МИНОБРНАУКИ РОССИИ

Передовые инженерные школы

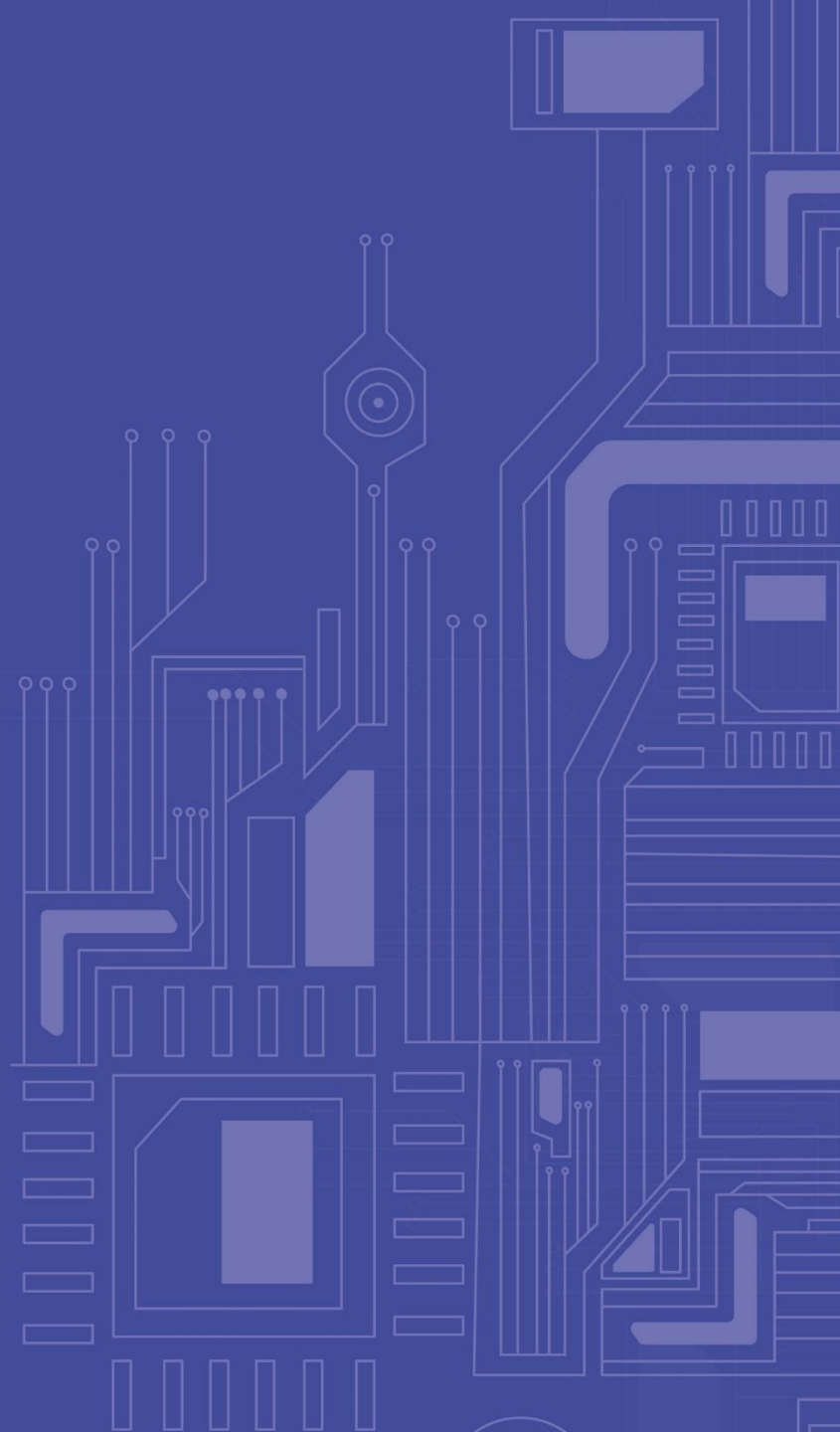
СИСТЕМЫ УПРАВЛЕНИЯ БАЗАМИ ДАННЫХ

Лекция 7

ПЛАН ЛЕКЦИИ

- B+ - деревья
- Хэширование
- Транзакции и их характеристики
- Методы сериализации транзакций

ОРГАНИЗАЦИЯ ИНДЕКСОВ: B+ - ДЕРЕВЬЯ



B+ - ДЕРЕВЬЯ (1/2)

Наиболее популярным подходом к организации индексов в базах данных является использование техники B+-деревьев. Техника B-деревьев была предложена в начале 1970-х гг. Рудольфом Байером (Rudolf Bayer) и Эдом Маккрейтом (Ed McCreight).

С точки зрения **внешнего логического представления** B-дерево – это сбалансированное сильно ветвистое дерево во внешней памяти.

- **Сбалансированность** означает, что длина пути от корня дерева к любому его листу одна и та же.
- **Ветвистость дерева** – это свойство каждого узла дерева ссылаться на большое число узлов-потомков.

С точки зрения **физической организации** B-дерево представляется как мультисписочная структура страниц внешней памяти, т.е. каждому узлу дерева соответствует блок внешней памяти (страница). В B+-дереве внутренние и листовые страницы обычно имеют разную структуру.

Типовая структура внутренней страницы B+-дерева

N_1 ключ$_1$ N_2 ключ$_2$ N_3 ключ$_3$ ... N_m ключ$_m$ N_{m+1}

- ключ$_1$ ≤ ключ$_2$ ≤ ... ≤ ключ$_m$;
- в странице дерева N_m находятся ключи k со значениями ключ$_m$ <= k <= ключ$_{m+1}$.

B+ - ДЕРЕВЬЯ (2/2)

Листовая страница дерева

$ключ_1$ $список_1$ $ключ_2$ $список_2$ . . . $ключ_k$ $список_k$

- $ключ_1 < ключ_2 < \ldots < ключ_k$;
- $список_r$ – упорядоченный список идентификаторов кортежей (tid), включающих значение $ключ_r$;
- листовые страницы связаны одно- или двунаправленным списком.

ПОИСК ПО ДЕРЕВУ

Поиск в B+-дереве – это прохождение от корня к листу в соответствии с заданным значением ключа. Заметим, что поскольку B+-деревья являются сильно ветвистыми и сбалансированными, для выполнения поиска по любому значению ключа потребуется одно и то же (и обычно небольшое) число обменов с внешней памятью.

ГЛУБИНА ДЕРЕВА

Более точно, в сбалансированном дереве, где длины всех путей от корня к листу одни и те же, если во внутренней странице помещается **n** ключей, то при хранении **m** записей требуется дерево глубиной $\log_n(m)$.. Если **n** достаточно велико (обычный случай), то глубина дерева невелика, и производится быстрый поиск.

СБАЛАНСИРОВАННОСТЬ B+ - ДЕРЕВЬЕВ (1/3)

Основной «изюминкой» B+-деревьев является автоматическое поддержание свойства сбалансированности. Рассмотрим, как это делается при выполнении операций занесения и удаления записей.

При занесении новой записи выполняются следующие шаги:

1. **Поиск листовой страницы**. Фактически, производится обычный поиск по ключу. Если в B+-дереве не содержится ключ с заданным значением, то будет получен номер страницы, в которой ему надлежит содержаться, и соответствующие координаты внутри страницы.
2. **Помещение записи на место**. Естественно, что вся работа производится в буферах оперативной памяти. Листовая страница, в которую требуется занести запись, считывается в буфер, и в нем выполняется операция вставки. Размер буфера должен превышать размер страницы внешней памяти.
3. **Если после выполнения вставки новой записи размер используемой части буфера не превосходит размера страницы**, то на этом выполнение операции занесения записи заканчивается. Буфер может быть немедленно вытолкнут во внешнюю память или временно сохранен в основной памяти в зависимости от политики управления буферами.
4. **Если же возникло переполнение буфера** (т.е. размер его используемой части превосходит размер страницы), то выполняется расщепление страницы. Для этого запрашивается новая страница внешней памяти, используемая часть буфера разбивается примерно пополам (так, чтобы вторая половина также начиналась с ключа), и вторая половина записывается во вновь выделенную страницу, а в старой странице модифицируется значение размера свободной памяти. Естественно, модифицируются ссылки по списку листовых страниц.

СБАЛАНСИРОВАННОСТЬ B+ - ДЕРЕВЬЕВ (2/3)

При занесении новой записи выполняются следующие шаги (продолжение):

5. Чтобы обеспечить **доступ от корня дерева к заново заведенной странице**, необходимо соответствующим образом модифицировать внутреннюю страницу, являющуюся предком ранее существовавшей листовой страницы, т.е. вставить в нее соответствующее значение ключа и ссылку на новую страницу. При выполнении этого действия может снова произойти переполнение теперь уже внутренней страницы, и она будет расщеплена на две. В результате потребуется вставить значение ключа и ссылку на новую страницу во внутреннюю страницу-предка выше по иерархии и т.д.
6. Предельным случаем является **переполнение корневой страницы B+-дерева**. В этом случае она тоже расщепляется на две, и заводится новая корневая страница дерева, т.е. его глубина увеличивается на единицу.

СБАЛАНСИРОВАННОСТЬ B+ - ДЕРЕВЬЕВ (3/3)

При удалении записи выполняются следующие шаги:

