяющая представить несколько операций в виде единого шага. Операции, выполняемые транзакциями, включают в себя чтение и запись значений базы данных. Транзакция базы данных должна обладать свойствами *атомарности*, *согласованности*, *изолированности* и *долговечности* (atomicity, consistency, isolation, durability, ACID) [HAERDER83].

Атомарность

Стадии транзакции *неотделимы* друг от друга, т. е. либо все стадии, связанные с транзакцией, выполняются успешно, либо не выполняется ни одна из них. Иными словами, транзакции не должны выполняться частично. Каждая транзакция может быть либо *зафиксирована* (committed) (в случае чего становятся видимыми все изменения, вносимые операциями записи, выполняемыми в рамках транзакции), либо *прервана* (в случае чего производится откат всех побочных эффектов транзакции, которые еще не стали видимыми). Фиксация — это заключительная операция. После прерывания может быть предпринята повторная попытка выполнения транзакции.

Согласованность

Согласованность — свойство, зависящее от приложения; транзакция должна переводить базу данных из одного допустимого состояния в другое допустимое состояние, сохраняя все инварианты базы данных (включая различные ограничения, ссылочную целостность и т. д.). Согласованность является наиболее слабо определенным свойством, возможно, потому, что это единственное свойство, которое контролируется пользователем, а не только самой базой данных.

Изолированность

Несколько параллельно выполняемых транзакций должны выполняться без взаимного влияния друг на друга, как если бы в это время не выполнялись никакие другие транзакции.

Изолированность определяет, *когда* должны становиться видимыми изменения в состоянии базы данных и какие изменения должны быть видимыми для параллельных транзакций. Ради повышения производительности многие базы данных используют более слабые уровни изолированности по сравнению с данным определением. В зависимости от методов и подходов, используемых для управления параллелизмом, изменения, вносимые транзакцией, могут быть видны или не видны другим параллельным транзакциям (см. подраздел «Уровни изолированности» на с. 114).

Долговечность

После фиксации транзакции все изменения в состоянии базы данных должны персистентно храниться на диске и сохраняться в случае перебоев в электроснабжении и сбоев и отказов системы.

Для реализации транзакций в СУБД в дополнение к структуре хранения, которая организует и сохраняет данные на диске, требуется совместная работа нескольких компонентов. Внутри узла *диспетчер транзакций* координирует, планирует и отслеживает транзакции и их отдельные стадии.

Диспетчер блокировок контролирует доступ к этим ресурсам и не допускает выполнения параллельных обращений, способных нарушить целостность данных. При каждом запросе блокировки диспетчер проверяет, не предоставлена ли она уже какой-либо другой транзакции в совмещаемом или исключительном режиме, и предоставляет блокировку при наличии требуемого уровня доступа. Поскольку исключительная блокировка в любой заданный момент может быть предоставлена только одной транзакции, для выполнения других запрашивающих ее транзакций необходимо дождаться освобождения блокировки либо прервать их выполнение и предпринять повторную попытку позже. После освобождения блокировки или завершения соответствующей транзакции диспетчер блокировок уведомляет одну из ожидающих транзакций, позволяя ей захватить блокировку и продолжить работу.

Кэш страниц выступает в роли посредника между персистентным носителем (диском) и остальной частью подсистемы хранения. Он аккумулирует изменения состояния в оперативной памяти и служит в качестве кэша для страниц, которые еще не были синхронизированы с персистентным носителем. Все изменения состояния базы данных сначала применяются к кэшированным страницам.

Диспетчер журналов хранит историю операций (записи журнала), примененных к кэшированным страницам, но еще не синхронизированных с персистентным носителем, чтобы изменения не были потеряны в случае сбоя. То есть журнал используется для повторного применения этих операций и восстановления кэшированного состояния во время запуска. Записи журнала также можно использовать для отмены изменений, внесенных прерванными транзакциями.

Распределенные (многосоставные) транзакции требуют дополнительной координации и удаленного выполнения. Протоколы распределенных транзакций будут рассмотрены в главе 13.

