МИНОБРНАУКИ РОССИИ

Передовые инженерные школы

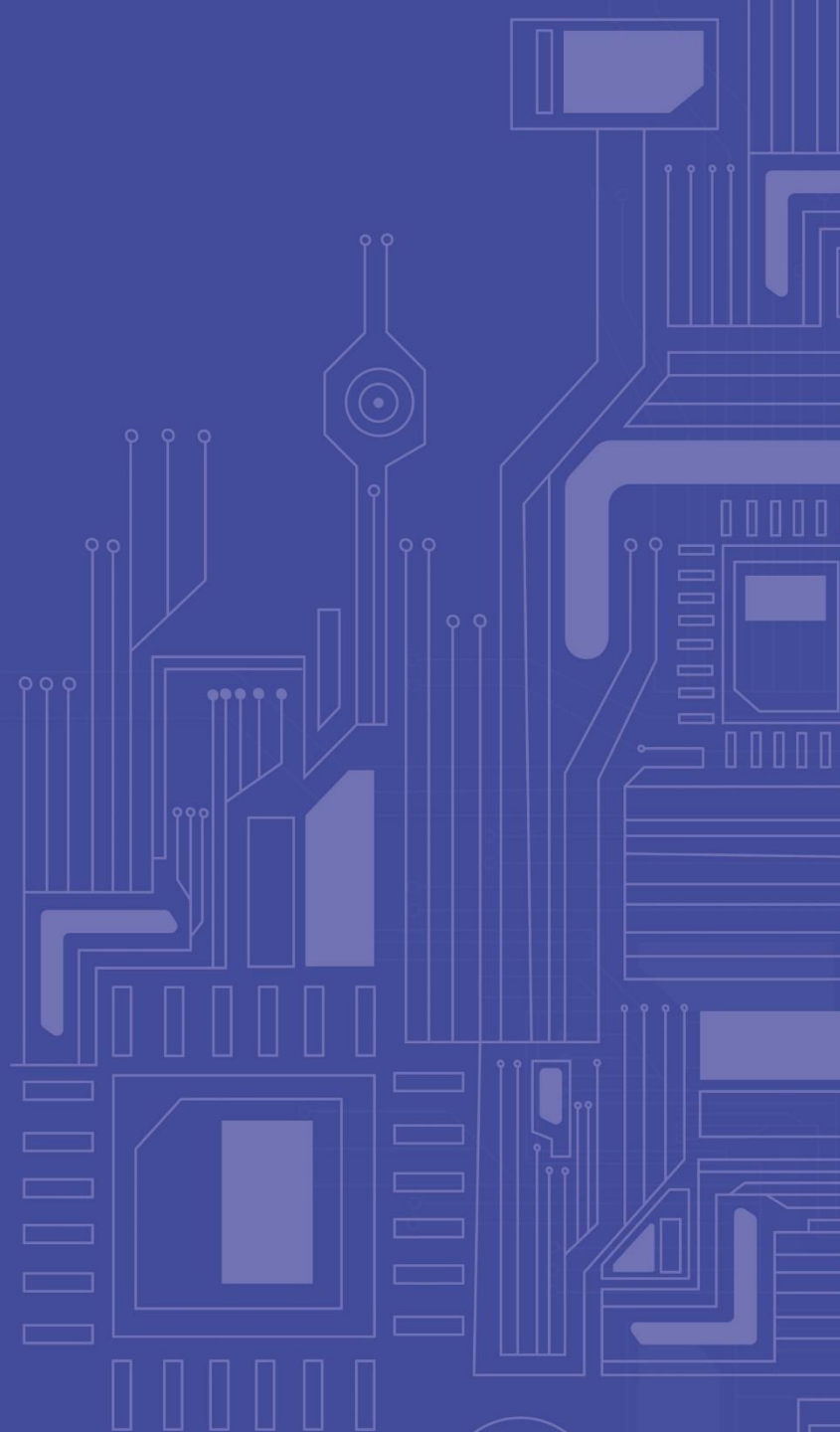
СИСТЕМЫ УПРАВЛЕНИЯ БАЗАМИ ДАННЫХ

Лекция 8

ПЛАН ЛЕКЦИИ

- Сериализация
- Журнализация
- Восстановление после сбоев
- Введение в SQL

СЕРИАЛИЗАЦИЯ НА ОСНОВЕ ВЕРСИОННОСТИ



СЕРИАЛИЗАЦИЯ НА ОСНОВЕ ВЕРСИОННОСТИ

Основная идея этих алгоритмов сериализации транзакций состоит в том, что в базе данных **допускается существование нескольких «версий» одного и того же объекта**. Эти алгоритмы, главным образом, направлены на преодоление конфликтов транзакций категорий R/W и W/R, позволяя выполнять операции чтения над некоторой предыдущей версией объекта базы данных. В результате операции чтения выполняются без задержек и тупиков, а также без некоторых откатов, возможных при применении метода временных меток.

Поддерживается Oracle и PostgreSQL

Версионный вариант алгоритма временных меток *(Multiversion Timestamp Ordering, MVTO)*

- Как и в простом методе временных меток, в алгоритме MVTO порядок выполнения операций одновременно выполняемых транзакций задается порядком временных меток, которые получают транзакции во время старта. Временные метки также используются для идентификации версий данных при чтении и модификации – каждая версия получает временную метку той транзакции, которая ее записала.

- Алгоритм MVTO не только следит за порядком выполнения операций транзакций, но также отвечает за трансформацию операций над объектами базы данных в операции над версиями этих объектов, т.е. каждая операция над объектом базы данных *o* преобразуется в соответствующую операцию над некоторой версией объекта *o*.

АЛГОРИТМ MVTO (1/2)

При описании алгоритма будем использовать следующие обозначения.

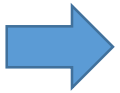
- Временную метку, полученную транзакцией T_i в начале ее работы, будем обозначать как $t(T_i)$.