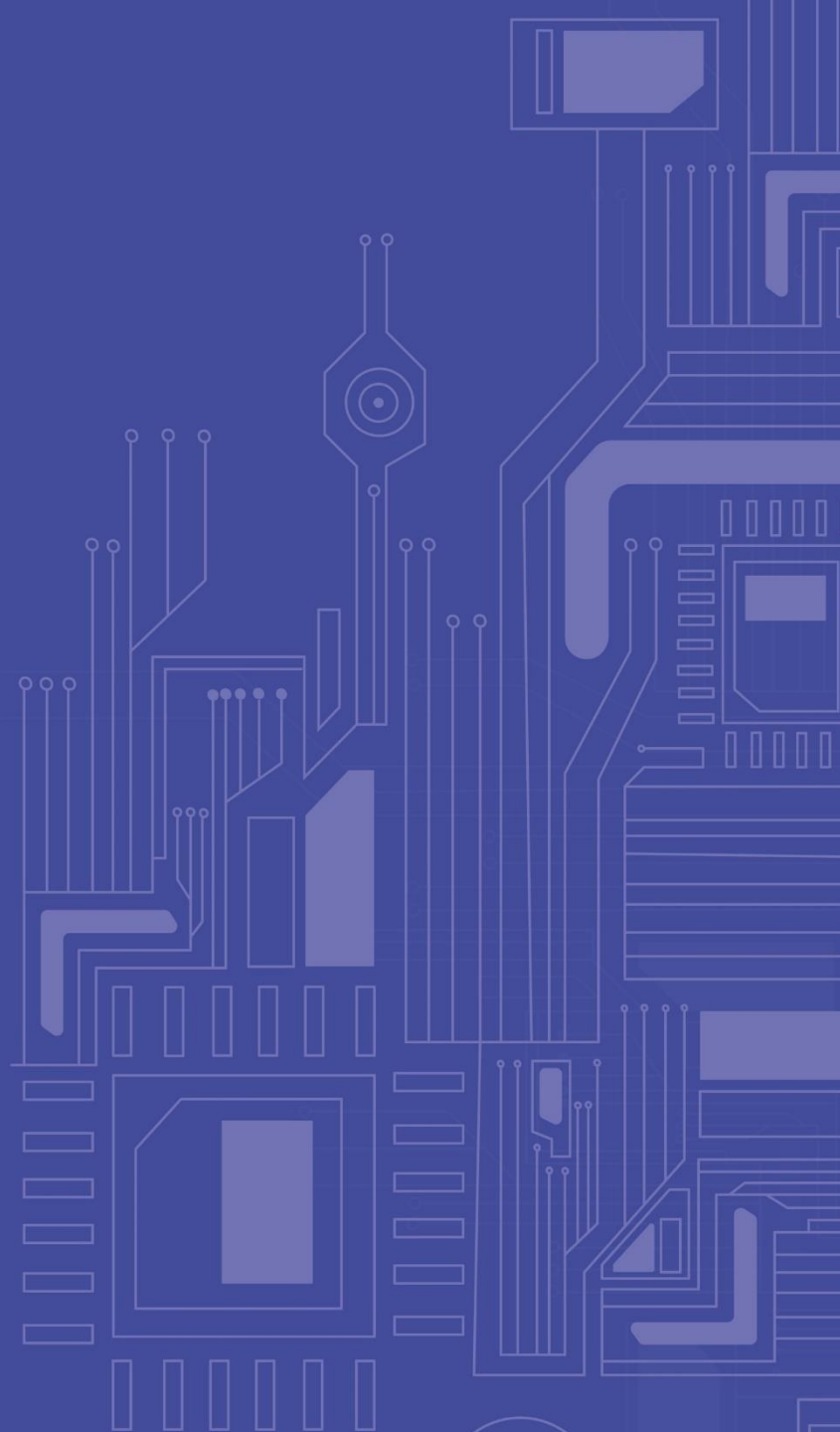
1. Поиск записи по ключу. Если запись не найдена, то удалять ничего не нужно.
2. Реальное удаление записи в буфере, в который прочитана соответствующая листовая страница.
3. Если после выполнения этой подоперации размер занятой в буфере области оказывается таковым, что его сумма с размером занятой области в листовых страницах, являющихся левым или правым братом данной страницы, больше, чем размер страницы, операция завершается.
4. Иначе производится слияние с правым или левым братом, т.е. в буфере производится новый образ страницы, содержащей общую информацию из данной страницы и ее левого или правого брата. Ставшая ненужной листовая страница заносится в список свободных страниц. Соответствующим образом корректируется список листовых страниц.
5. Чтобы устранить возможность доступа от корня к освобожденной странице, нужно удалить соответствующее значение ключа и ссылку на освобожденную страницу из внутренней страницы – ее предка. При этом может возникнуть потребность в слиянии этой страницы с ее левым или правым братом и т.д.
6. Предельным случаем является полное опустошение корневой страницы дерева, которое возможно после слияния последних двух потомков корня. В этом случае корневая страница освобождается, а глубина дерева уменьшается на единицу.

ПРОБЛЕМА B+-ДЕРЕВЬЕВ: МОДИФИКАЦИЯ ДАННЫХ

Проблемой является то, что при выполнении операций модификации слишком часто могут возникать расщепления и слияния. Чтобы добиться эффективного использования внешней памяти с минимизацией числа расщеплений и слияний, применяются более сложные приемы, в том числе:

- упреждающие расщепления, т.е. расщепления страницы не при ее переполнении, а несколько раньше, когда степень заполненности страницы достигает некоторого уровня;
- переливания, т.е. поддержание равновесного заполнения соседних страниц;
- слияния 3-в-2, т.е. порождение двух листовых страниц на основе содержимого трех соседних.

ХЭШИРОВАНИЕ



ОБЩИЕ ПРИНЦИПЫ

Альтернативным и достаточно популярным подходом к организации индексов является использование техники ***хэширования***.

- Общей идеей методов хэширования является применение к значению ключа некоторой *функции свертки (**хэш-функции**)*, вырабатывающей значение меньшего размера. Значение хэш-функции затем используется для доступа к записи. В самом простом, классическом случае свертка ключа используется как адрес в таблице, содержащей ключи и записи.
- Основным требованием к хэш-функции является **равномерное распределение значений свертки** (одним из распространенных видов «хороших» хэш-функций являются функции, выдающие остаток от деления значения ключа на некоторое простое число).
- При возникновении ***коллизий*** (одна и та же свертка для нескольких значений ключа) образуются ***цепочки переполнения.***

Главным ограничением этого метода является фиксированный размер таблицы. Если таблица заполнена слишком сильно или переполнена, но возникнет слишком много цепочек переполнения, и главное преимущество хэширования – доступ к записи почти всегда за одно обращение к таблице – будет утрачено. Расширение таблицы требует ее полной переделки на основе новой хэш-функции (со значением свертки большего размера).

ВИДЫ ХЭШИРОВАНИЯ (1/2)

РАБОТА С ВНЕШНЕЙ ПАМЯТЬЮ

Исходная идея кажется очевидной: если при управлении данными на основе хэширования в основной памяти хэш-функция вырабатывает адрес требуемого элемента, то при обращении к внешней памяти необходимо генерировать номер блока дискового пространства, в котором находится запрашиваемый элемент данных. Основная проблема относится к коллизиям. Если при работе в основной памяти потенциально возникающими потребностями дополнительного поиска информации при возникновении коллизий можно, вообще говоря, пренебречь (поскольку время доступа к основной памяти мало), то при использовании внешней памяти любое дополнительное обращение вызывает существенные накладные расходы.

РАСШИРЯЕМОЕ ХЭШИРОВАНИЕ

В основе подхода *расширяемого хэширования (Extendible Hashing)* лежит принцип использования деревьев цифрового поиска в основной памяти. В основной памяти поддерживается справочник, организованный на основе бинарного дерева цифрового поиска, ключами которого являются значения хэш-функции, а в листовых вершинах хранятся номера блоков записей во внешней памяти. В этом случае любой поиск в дереве цифрового поиска является «успешным», т.е. ведет к некоторому блоку внешней памяти. Входит ли в этот блок искомая запись, обнаруживается уже после прочтения блока в основную память.

Расширяемое хэширование хорошо работает в условиях динамически изменяемого набора записей в хранимом файле, но требует наличия в основной памяти справочного дерева.

ВИДЫ ХЭШИРОВАНИЯ (2/2)

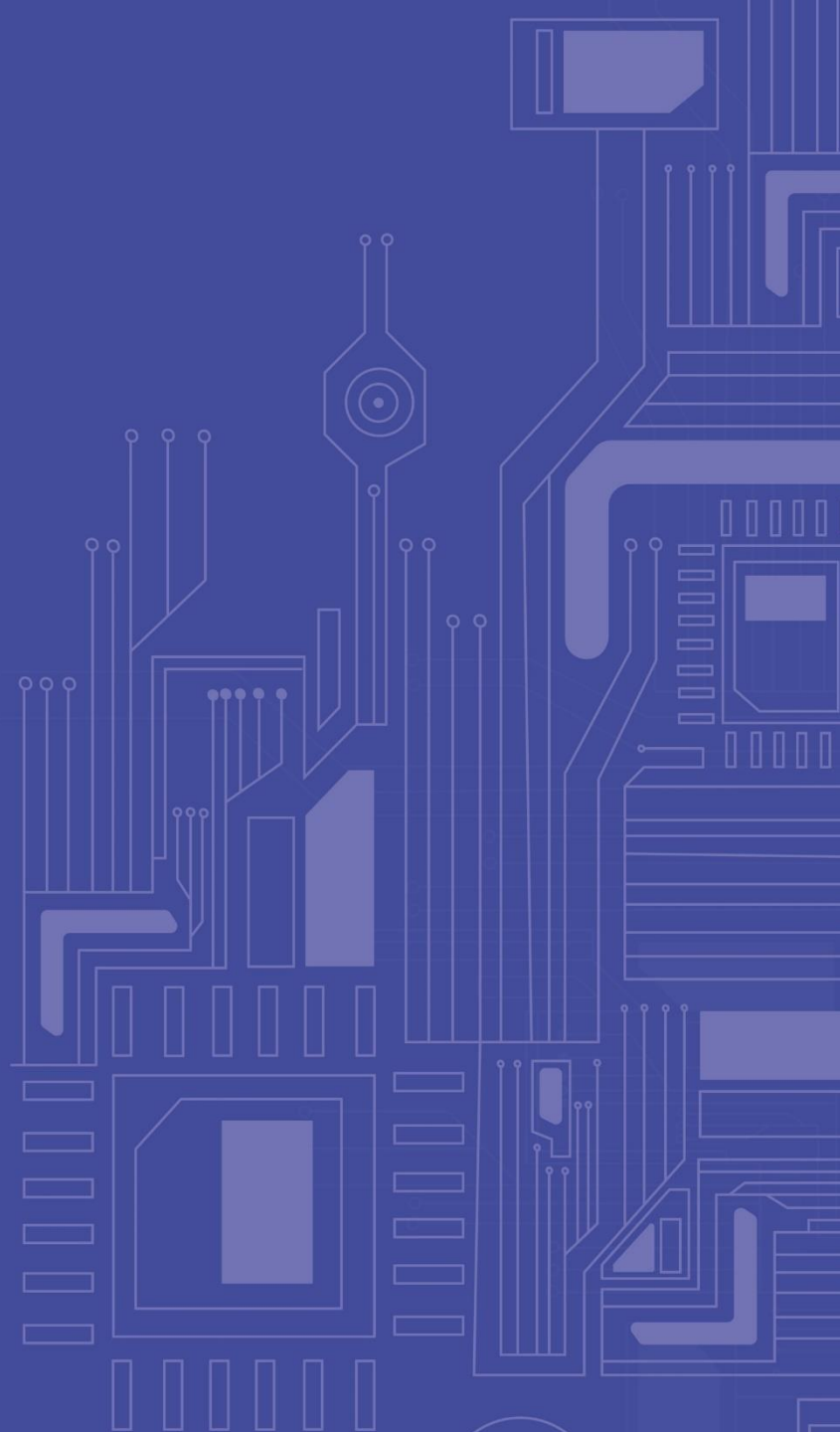
ПРОБЛЕМЫ КОЛЛИЗИЙ ПРИ РАСШИРЯЕМОМ ХЭШИРОВАНИИ

Проблема коллизий переформулируется следующим образом. Как таковых, коллизий не существует. Может возникнуть лишь ситуация переполнения блока внешней памяти. Значение хэш-функции указывает на этот блок, но места для включения записи в нем уже нет. Эта ситуация обрабатывается так. Блок расщепляется на два, и дерево цифрового поиска переформируется соответствующим образом. Конечно, при этом может потребоваться расширение самого справочника.

ЛИНЕЙНОЕ ХЭШИРОВАНИЕ

Идея *линейного хэширования (Linear Hashing)* состоит в том, чтобы можно было обойтись без поддержания справочника в основной памяти. Основой метода является то, что для адресации блока внешней памяти всегда используются младшие биты значения хэш-функции. Если возникает потребность в расщеплении, то записи перераспределяются по блокам так, чтобы адресация осталась правильной.