Организация буферизации данных

Большинство баз данных использует двухуровневую схему организации памяти, где одним уровнем является более медленный персистентный носитель (диск), а другим — более быстрая оперативная память (ОЗУ). Чтобы снизить количество обращений к персистентному носителю, страницы *кэшируются* в ОЗУ. При повторном запрашивании страницы уровнем хранения возвращается ее кэшированная копия.

Имеющиеся в памяти кэшированные страницы можно повторно использовать при условии, что другие процессы не будут вносить изменения в данные на диске. Такой подход иногда называют использованием *виртуального диска* [BAYER72]. При чтении виртуального диска данные считываются из физического хранилища только в том случае, если в памяти еще нет копии считываемой страницы. Более распространенное название этой концепции — *кэш страниц*, или *буферный пул*. Кэш страниц отвечает за кэширование страниц, считываемых с диска в память. В случае сбоя СУБД или некорректного завершения работы кэшированное содержимое теряется.

Поскольку термин *кэш страниц* лучше отражает назначение этой структуры, в этой книге по умолчанию используется это название. Термин *буферный пул* звучит так, как будто его основная цель — объединение и повторное использование *пустых* буферов без совместного использования их содержимого, что может быть полезной составляющей кэширования страниц и даже может использоваться в виде отдельного компонента, но не отражает целиком назначение этого механизма.

Кэширование страниц не ограничивается рамками баз данных. В операционных системах также есть понятие кэша страниц. Операционные системы используют *незанятые* сегменты памяти, чтобы прозрачно кэшировать содержимое диска для повышения производительности системных вызовов ввода-вывода.

Процесс загрузки некэшированных страниц в память называют *подкачкой страниц*. Если в кэшированную страницу вносятся изменения, она считается *грязной* до тех пор, пока эти изменения не будут *выгружены* (или *сброшены*) на диск.

Поскольку область памяти, предназначенная для хранения кэшированных страниц, обычно гораздо меньше, чем весь набор данных, кэш страниц в конечном итоге заполняется, после чего для загрузки новой страницы уже требуется *вытеснить* из кэша одну из кэшированных страниц.

На рис. 5.1 можно увидеть связь между логическим представлением страниц B-дерева, их кэшированными версиями и страницами на диске. Кэш страниц загружает страницы в свободные слоты в произвольном порядке, поэтому нет прямого соответствия между порядком следования страниц на диске и в памяти.

Основные функции кэша страниц можно свести к следующему:

- Сохраняет содержимое кэшированных страниц в памяти.
- Позволяет буферизировать изменения, вносимые в дисковые версии страниц и вносить их в кэшированные версии.

- Если запрашиваемая страница отсутствует в памяти и для нее имеется достаточно свободного места, она загружается в кэш страниц, а затем возвращается ее кэшированная версия.
- Если запрашиваемая страница уже находится в кэше, то просто возвращается ее кэшированная версия.
- Если для новой страницы недостаточно свободного места, некоторая другая страница *вытесняется* из кэша, а ее содержимое *выгружается* на диск.

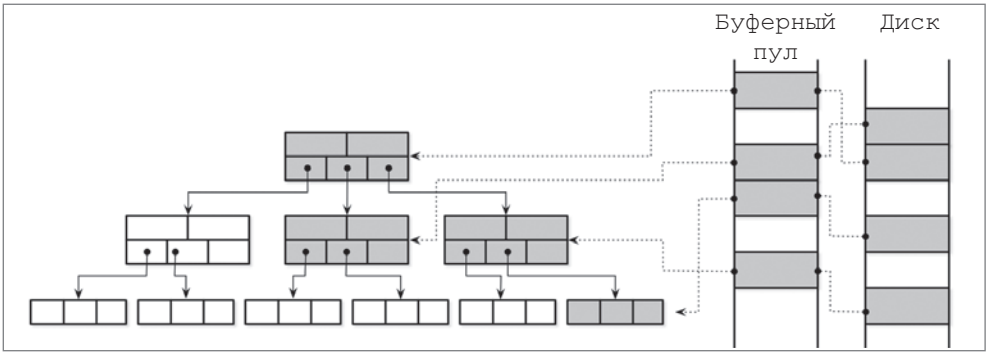

Рис. 5.1. Кэш страниц

ОБХОД СТРАНИЧНОГО КЭША ЯДРА

Многие СУБД открывают файлы с помощью флага `O_DIRECT`. Этот флаг позволяет системным вызовам ввода-вывода обходить кэш страниц ядра, обращаясь к диску напрямую и используя управление буфером конкретной базы данных. К этому порой неодобрительно относятся разработчики операционных систем.