- Операция чтения объекта базы данных *o*, выполняемая в транзакции T_i, будет обозначаться как $R_i(o)$.
- Для обозначения того, что транзакция T_i читает версию объекта базы данных *o*, созданную транзакцией T_k, будем использовать запись $R_i(o_k)$.
- Для обозначения того, что транзакция T_i записывает версию элемента данных *o*, будем использовать запись $W_i(o_i)$.

Алгоритм MVTO работает следующим образом.

1. Любая операция $R_i(o)$ преобразуется в операцию $R_i(o_k)$, где o_k – это версия объекта *o*, помеченная наибольшей временной меткой $t(T_k)$, такой что $t(T_k) \leq t(T_i)$. Другими словами, транзакции T_i для чтения дается версия объекта *o*, созданная транзакцией T_k, которая не моложе T_i, но старше любой другой транзакции T_n, создававшей свою версию объекта *o*.
2. При обработке операции $W_i(o)$ выполняются следующие действия:
   - если к этому времени некоторой незафиксированной транзакцией T_n уже выполнена некоторая операция $R_n(o_k)$, такая что $t(T_k) \leq t(T_i) < t(T_n)$, то операция $W_i(o)$ не выполняется, а транзакция T_i откатывается;
   - в противном случае $W_i(o)$ преобразуется в $W_i(o_i)$, т.е. образуется еще одна версия объекта *o*.

АЛГОРИТМ MVTO (2/2)

3. При откате любой транзакции уничтожаются все созданные ею версии объектов базы данных и откатываются все транзакции, прочитавшие хотя бы одну из этих версий. Тем самым, откаты транзакций могут быть «каскадными».
4. Выполнение операции фиксации транзакции T_i (COMMIT) откладывается до того момента, когда завершатся все транзакции, записавшие версии данных, прочитанные T_i. Легко видеть, что без соблюдения этого требования не соблюдалось бы свойство долговечности (durability) транзакций, поскольку при откате некоторых транзакций потребовалось бы откатывать и ранее зафиксированные транзакции.