ЖУРНАЛ И СЛУЖЕБНАЯ ИНФОРМАЦИЯ (ОБЩАЯ КОНЦЕПЦИЯ)



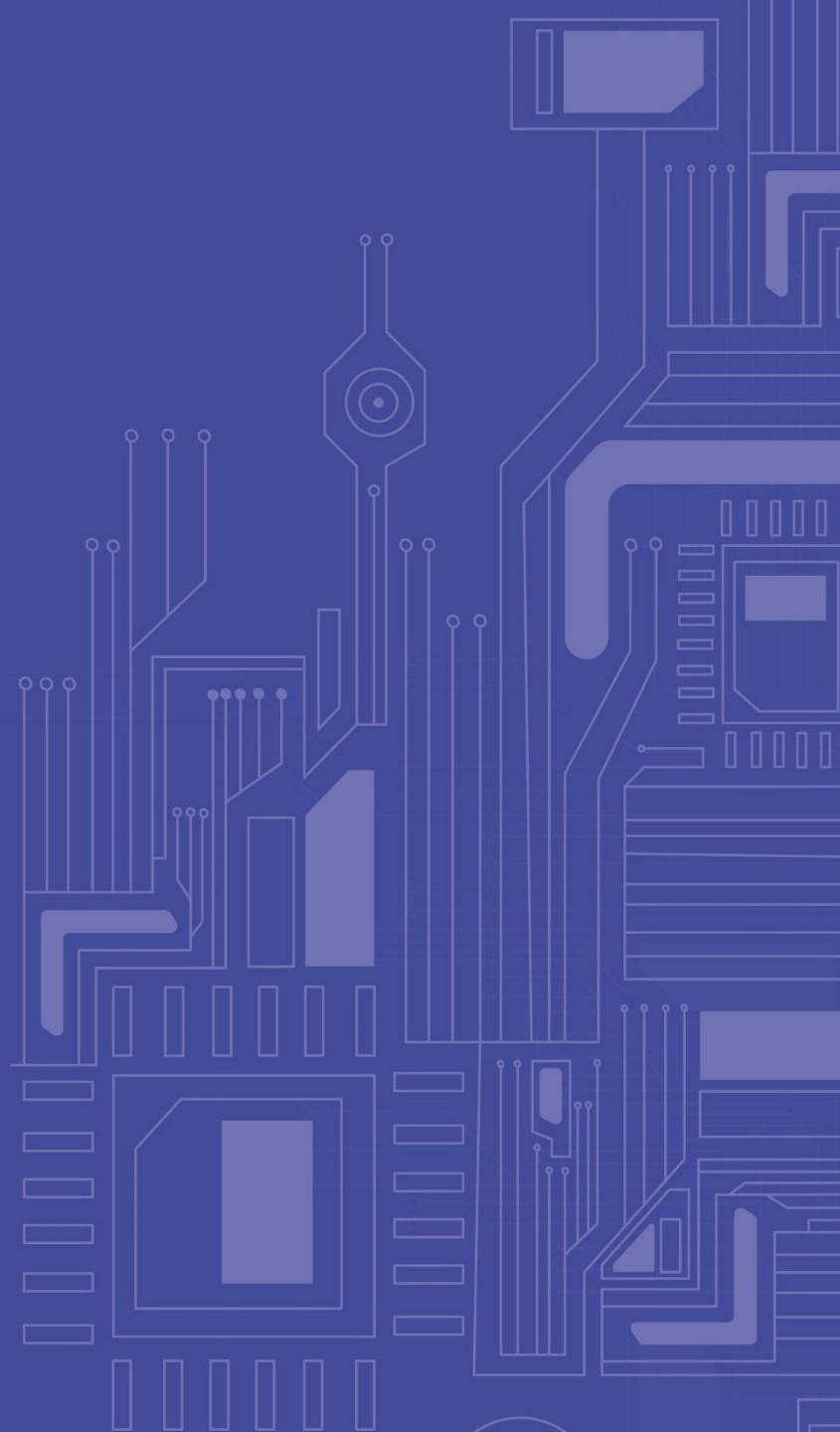
ЖУРНАЛ

- Журнал обычно представляет собой чисто последовательный файл с записями переменного размера, которые можно просматривать в прямом или обратном порядке. Обмены производятся стандартными порциями (страницами) с использованием буфера оперативной памяти. В грамотно организованных системах структура (и тем более, смысл) журнальных записей известна только компонентам СУБД, ответственным за журнализацию и восстановление.
- Поскольку содержимое журнала является критичным при восстановлении базы данных после сбоев, к ведению файла журнала предъявляются особые требования по части надежности. В частности, обычно стремятся поддерживать две идентичные копии журнала на разных устройствах внешней памяти.

СЛУЖЕБНАЯ ИНФОРМАЦИЯ

- Для корректной работы подсистемы управления данными во внешней памяти необходимо поддерживать информацию, которая используется только этой подсистемой и не видна подсистеме языкового уровня.
- Набор структур служебной информации зависит от общей организации системы, но обычно требуется поддержание следующих служебных данных:
  - Внутренние каталоги, описывающие физические свойства объектов базы данных, например, число атрибутов таблицы, их размер и, возможно, типы данных; описание индексов, определенных для данной таблицы и т.д.
  - Описатели свободной и занятой памяти в страницах данных. Такая информация требуется для нахождения свободного места при занесении кортежа. Отдельно приходится решать задачу поиска свободного места в случаях некластеризованных и кластеризованных таблиц (в последнем случае приходится дополнительно использовать кластеризованный индекс). Как уже отмечалось, нетривиальной является проблема освобождения страницы в условиях мультидоступа.
  - Связывание страниц одной таблицы. Если в одном файле внешней памяти могут располагаться страницы нескольких таблиц (обычно к этому стремятся), то нужно каким-то образом связать страницы одной таблицы. Тривиальный способ использования прямых ссылок между страницами часто приводит к затруднениями при синхронизации транзакций (например, особенно трудно освобождать и заводить новые страницы таблицы). Поэтому стараются использовать косвенное связывание страниц с использованием служебных индексов. В частности, известен общий механизм для описания свободной памяти и связывания страниц на основе B-деревьев.

ТРАНЗАКЦИИ И ИХ ХАРАКТЕРИСТИКИ



ОПРЕДЕЛЕНИЕ ТРАНЗАКЦИИ ЧЕРЕЗ ACID

Транзакция - последовательность операций над базой данных, обладающая следующими свойствами.

- ***Атомарность (Atomicy).*** Это свойство означает, что результаты всех операций, успешно выполненных в пределах транзакции, должны быть отражены в состоянии базы данных, либо в состоянии базы данных не должно быть отражено действие ни одной операции (конечно, здесь речь идет об операциях, изменяющих состояние базы данных).
- ***Согласованность (Consistency).*** В классическом смысле это свойство означает, что транзакция может быть успешно завершена с *фиксацией* результатов своих операций только в том случае, когда действия операций не нарушают *целостность* базы данных, т.е. удовлетворяют набору ограничений целостности, определенных для этой базы данных. Это свойство расширяется тем, что во время выполнения транзакции разрешается устанавливать точки согласованности и явным образом проверять ограничения целостности.
- ***Изоляция (Isolation).*** Требуется, чтобы две одновременно (параллельно или квазипараллельно) выполняемые транзакции никоим образом не действовали одна на другую. Другими словами, результаты выполнения операций транзакции *T1* не должны быть видны никакой другой транзакции *T2* до тех пор, пока транзакция *T1* не завершится успешным образом.
- ***Долговечность (Durability).*** После успешного завершения транзакции все изменения, которые были внесены в состояние базы данных операциями этой транзакции, должны гарантированно сохраняться, даже в случае сбоев аппаратуры или программного обеспечения.

АТОМАРНОСТЬ ТРАНЗАКЦИЙ

В этом смысле под транзакцией понимается неделимая с точки зрения воздействия на БД последовательность операторов манипулирования данными (чтения, удаления, вставки, модификации), такая, что либо результаты всех операторов, входящих в транзакцию, отображаются в состоянии базы данных, либо воздействие всех этих операторов полностью отсутствует.

Лозунгом транзакции является «Все или ничего»: при завершении транзакции оператором COMMIT (высокоуровневый аналог операции END TRANSACTION в интерфейсе RSS) результаты гарантированно фиксируются во внешней памяти (смысл термина *commit* состоит в запросе «фиксации» результатов транзакции); при завершении транзакции оператором ROLLBACK (высокоуровневый аналог операции RESTORE в интерфейсе RSS) результаты гарантированно отсутствуют во внешней памяти (смысл термина rollback состоит в запросе ликвидации результатов транзакции).

ТРАНЗАКЦИИ И ЦЕЛОСТНОСТЬ ДАННЫХ (1/2)

1. Есть транзакции несовместимые с целостностью (добавление нового сотрудника пока таблица с информацией об отделах не меняет данные о численности).
2. Поэтому для поддержки подобных ограничений целостности допускается их нарушение внутри транзакции с тем условием, чтобы к моменту завершения транзакции условия целостности были соблюдены. В системах с развитыми средствами ограничения и контроля целостности каждая транзакция начинается при целостном состоянии базы данных и должна оставить это состояние целостными после своего завершения.
3. Немедленно проверяемые ограничения целостности (операции)
4. Откладываемые ограничения целостности – это ограничения на базу данных, а не на какие-либо отдельные операции.

ТРАНЗАКЦИИ И ЦЕЛОСТНОСТЬ ДАННЫХ (2/2)

1. Никакая транзакция не может быть зафиксирована, если ее действия нарушили целостность базы данных. Однако в этом подходе имеются два серьезных дефекта:
   - Без точек сохранения вылетит на COMMIT
   - Чем длиннее транзакция, тем больше ограничений целостности в конце проверять
2. Дейт и Дарвен в 3м манифесте предложили отказаться от откладываемых ограничений целостности базы данных, а вместо этого ввести составные операторы изменения базы данных (нечто наподобие блоков BEGIN … END, поддерживаемых в языках программирования). После выполнения каждого такого блока (или отдельного оператора изменения базы данных, используемого без операторов начала и конца блока) база данных должна находится в целостном состоянии. Если составной оператор нарушает ограничение целостности, то он целиком отвергается, и вырабатывается соответствующий код ошибки. Транзакция в этом случае не откатывается. Понятно, что при использовании такого подхода при выполнении оператора COMMIT не требуется проверять ограничения целостности, и каждая зафиксированная транзакция будет оставлять базу данных в целостном состоянии.
3. Для реализации описанного подхода не требуются какие-либо новые механизмы, кроме точек сохранения транзакции, насильственной проверки ограничений целостности и частичных откатов транзакций, а отмеченные ранее проблемы снимаются. К сожалению, неизвестно, применяется ли этот подход на практике.