Линус Торвальдс не одобряет (https://databass.dev/links/32) использование флага `O_DIRECT` по той причине, что такой подход не асинхронен и не предусматривает упреждающее чтение или другие средства для информирования ядра о схемах доступа. Однако до тех пор, пока операционные системы не начнут предлагать лучшие механизмы, флаг `O_DIRECT` по-прежнему будет полезен.

Мы можем получить некоторый контроль над процессом вытеснения страниц из кэша операционной системы, используя функцию `fadvise` (https://databass.dev/links/33), но такой подход позволяет лишь попросить ядро учесть наше мнение и не гарантирует, что оно так и сделает. Чтобы отказаться от системных вызовов при вводе-выводе, мы можем использовать отображение в память, но тогда мы теряем контроль над кэшированием.

Семантика кэширования

Все изменения, внесенные в буферы, сохраняются в памяти, пока они не будут записаны обратно на диск. Поскольку ни один другой процесс не может вносить

изменения в резервный файл, эта синхронизации производится в одном направлении — из памяти на диск, но не наоборот. Кэш страниц обеспечивает базе данных бóльшую степень контроля над управлением памятью и доступом к диску. Его можно рассматривать как специфичный для приложения эквивалент кэша страниц ядра: он напрямую обращается к устройству блочного ввода-вывода, реализует аналогичную функциональность и служит той же цели. Он абстрагирует доступ к диску и отделяет логические операции записи от физических.

Кэширование страниц позволяет хранить часть дерева в памяти без внесения дополнительных изменений в алгоритм и реализации объектов в памяти. Все, что нам нужно сделать, — это заменить операции доступа к диску вызовами кэша страниц.

Когда подсистема хранения обращается к странице (т. е. запрашивает ее), мы сначала проверяем, не кэшировано ли уже ее содержимое, в случае чего возвращается содержимое кэшированной страницы. Если содержимое страницы еще не кэшировано, кэш преобразует логический адрес страницы или номер страницы в свой физический адрес, загружает ее содержимое в память и возвращает кэшированную версию подсистеме хранения. При этом подсистема хранения получает ссылку на буфер с содержимым кэшированной страницы, которую необходимо вернуть обратно кэшу страницы или удалить после завершения операции. Также можно дать кэшу страниц указание не вытеснять страницу из кэша, *закрепив* ее.

После изменения страницы (например, при добавлении ячейки) она помечается как «грязная». Если на странице установлен «грязный» флаг, это говорит о том, что ее содержимое не синхронизировано с диском и должно быть выгружено для обеспечения долговечности данных.

Вытеснение из кэша

Нужно стремиться к тому, чтобы кэш всегда был заполненным: это позволит производить больше операций чтения без обращения к персистентному носителю и буферизировать больше операций записи, выполняемых в рамках одной страницы. Однако емкость кэша страниц ограничена, и рано или поздно для доставки нового содержимого потребуется вытеснять старые страницы. Если содержимое страницы синхронизировано с диском (т. е. уже выгружено или никогда не изменялось), а страница не закреплена и на нее нет ссылок, то ее можно сразу же вытеснить. «Грязные» страницы перед вытеснением из кэша необходимо *выгрузить на диск*. Страницы, на которые есть ссылки, не должны вытесняться, пока их использует какой-то другой поток.

Так как запуск выгрузки при каждом вытеснении может плохо сказаться на производительности, некоторые базы данных используют отдельный фоновый процесс, который циклически обходит те «грязные» страницы, которые могут подвергаться вытеснению, и обновляет их дисковые версии. Например, именно это делает фоновый модуль выгрузки (https://databass.dev/links/34) в СУБД PostgreSQL.

Также важно не забывать о необходимости обеспечения *долговечности*: если в базе данных произойдет сбой, будут потеряны все невыгруженные данные. Чтобы обес-

печить персистентное сохранение всех изменений, операции выгрузки координируются процессом создания *контрольных точек*. Этот процесс управляет журналом упреждающей записи и кэшем страниц и обеспечивает их согласованную работу. Из журнала упреждающей записи могут удаляться только те записи, которые относятся к операциям, выполненным на уже выгруженных кэшированных страницах. «Грязные» страницы не подлежат вытеснению из кэша до завершения этого процесса.

Это означает, что всегда приходится находить компромисс между несколькими целями:

- Откладывать выгрузку, чтобы уменьшить количество обращений к диску.
- Выполнять упреждающую выгрузку страниц для обеспечения быстрого вытеснения.
- Выбирать страницы для вытеснения и выгрузки в оптимальном порядке.
- Поддерживать размер кэша в пределах выделенной для него области памяти.
- Не допускать потерь данных до их персистентного сохранения на основном носителе.

Далее мы рассмотрим ряд методов, позволяющих улучшить первые три характеристики без выхода за пределы, указанные в последних двух пунктах.

Блокировка страниц в кэше

Выполнять дисковый ввод-вывод при выполнении каждой операции чтения или записи нецелесообразно, поскольку последовательные операции чтения могут запрашивать одну и ту же страницу, а последовательные операции записи могут модифицировать одну и ту же страницу. Поскольку B-дерево сужается кверху, узлы более высокого уровня (расположенные ближе к корню) затрагиваются в ходе большинства операций чтения. Операции разделения и слияния также в конечном счете распространяются до узлов более высокого уровня. Это означает, что всегда имеется как минимум часть дерева, кэширование которой дает значительную пользу.

Мы можем «заблокировать» те страницы, которые с большой вероятностью будут использованы в ближайшее время. Блокировка страниц в кэше называется *закреплением*. Закрепленные страницы хранятся в памяти дольше, что позволяет сократить число обращений к диску и повысить производительность [GRAEFE11].

Поскольку каждый более низкий уровень B-дерева имеет экспоненциально больше узлов по сравнению с предыдущим, в силу чего узлы более высоких уровней составляют лишь небольшую часть дерева, мы можем держать эту часть дерева в памяти постоянно, подкачивая остальные части по мере необходимости. Это означает, что при выполнении запроса не потребуется делать h обращений к диску (где h — высота дерева, как упоминалось в подразделе «Сложность поиска в B-дереве» на с. 55); достаточно будет считать с диска только некэшированные страницы нижних уровней.

Операции, выполняемые с поддеревом, могут порождать противоречащие друг другу структурные изменения — так, после нескольких операций удаления, порождающих

слияние, могут быть выполнены операции записи, порождающие разбиение, или наоборот. То же самое справедливо и для структурных изменений, распространяющихся из разных поддеревьев (которые происходят близко друг к другу по времени в разных частях дерева и распространяются вверх). Эти операции можно буферизовать, применяя изменения только в памяти, что позволит уменьшить количество операций записи на диск и снизить затраты на операции, так как вместо нескольких операций записи будет выполняться только одна.

ПРЕДВАРИТЕЛЬНАЯ ВЫБОРКА И МГНОВЕННОЕ ВЫТЕСНЕНИЕ

Кэш страниц также обеспечивает подсистеме хранения точный контроль над предварительной выборкой и вытеснением из кэша. Кэшу страниц можно дать указание загружать страницы заранее, еще до обращения к ним. Например, при обходе листовых узлов во время сканирования диапазона можно предварительно загружать следующие листы. Аналогичным образом в случае загрузки страницы процессом обслуживания ее можно вытеснять из кэша сразу после завершения этого процесса, так как она вряд ли пригодится для текущих запросов. Некоторые базы данных, например PostgreSQL (https://databass.dev/links/35), используют циклический буфер (т. е. производят замену страниц по принципу FIFO) при сканировании больших последовательных областей данных.