ИЗОЛИРОВАННОСТЬ ТРАНЗАКЦИЙ (1/5)

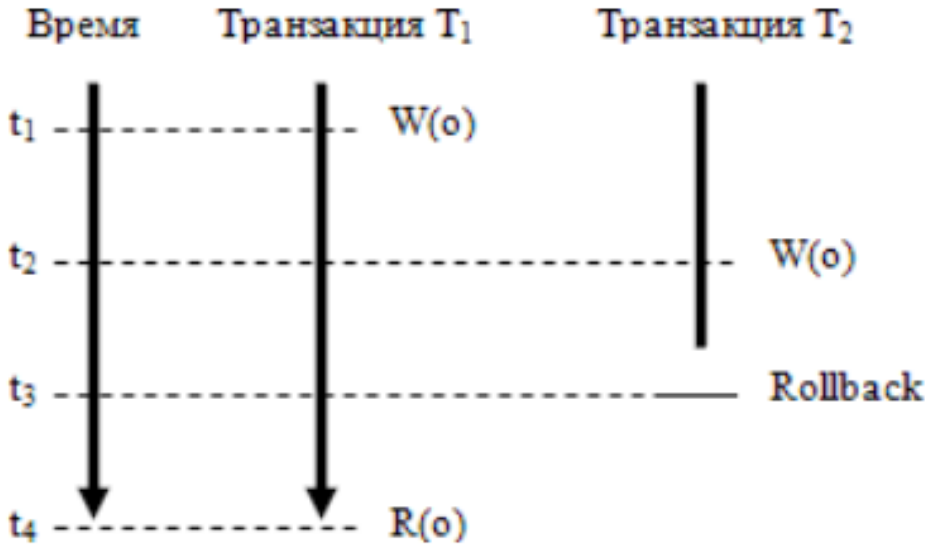
1. В многопользовательских системах с одной базой данных одновременно может работать несколько пользователей или прикладных программ. Предельной задачей системы является обеспечение изолированности пользователей, т.е. создание достоверной и надежной иллюзии того, что каждый из пользователей работает с базой данных в одиночку.
2. В связи со свойством сохранения целостности базы данных транзакции являются подходящими единицами изолированности пользователей. Действительно, если с каждым сеансом работы пользователя или приложений с базой данных ассоциируется транзакция, то каждый пользователь начинает работу с согласованным состоянием базы данных, т.е. с таким состоянием, в котором база данных могла бы находиться, даже если бы пользователь работал с ней в одиночку.
3. При соблюдении обязательного требования поддержки целостности базы данных возможно наличие нескольких уровней изолированности транзакций.

ИЗОЛИРОВАННОСТЬ ТРАНЗАКЦИЙ (2/5)

ПЕРВЫЙ УРОВЕНЬ ИЗОЛЯЦИИ – ОТСУТСТВИЕ ПОТЕРЯННЫХ ИЗМЕНЕНИЙ

В момент времени t_1 транзакция T_1 изменяет объект базы данных o (выполняет операцию W(o)). До завершения транзакции T_1 в момент времени $t_2 > t_1$ транзакция T_2 также изменяет объект o. В момент времени $t_3 > t_2$ транзакция T_2 завершается оператором ROLLBACK (например, по причине нарушения ограничений целостности).

Тогда при повторном чтении объекта o (выполнении операции R(o)) в момент времени $t_4 > t_3$ транзакция T_1 не видит своих изменений этого объекта, произведенных ранее (в частности, из-за этого может не удастся фиксация этой транзакции, что, возможно, повлечет потерю изменений у еще одной транзакции и т.д.).

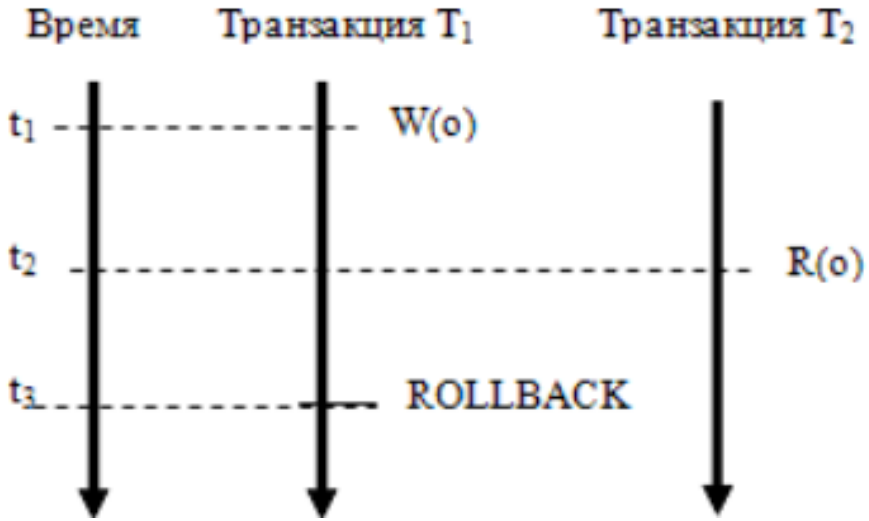


Такая ситуация называется ситуацией ***потерянных изменений***. Естественно, она противоречит требованию изолированности пользователей. Чтобы избежать такой ситуации в транзакции T_1 требуется, чтобы до завершения транзакции T_1 никакая другая транзакция не могла изменять никакой измененный транзакцией T_1 объект o (в частности, достаточно заблокировать доступ по изменению к объекту o до завершения транзакции T_1).

ИЗОЛИРОВАННОСТЬ ТРАНЗАКЦИЙ (3/5)

ВТОРОЙ УРОВЕНЬ ИЗОЛЯЦИИ – ОТСУТСТВИЕ ЧТЕНИЯ «ГРЯЗНЫХ» ДАННЫХ

В момент времени t_1 транзакция T_1 изменяет объект базы данных o (выполняет операцию W(o)). В момент времени $t_2 > t_1$ транзакция T_2 читает объект o (выполняет операцию R(o)). Поскольку транзакция T_1 еще не завершена, транзакция T_2 видит несогласованные *«грязные» данные*. В частности, в момент времени $t_3 > t_2$ транзакция T_1 может завершиться откатом (например, по причине нарушения ограничений целостности).

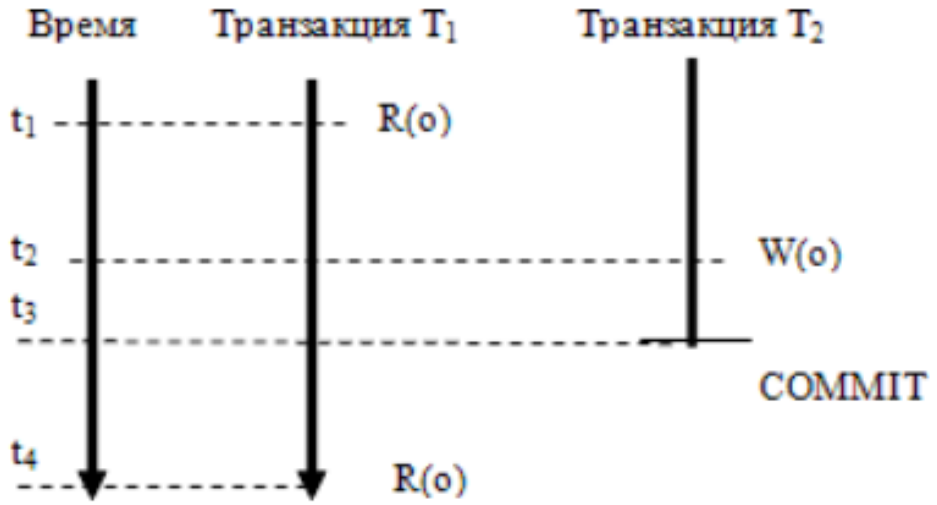


Эта ситуация тоже не соответствует требованию изолированности пользователей (каждый пользователь начинает свою транзакцию при согласованном состоянии базы данных и имеет право видеть только согласованные данные). Чтобы избежать ситуации чтения "грязных" данных, до завершения транзакции T_1, изменившей объект базы данных o, никакая другая транзакция не должна читать объект o (например, достаточно заблокировать доступ по чтению к объекту o до завершения изменившей его транзакции T_1).

ИЗОЛИРОВАННОСТЬ ТРАНЗАКЦИЙ (4/5)

ТРЕТИЙ УРОВЕНЬ ИЗОЛЯЦИИ – ОТСУТСТВИЕ НЕПОВТОРЯЮЩИХСЯ ЧТЕНИЙ

В момент времени t_1 транзакция T_1 читает объект базы данных o (выполняет операцию R(o)). До завершения транзакции T_1 в момент времени $t_2 > t_1$ транзакция T_2 изменяет объект o (выполняет операцию W(o)) и успешно завершается оператором COMMIT. В момент времени $t_3 > t_2$ транзакция T_1 повторно читает объект o и видит его измененное состояние.



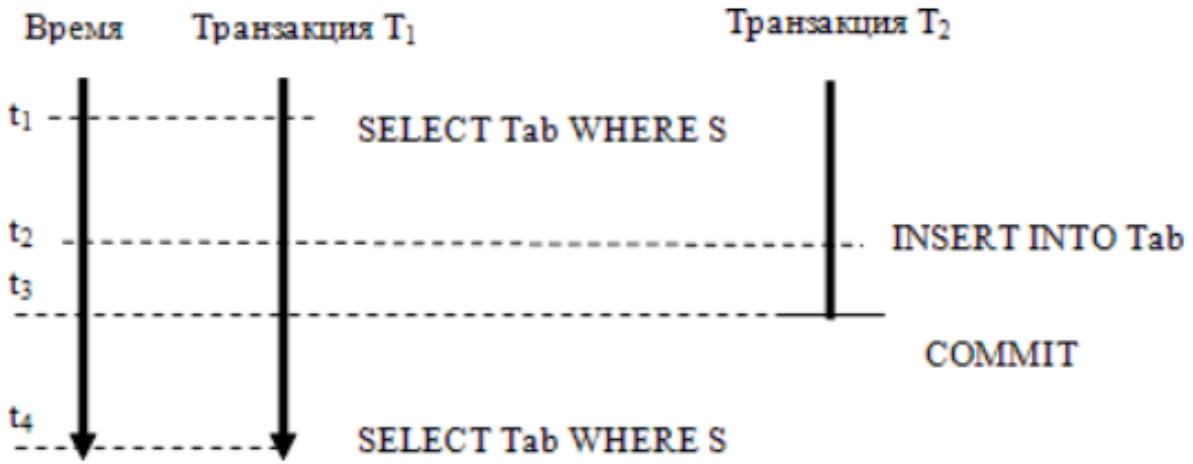
Чтобы избежать ***неповторяющихся чтений***, до завершения транзакции T_1 никакая другая транзакция не должна изменять объект o (для этого достаточно заблокировать доступ по записи к объекту o до завершения транзакции T_1). Часто это является максимальным требованием к средствам обеспечения изолированности транзакций, хотя, как будет видно немного позже, отсутствие неповторяющихся чтений еще не гарантирует реальной изолированности пользователей.

Существует возможность обеспечения разных уровней изолированности для разных транзакций, выполняющихся в одной системе баз данных (соответствующие операторы были предусмотрены уже в стандарте SQL:1992). Для корректного соблюдения ограничений целостности достаточен первый уровень. При этом удается существенно сократить накладные расходы СУБД и повысить общую эффективность.