Замещение страниц

При достижении предельной емкости кэша для загрузки новых страниц требуется вытеснять старые страницы. Однако следует вытеснять именно те страницы, которые с наименьшей вероятностью понадобятся в ближайшее время, иначе нам придется загружать вытесняемые страницы снова и снова, хотя мы могли бы просто держать их в памяти. Для оптимизации процесса нам нужно каким-то образом оценивать вероятность последующего доступа к странице.

Страницы необходимо вытеснять в соответствии с *политикой вытеснения* (также иногда называемой *политикой замещения страниц*). Согласно этой политике, для вытеснения выбираются страницы, которые с наименьшей вероятностью понадобятся в ближайшее время. После вытеснения страницы на ее место в кэше можно загрузить новую страницу.

Чтобы реализация кэша страниц могла работать эффективным образом, она должна использовать эффективный алгоритм замещения страниц. Идеальная стратегия сводится к тому, чтобы каким-то образом предсказать точную последовательность использования страниц и вытеснить только те страницы, которые не будут использоваться максимально долго. Поскольку последовательность запросов редко соответствует какому-либо конкретному паттерну или способу распределения, предсказать ее точно практически невозможно, однако мы можем уменьшить количество вытеснений, используя правильную стратегию замещения страниц.

На первый взгляд может показаться, что уменьшить количество вытеснений можно просто путем увеличения кэша. Однако на самом деле это не так. Один из примеров,

демонстрирующих эту дилемму, называется *аномалией Белади* [BEDALY69]. Он показывает, что увеличение числа страниц может увеличить число вытеснений, если используется неоптимальный алгоритм замещения страниц. Когда страницы, которые могут потребоваться в ближайшее время, вытесняются, а затем загружаются снова, возникает конкуренция страниц за место в кэше. Поэтому нужно как следует продумать используемый алгоритм, чтобы он улучшал, а не ухудшал ситуацию.

FIFO и LRU

Простейшей стратегией замещения страниц является принцип «первым пришел — первым ушёл» (first in — first out, FIFO). Алгоритм FIFO поддерживает очередь идентификаторов страниц, расположенных в порядке их вставки, добавляя новые страницы в хвост очереди. Когда кэш страниц полностью заполняется, извлекается элемент из вершины очереди, соответствующий странице, которая была загружена раньше других. Поскольку данный алгоритм учитывает только событие загрузки в кэш и не берет в расчет последующие обращения к странице, его нецелесообразно использовать в большинстве реальных систем. Например, корневые и самые верхние страницы загружаются первыми и, согласно этому алгоритму, являются первыми кандидатами на вытеснение, хотя, согласно свойствам структуры дерева, эти страницы потребуется загрузить снова если не сразу, то в самое ближайшее время.

Естественным развитием алгоритма FIFO является алгоритм «наиболее давно использовавшийся» (least-recently used, LRU) [TANENBAUM14]. Данный алгоритм также поддерживает очередь кандидатов на вытеснение, расположенных в порядке вставки, но при этом позволяет снова поместить страницу в хвост очереди при повторном обращении к ней, как если бы это была первоначальная загрузка страницы в кэш. Однако обновление ссылок и повторное связывание узлов при каждом обращении могут оказаться затратными в параллельной среде.

Существуют и другие стратегии вытеснения из кэша на основе LRU. Например, алгоритм 2Q (LRU с двумя очередями) поддерживает две очереди и помещает страницы в первую очередь при первоначальном обращении и во вторую «горячую» очередь — при последующих обращениях, что позволяет различать недавно и часто запрашиваемые страницы [JONSON94]. Алгоритм LRU-K определяет, ссылки на какие страницы используются наиболее часто, отслеживая последние K обращений и используя эту информацию для оценки количества обращений к каждой странице [ONEIL93].

CLOCK

В некоторых ситуациях эффективность может быть важнее точности. В качестве альтернативы для алгоритма LRU часто используются различные варианты алгоритма CLOCK. Они отличаются компактностью и дружественностью к кэшу и могут использоваться в параллельных средах [SOUNDARARARJAN06]. Один из вариантов этого алгоритма (https://databass.dev/links/36), например, используется в Linux.

Алгоритм CLOCK хранит ссылки на страницы и связанные с ними биты доступа в циклическом буфере. Некоторые варианты используют счетчики (https://databass.