ИЗОЛИРОВАННОСТЬ ТРАНЗАКЦИЙ (5/5)

ПРОБЛЕМА ФАНТОМОВ

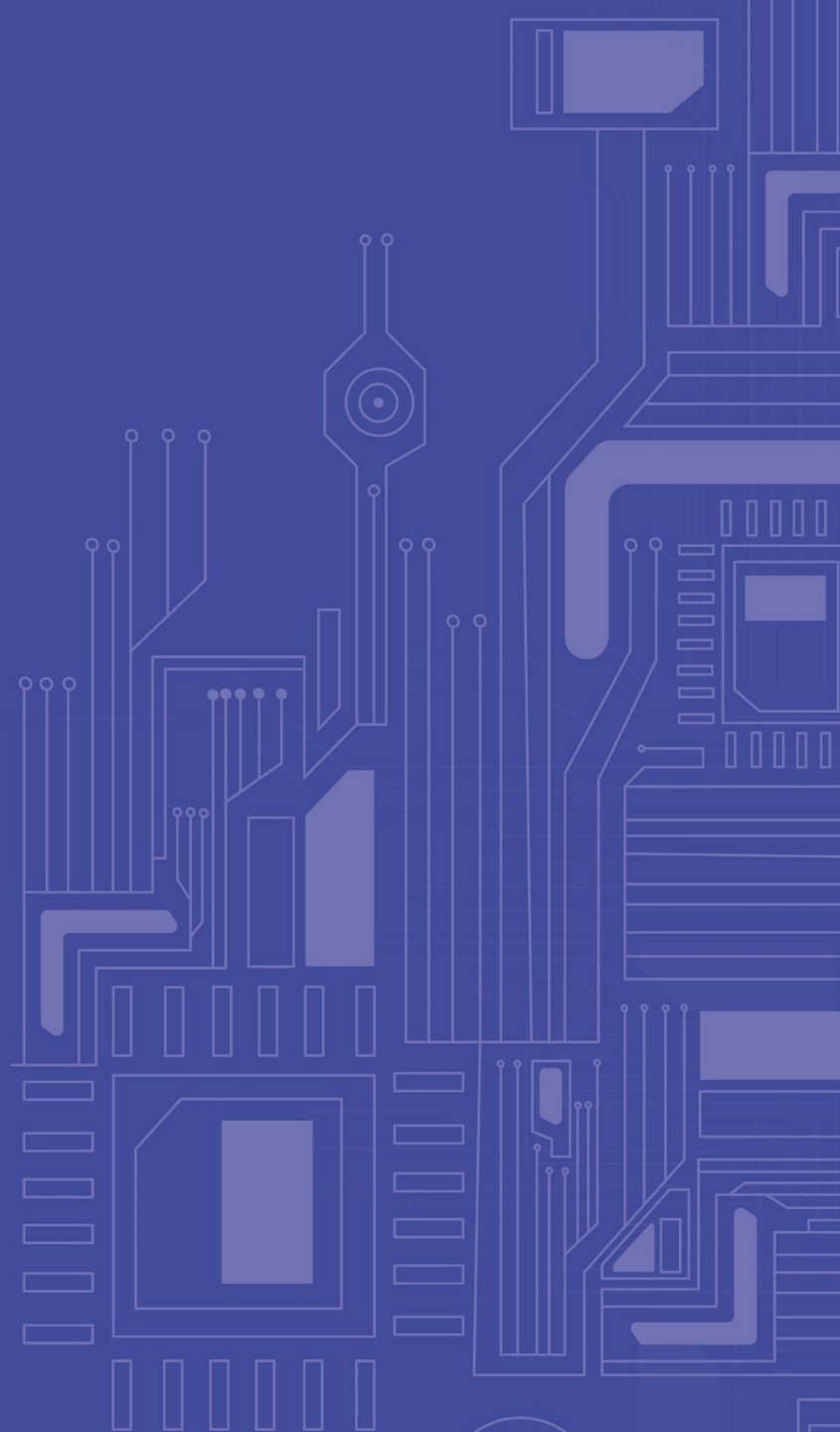
В момент времени t_1 транзакция T_1 выполняет оператор выборки кортежей таблицы Tab с условием выборки S (т.е. выбирается часть кортежей таблицы Tab, удовлетворяющих условию S). До завершения транзакции T_1 в момент времени $t_2 > t_1$ транзакция T_2 вставляет в таблицу Tab новый кортеж r, удовлетворяющий условию S, и успешно завершается.



В момент времени $t_3 > t_2$ транзакция T_1 повторно выполняет тот же оператор выборки, и в результате появляется кортеж, который отсутствовал при первом выполнении оператора.

Конечно, такая ситуация противоречит идее изолированности транзакций и может возникнуть даже на третьем уровне изолированности транзакций. Чтобы избежать появления кортежей-фантомов, требуется более высокий «логический» уровень изоляции транзакций. Идеи требуемого механизма (предикатные синхронизационные блокировки) появились также еще во время выполнения проекта System R, но в большинстве систем не реализованы.

МЕТОДЫ СЕРИАЛИЗАЦИИ ТРАНЗАКЦИЙ



СЕРИАЛИЗАЦИЯ ТРАНЗАКЦИЙ

Сериализация транзакций – это механизм их выполнения по некоторому сериальному плану. Обеспечение такого механизма является основной функцией компонента СУБД, ответственного за управление транзакциями. Система, в которой поддерживается сериализация транзакций, обеспечивает реальную изолированность пользователей.

Можно работать последовательно, можно читать параллельно, но можно и что-то делать не мешая друг другу

ВИДЫ КОНФЛИКТОВ ТРАНЗАКЦИЙ

Между транзакциями T_1 и T_2 могут существовать следующие виды конфликтов:

- W/W – транзакция T_2 пытается изменять объект, измененный не закончившейся транзакцией T_1 (наличие такого конфликта может привести к возникновению ситуации потерянных изменений);
- R/W – транзакция T_2 пытается изменять объект, прочитанный не закончившейся транзакцией T_1 (наличие такого конфликта может привести к возникновению ситуации неповторяющихся чтений);
- W/R – транзакция T_2 пытается читать объект, измененный не закончившейся транзакцией T_1 (наличие такого конфликта может привести к возникновению ситуации «грязного» чтения).

Практические методы сериализации транзакций основываются на учете этих конфликтов.

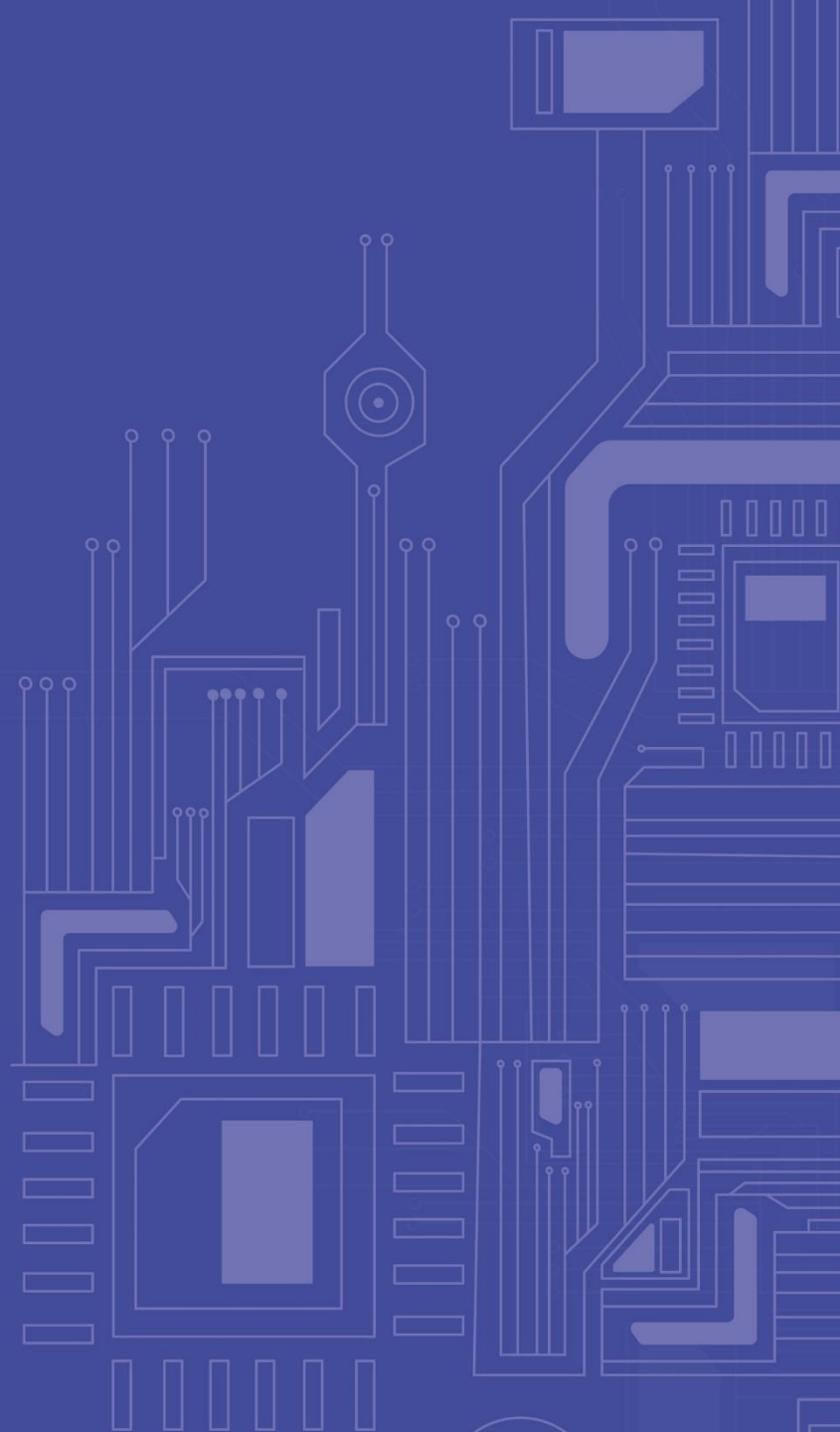
МЕТОДЫ СЕРИАЛИЗАЦИИ ДАННЫХ

Существуют два базовых подхода к сериализации транзакций – основанный на синхронизационных захватах объектов базы данных и на использовании временных меток. Суть обоих подходов состоит в обнаружении конфликтов транзакций и их устранении.

Для каждого из подходов имеются две разновидности – пессимистическая и оптимистическая. При применении пессимистических методов, ориентированных на ситуации, когда конфликты возникают часто, конфликты распознаются и разрешаются немедленно при их возникновении. Оптимистические методы основываются на том, что результаты всех операций модификации базы данных сохраняются в рабочей памяти транзакций. Реальная модификация базы данных производится только на стадии фиксации транзакции. Тогда же проверяется, не возникают ли конфликты с другими транзакциями.

Далее мы ограничимся рассмотрением более распространенных пессимистических разновидностей методов сериализации транзакций. Пессимистические методы сравнительно просто трансформируются в свои оптимистические варианты.

СИНХРОНИЗАЦИОННЫЕ БЛОКИРОВКИ



РЕЖИМЫ СИНХРОНИЗАЦИОННЫХ БЛОКИРОВОК

Наиболее распространенным в централизованных СУБД (включающих системы, основанные на архитектуре «клиент-сервер») является подход, основанный на соблюдении двухфазного протокола синхронизационных захватов объектов баз данных (Two-Phase Locking Protocol, 2PL). В общих чертах подход состоит в том, что перед выполнением любой операции в транзакции T над объектом базы данных *o* от имени транзакции T запрашивается синхронизационная блокировка объекта *o* в соответствующем режиме (в зависимости от вида операции).

Основными режимами синхронизационных блокировок являются следующие:

- совместный режим – S (Shared), означающий совместную (по чтению) блокировку объекта и требуемый для выполнения операции чтения объекта;
- монопольный режим – X (eXclusive), означающий монопольную (по записи) блокировку объекта и требуемый для выполнения операций вставки, удаления и модификации объекта.

СОВМЕСТИМОСТЬ БЛОКИРОВОК S И X

В первом столбце приведены возможные состояния объекта с точки зрения синхронизационных захватов. При этом "-" соответствует состоянию объекта, для которого не установлен никакой захват. Транзакция, запросившая синхронизационный захват объекта БД, уже захваченный другой транзакцией в несовместимом режиме, блокируется до тех пор, пока захват с этого объекта не будет снят.

	X	S
-	да	да
X	нет	нет
S	нет	да

2PL И ОБЪЕКТЫ БЛОКИРОВОК

Для обеспечения сериализации транзакций (третьего уровня изолированности) синхронизационные блокировки объектов, произведенные по инициативе транзакции, можно снимать только при ее завершении.

Это требование порождает двухфазный протокол синхронизационных захватов – **2PL**. В соответствии с этим протоколом выполнение транзакции разбивается на две фазы:

- первая фаза транзакции (**выполнение операций над базой данных**) – накопление блокировок;
- вторая фаза (**фиксация или откат**) – снятие блокировок.

ОБЪЕКТЫ СИНХРОНИЗАЦИОННОГО ЗАХВАТА

- Файл (сегмент в терминах System R) – физический (с точки зрения базы данных) объект, область хранения нескольких таблиц и, возможно, индексов;
- Таблица – логический объект, соответствующий множеству кортежей данной таблицы;
- Страница данных – физический объект, хранящий кортежи одной или нескольких таблиц, индексную или служебную информацию;
- Кортеж – элементарный физический объект базы данных.

Начинали все со страниц.
Меньше объект –больше блокировок – больше ресурсов потратили – меньше конфликтов.
Сейчас, в основном, на уровне кортежей блокировки реализуются.

ГРАНУЛИРОВАННЫЕ СИНХРОНИЗАЦИОННЫЕ БЛОКИРОВКИ (1/2)

При применении этого подхода синхронизационные блокировки могут запрашиваться по отношению к объектам разного уровня: файлам, таблицам и кортежам. Требуемый уровень объекта определяется тем, какая операция выполняется (например, для выполнения операции уничтожения таблицы объектом синхронизационной блокировки должна быть вся таблица, а для выполнения операции удаления кортежа – этот кортеж). Объект любого уровня может быть заблокирован в режиме S или X.

Для согласования блокировок разного уровня вводятся специальный протокол гранулированных блокировок и новые типы блокировок. Перед установкой блокировки на некоторый объект базы данных в режиме S или X соответствующий объект верхнего уровня должен быть заблокирован в режиме IS, IX или SIX.

IS

Блокировка в режиме **IS (Intented for Shared lock)** некоторого составного объекта ***o*** базы данных означает намерение заблокировать некоторый объект ***o'***, входящий в ***o***, в совместном режиме (режиме S). Например, при намерении читать кортежи из таблицы *Tab* эта таблица должна быть заблокирована в режиме IS (а до этого в таком же режиме должен быть заблокирован файл, в котором располагается таблица *Tab*).

IX

Блокировка в режиме **IX (Intented for eXclusive lock)** некоторого составного объекта o базы данных означает намерение заблокировать некоторый объект o', входящий в o, в монопольном режиме (режиме X). Например, для удаления кортежей из таблицы Tab эта таблица должна быть заблокирована в режиме IX (а до этого в таком же режиме должен быть заблокирован файл, в котором располагается таблица Tab).

ГРАНУЛИРОВАННЫЕ СИНХРОНИЗАЦИОННЫЕ БЛОКИРОВКИ (2/2)

SIX

Блокировка в режиме **SIX (Shared, Intented for eXclusive lock)** некоторого составного объекта ***o*** базы данных означает совместную блокировку всего этого объекта с намерением впоследствии блокировать какие-либо входящие в него объекты в монопольном режиме (режиме X). Например, если выполняется длинная операция просмотра таблицы *Tab* с возможностью удаления некоторых просматриваемых кортежей, то экономичнее всего заблокировать таблицу *Tab* в режиме SIX (а до этого заблокировать в режиме IS файл, в котором располагается таблица Tab).

СОВМЕСТИМОСТЬ СИНХРОНИЗАЦИОННЫХ БЛОКИРОВОК

	X	S	IX	IS	SIX
-	да	да	да	да	да
X	нет	нет	нет	нет	нет
S	нет	да	нет	да	нет
IX	нет	нет	да	да	нет
IS	нет	да	да	да	да
SIX	нет	нет	нет	да	нет

Для атомарных объектов разумны только блокировки в режимах S и X. Пусть теперь *o* – это некоторый составной объект. Пример:

Блокировка объекта o в режиме X в транзакции T1 направлена на то, чтобы изменять объект o целиком. Несовместимость блокировки объекта o в режиме X в транзакции T1 с его блокировкой в режиме X или IX в транзакции T2 устраняет конфликты транзакций T1 и T2 вида W/W. Несовместимость блокировки объекта o в режиме X в транзакции T1 с его блокировкой в режиме S или IS в транзакции T2 устраняет конфликты транзакций T1 и T2 вида W/R. Наконец, несовместимость блокировки объекта o в режиме X в транзакции T1 с его блокировкой в режиме SIX в транзакции T2 устраняет конфликты транзакций T1 и T2 вида W/R и W/W.

ПРЕДИКАТНЫЕ СИНХРОНИЗАЦИОННЫЕ БЛОКИРОВКИ

Любая операция над реляционной базой данных задается некоторым условием (т.е. в ней указывается не конкретный набор объектов базы данных, над которыми нужно выполнить операцию, а условие, которому должны удовлетворять объекты этого набора).

Для решения **проблемы фантомов** необходимо перейти от блокировок индивидуальных («физических») объектов базы данных, к блокировке условий (**предикатов**), которым удовлетворяют эти объекты.

Проблема фантомов не возникает при использовании для блокировок уровня таблиц именно потому, что таблица как логический объект представляет собой неявное условие для входящих в него кортежей. Блокировка таблицы – это простой и частный случай предикатной блокировки.

В System R блокировки сегментов (файлов), таблиц и кортежей технически трактовались единообразно, как блокировки идентификаторов кортежей (tid'ов).

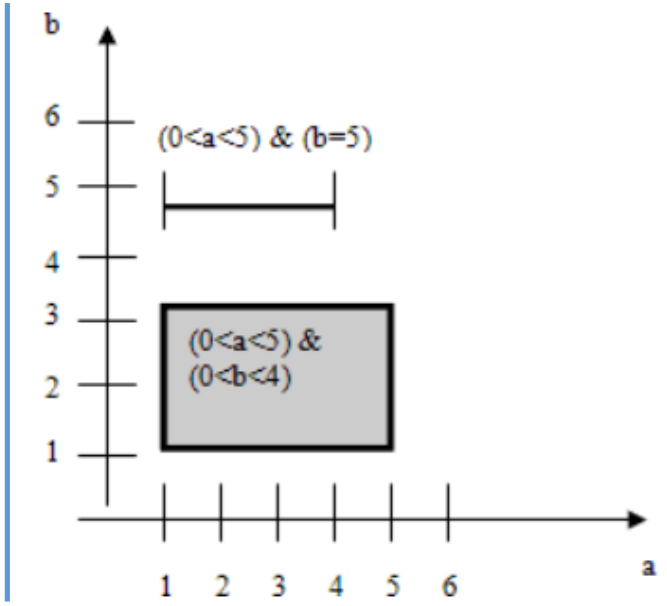
Предлагалось расширить систему синхронизации, разрешив применять блокировки к паре «идентификатор индекса, интервал значений ключа этого индекса». К такой паре можно было применять блокировки в любом из допустимых режимов, причем две такие блокировки считались совместимыми в том и только в том случае, если они были совместимы в соответствии с таблицей совместимости или указанные диапазоны значений ключей не пересекались. Этот вариант решения не работает при полном сканировании, например.

СИСТЕМА БЛОКИРОВОК ПРОСТЫХ УСЛОВИЙ

Простое условие: имя_поля { = > < } значение

Сложный SQL – запрос компилируется в последовательность обращений к памяти на основе простых условий. Поэтому в случае типовой организации SQL-ориентированной СУБД простые условия можно использовать как основу предикатных захватов.

Пусть Tab – таблица с полями a_1, a_2, ..., a_n, а m_1, m_2, ..., m_n – множества допустимых значений a_1, a_2, ..., a_n соответственно (естественно, все эти множества – конечные). Тогда можно сопоставить Tab конечное *n*-мерное пространство возможных значений кортежей Tab. Легко видеть, что любое простое условие, представляющее собой конъюнкцию простых предикатов, «вырезает» в этом пространстве *k*-мерный прямоугольник ($k \le n$).

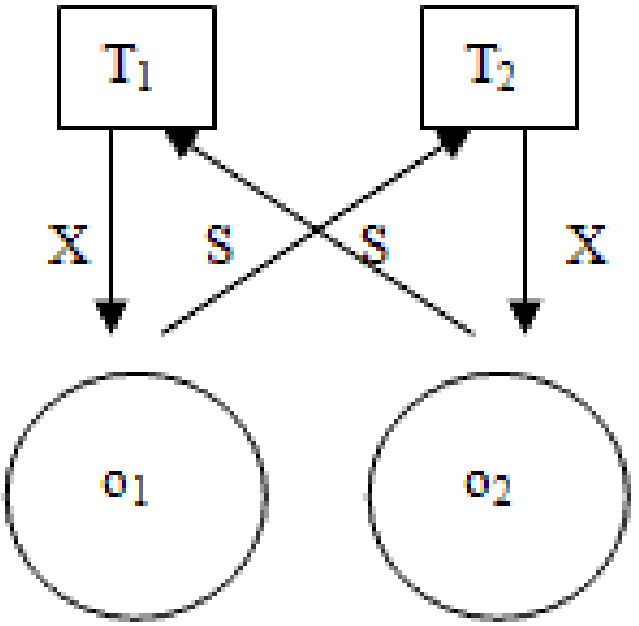


Пример.
В каких бы режимах не требовала транзакция T1 блокировки условия (0 < a < 5) & (b = 5), а транзакция T2 – блокировки условия (0 < a <6) & (0 < b <4), эти блокировки всегда будут совместимы.

Пусть имеются два простых условия $scond_1$ и $scond_2$. Пусть транзакция T_1 запрашивает блокировку $scond_1$, а транзакция T_2 – $scond_2$ в режимах, которые были бы несовместимы, если бы $scond_1$ и $scond_2$ являлись не условиями, а объектами базы данных (S-X, X-S, X-X). Эти блокировки совместимы в том и только в том случае, когда прямоугольники, соответствующие $scond_1$ и $scond_2$, не пересекаются.

СИНХРОНИЗАЦИОННЫЕ ТУПИКИ

Одним из наиболее чувствительных недостатков метода сериализации транзакций на основе синхронизационных блокировок является возможность возникновение тупиков (**deadlocks**) между транзакциями. Синхронизационные тупики возможны при применении любого из рассмотренных выше вариантов механизмов блокировок.



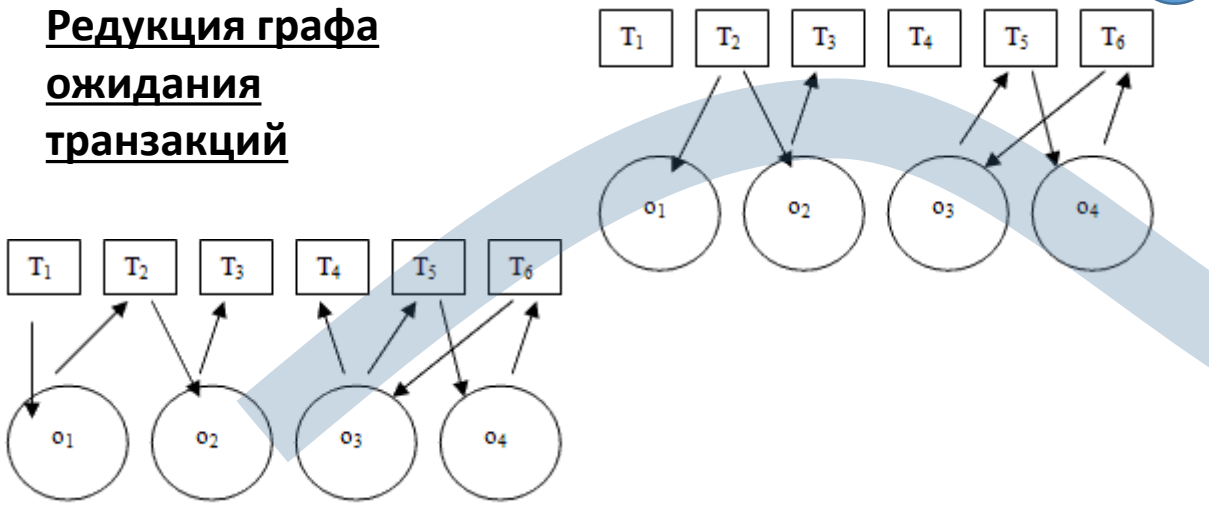
- Транзакции T1 и T2 устанавливают монопольные блокировки объектов o1 и o2 соответственно;
- после этого T1 требуется совместная блокировка объекта o2, а T2 – совместная блокировка объекта o1;
- ни одно из этих требований блокировки не может быть удовлетворено, следовательно, ни одна из транзакций не может продолжаться; поэтому монопольные блокировки объектов никогда не будут сняты, а требования совместных блокировок не будут удовлетворены.

Основой обнаружения тупиковых ситуаций является построение (или постоянное поддержание) графа ожидания транзакций. Граф ожидания транзакций – это ориентированный двудольный граф, в котором существует два типа вершин – вершины, соответствующие транзакциям (будем изображать их прямоугольниками), и вершины, соответствующие объектам блокировок (будем изображать их окружностями). В этом графе дуги соединяют только вершины-транзакции с вершинами-объектами.

ОБНАРУЖЕНИЕ ТУПИКОВ

Дуга из вершины-транзакции к вершине-объекту существует в том и только в том случае, если для этой транзакции имеется удовлетворенная блокировка данного объекта. Дуга из вершины-объекта к вершине-транзакции существует тогда и только тогда, когда эта транзакция ожидает удовлетворения запроса блокировки данного объекта. Легко показать, что в системе существует тупиковая ситуация в том и только в том случае, когда в графе ожидания транзакций имеется хотя бы один цикл.

Редукция графа ожидания транзакций



1 Удаляются все дуги, исходящие из вершин-транзакций, в которые не входят дуги из вершин-объектов. (Это основывается на том разумном предположении, что транзакции, не ожидающие удовлетворения запроса блокировок, могут успешно завершиться и освободить блокировки).

2 Удаляются дуги, входящие в вершины-транзакции, из которых не исходят, ведущие к вершинам-объектам (транзакции, ожидающие удовлетворения блокировок, но не удерживающие заблокированные объекты, не могут быть причиной тупика).

3 Для тех вершин-объектов, для которых не осталось входящих дуг, но существуют исходящие, ориентация одной из исходящих дуг (выбираемой произвольным образом) изменяется на противоположную (это моделирует удовлетворение запроса блокировки).

РАЗРУШЕНИЕ ТУПИКОВ

Нужно каким-то образом обеспечить возможность продолжения работы хотя бы для части транзакций, попавших в тупик. Разрушение тупика начинается с выбора в цикле транзакций так называемой транзакции-жертвы, т.е. транзакции, которой решено пожертвовать, чтобы обеспечить возможность продолжения работы других транзакций.

Выбор транзакции-жертвы

Обычно реализуют многофакторную оценку приоритета транзакции

Больше всего блокировок	?	Может, уже скоро сама отработает
Недавно началась	?	Еще ничего непонятно, может, быстрая, может, приоритетная
Случайная	?	Может от имени шефа быть
От имени младшего сотрудника	?	Ему теперь вообще не работать?

После выбора транзакции-жертвы выполняется откат этой транзакции, который может носить полный или частичный (до некоторой точки сохранения) характер. При этом, естественно, освобождаются блокировки, и может быть продолжено выполнение других транзакций. Естественно, такое насильственное устранение тупиковых ситуаций является нарушением принципа изолированности пользователей, которого невозможно избежать.

В централизованных системах стоимость построения графа ожидания сравнительно невелика, но она становится слишком большой в распределенных СУБД, в которых транзакции могут выполняться в разных узлах сети. Поэтому в таких системах обычно используются другие методы сериализации транзакций.

МЕТОД ВРЕМЕННЫХ МЕТОК

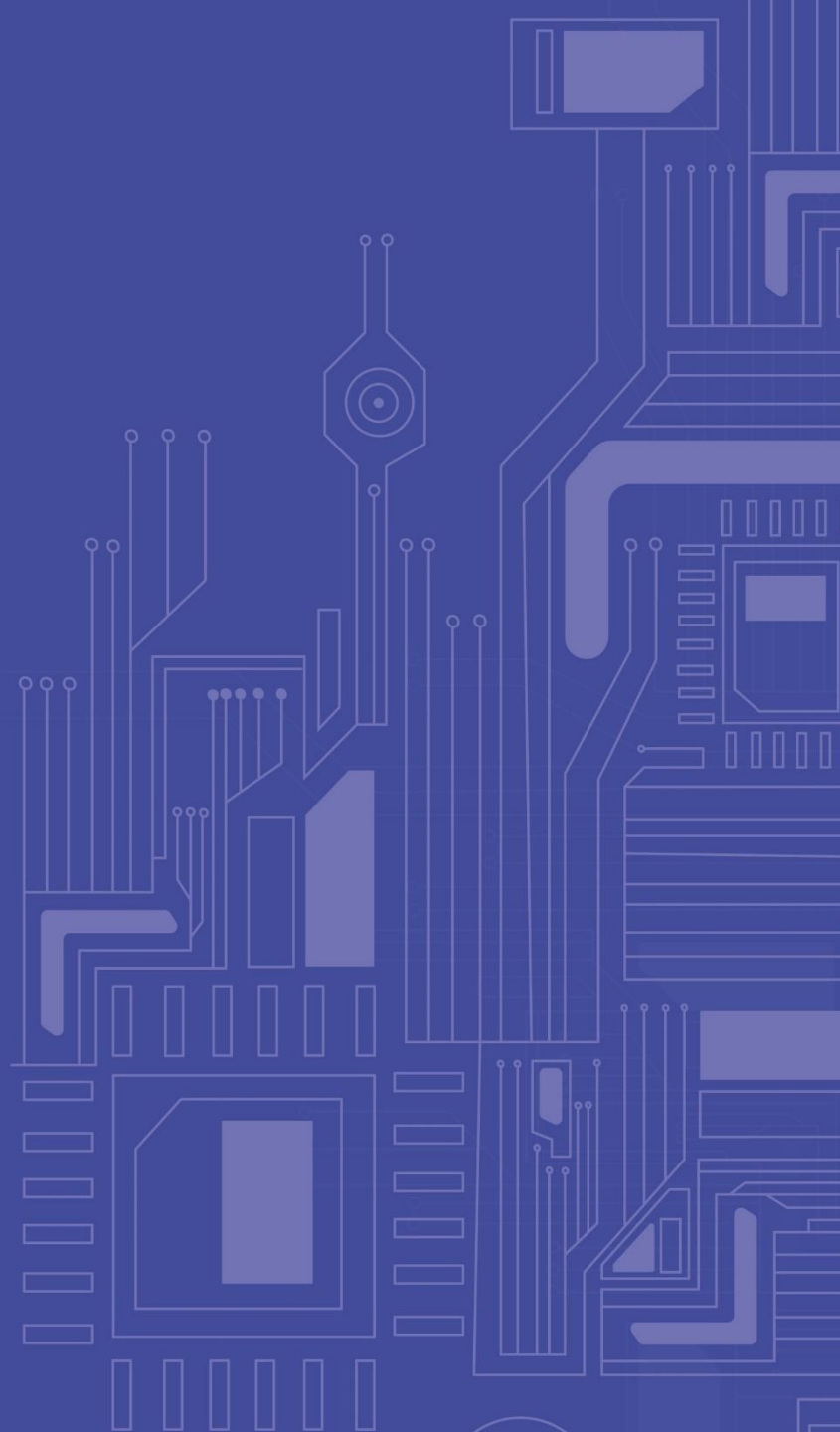
Основная идея метода временных меток (Timestamp Ordering, TO), у которого существует множество разновидностей, состоит в следующем: если транзакция T1 началась раньше транзакции T2, то система обеспечивает такой сериальный план, как если бы транзакция T1 была целиком выполнена до начала T2.

Для этого каждой транзакции T предписывается временная метка t(T), соответствующая времени начала выполнения транзакции T. При выполнении операции над объектом o транзакция T помечает его своими идентификатором, временной меткой и типом операции (чтение или изменение).
Перед выполнением операции над объектом o транзакция T2 выполняет следующие действия:

- Проверяет, помечен ли объект o какой-либо транзакцией T1. Если не помечен, то помечает этот объект своей временной меткой и типом операции и выполняет операцию. Конец действий.
- Иначе транзакция T2 проверяет, не завершилась ли транзакция T1, пометившая этот объект. Если транзакция T1 закончилась, то T2 помечает объект o и выполняет свою операцию. Конец действий.
- Если транзакция T1 не завершилась, то T2 проверяет конфликтность операций. Если операции неконфликтны, то при объекте o запоминается идентификатор транзакции T2, остается или проставляется временная метка с меньшим значением, и транзакция T2 выполняет свою операцию.
- Если операции транзакций T2 и T1 конфликтуют, то если t(T1) > t(T2) (т.е. транзакция T1 является более «молодой», чем T2), то производится откат T1 и всех других транзакций, идентификаторы которых сохранены при объекте o, и T2 выполняет свою операцию.
- Если же t(T1) < t(T2) (T1 «старше» T2), то производится откат T2; T2 получает новую временную метку и начинается заново.


ВСЁ!

СЕМИНАР



ХЕШИРОВАНИЕ

Хеширование (или *хэширование*, англ. *hashing*) – это преобразование входного массива данных определенного типа и произвольной длины в выходную битовую строку фиксированной длины. Такие преобразования также называются *хеш-функциями* или функциями *свертки*, а их результаты называют *хешем, хеш-кодом, хеш-таблицей* или *дайджестом* сообщения (англ. *message digest*).

Хеш-таблица – это *структура данных*, реализующая *интерфейс* ассоциативного массива, то есть она позволяет хранить пары вида "*ключ- значение*" и выполнять три *операции*: операцию добавления новой пары, операцию поиска и операцию удаления пары по ключу. Хеш-таблица является массивом, формируемым в определенном порядке *хеш-функцией*.

УСЛОВИЯ ХОРОШЕЙ ХЭШ-ФУНКЦИИ:

- функция должна быть простой с вычислительной точки зрения;
- функция должна распределять ключи в хеш-таблице наиболее равномерно;
- функция не должна отображать какую-либо связь между значениями ключей в связь между значениями адресов;
- функция должна минимизировать число *коллизий* – то есть ситуаций, когда разным ключам соответствует одно значение *хеш-функции* (ключи в этом случае называются *синонимами*).

СВОЙСТВА ХЭШ-ТАБЛИЦЫ

Если *хеш-функция* распределяет совокупность *возможных ключей* равномерно по множеству индексов, то *хеширование* эффективно разбивает множество ключей. Наихудший случай – когда все ключи хешируются в один *индекс.*

При возникновении *коллизий* необходимо найти новое *место* для хранения ключей, претендующих на одну и ту же ячейку хеш-таблицы. Причем, если *коллизии* допускаются, то их количество необходимо минимизировать.

Хеш-таблицы должны соответствовать следующим *свойствам:*

- Выполнение операции в хеш-таблице начинается с вычисления *хеш-функции* от ключа. Получающееся хеш-значение является индексом в исходном массиве.
- Количество хранимых элементов массива, деленное на число возможных значений *хеш-функции*, называется *коэффициентом заполнения хеш-таблицы* (*load factor*) и является важным параметром, от которого зависит среднее время выполнения операций.
- Операции поиска, вставки и удаления должны выполняться в среднем за время O(1). Однако при такой оценке не учитываются возможные аппаратные затраты на перестройку индекса хеш-таблицы, связанную с увеличением значения размера массива и добавлением в хеш-таблицу новой пары.
- Механизм разрешения *коллизий* является важной составляющей любой хеш-таблицы.

При этом первое свойство хорошей *хеш-функции* зависит от характеристик компьютера, а второе – от значений данных.

РАЗРЕШЕНИЕ КОЛЛИЗИЙ (1/3)

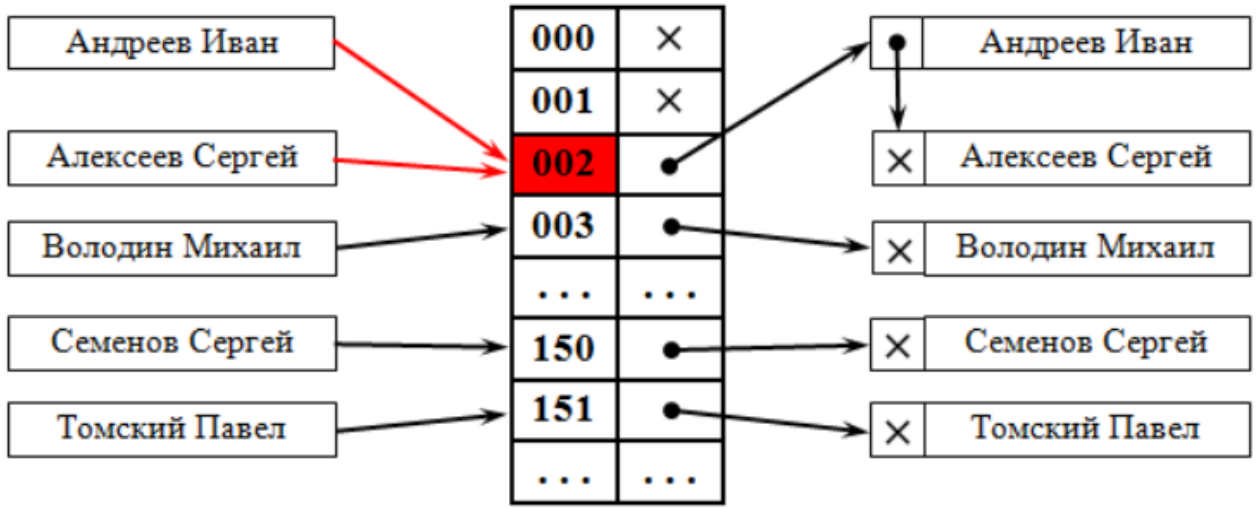
Коллизии осложняют использование хеш-таблиц, так как нарушают однозначность соответствия между хеш-кодами и данными.

ХЕШ-ТАБЛИЦЫ С ПРЯМОЙ АДРЕСАЦИЕЙ

В некоторых специальных случаях удается избежать *коллизий* вообще. Например, если все ключи элементов известны заранее (или очень редко меняются), то для них можно найти некоторую инъективную хеш-функцию, которая распределит их по ячейкам хеш-таблицы без *коллизий*. Хеш-таблицы, использующие подобные *хеш-функции*, не нуждаются в механизме разрешения *коллизий*, и называются хеш-таблицами с *прямой адресацией*.

МЕТОД ЦЕПОЧЕК

Технология сцепления элементов состоит в том, что *элементы множества*, которым соответствует одно и то же хеш-*значение*, связываются в цепочку-*список*. В позиции номер `i` хранится *указатель* на *голову списка* тех элементов, у которых хеш-*значение* ключа равно `i` ; если таких элементов в множестве нет, в позиции `i` записан `NULL`.



РАЗРЕШЕНИЕ КОЛЛИЗИЙ (2/3)

МЕТОД ЦЕПОЧЕК: КОЛЛИЗИИ

Каждая *ячейка* массива является указателем на связный *список* (цепочку) пар *ключ-значение*, соответствующих одному и тому же хеш-значению ключа. *Коллизии* просто приводят к тому, что появляются цепочки длиной более одного элемента.

МЕТОД ЦЕПОЧЕК: ПОИСК

Операции поиска или удаления данных требуют просмотра всех элементов соответствующей ему цепочки, чтобы найти в ней элемент с заданным ключом. Для добавления данных нужно добавить элемент в конец или начало соответствующего списка, и, в случае если коэффициент заполнения станет слишком велик, увеличить размер массива и перестроить таблицу.

МЕТОД ЦЕПОЧЕК: ВРЕМЯ ПОИСКА

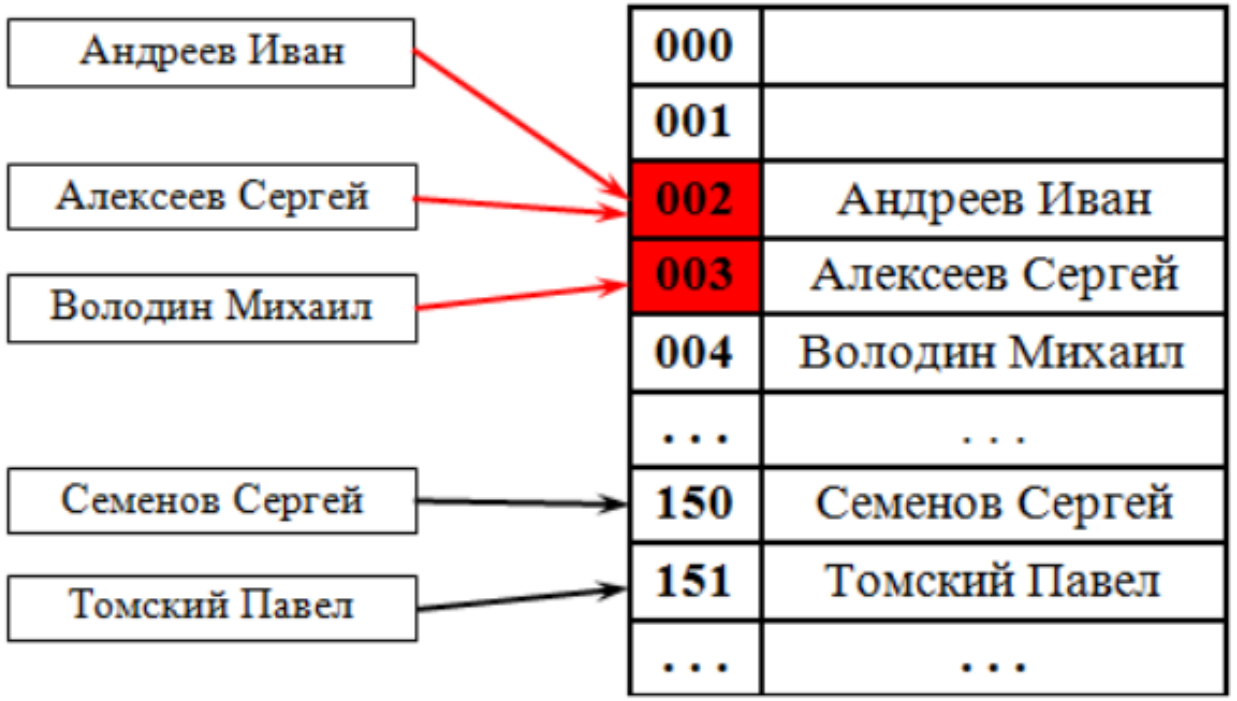
При предположении, что каждый элемент может попасть в любую позицию таблицы с равной вероятностью и независимо от того, куда попал любой другой элемент, *среднее время* работы *операции* поиска элемента составляет $O(1+k)$, где k – коэффициент заполнения таблицы.

РАЗРЕШЕНИЕ КОЛЛИЗИЙ (3/3)

МЕТОД ОТКЫТОЙ АДРЕСАЦИИ

В отличие от *хеширования* с цепочками, при открытой адресации никаких списков нет, а все записи хранятся в самой хеш-таблице.
Каждая *ячейка* таблицы содержит либо элемент динамического *множества*, либо `NULL`.

В этом случае, если *ячейка* с вычисленным индексом занята, то можно просто просматривать следующие записи таблицы по порядку до тех пор, пока не будет найден *ключ* `K` или пустая позиция в таблице.



На рисунке разрешение *коллизий* осуществляется методом открытой адресации. Два значения претендуют на *ключ* 002, для одного из них находится первое свободное (еще незанятое) *место* в таблице.