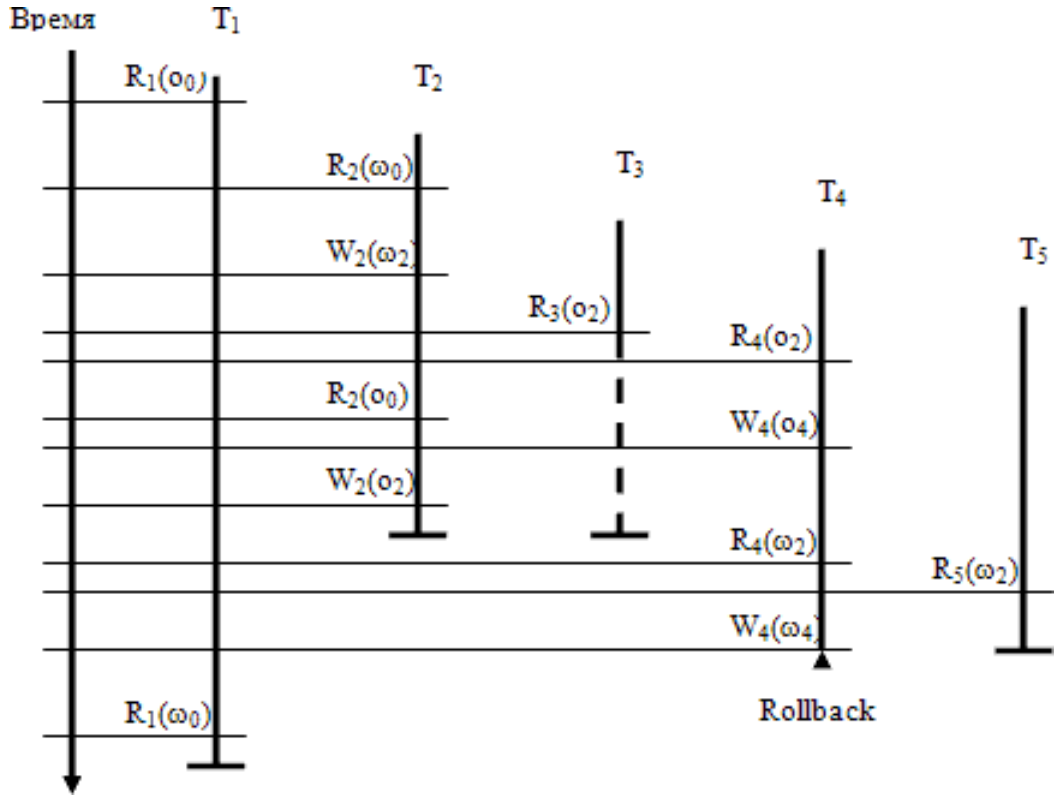


Основным преимуществом алгоритма MVTO является отсутствие задержек и откатов при выполнении операций чтения, а основным недостатком – возможность возникновения каскадных откатов транзакций при выполнении операций записи. Кроме того, в базе данных может накапливаться произвольное число версий одного и того же объекта, и определение того, какие версии больше не требуются, является серьезной технической проблемой.

ПРИМЕР АЛГОРИТМА MVTO

1. При использовании метода блокировок между T1 и T2 возник бы синхронизационный тупик, а при использовании обычного метода временных меток одна из транзакций подверглась бы откату. Однако при применении версий такие неприятности не возникают из-за того, что первая транзакция читает «старые» версии объектов o и ω.

2. Транзакция T_3 ожидает фиксации транзакции T_2 перед своим собственным завершением (на рис. это показано пунктирной линией). Это происходит потому, что транзакция T_3 прочитала версию o_2 объекта o, образованную еще не зафиксированной транзакцией.



3. Транзакция T_4 пытается создать версию ω_4 объекта ω после того, как еще не зафиксированная транзакция T_5 (начавшаяся позже) уже прочитала более раннюю версию ω_4. Поэтому транзакция T_5 не сможет «увидеть» изменения объекта ω, произведенные транзакцией T_4. Следовательно, сериализация транзакций в порядке получения ими временных меток становится невозможной, и приходится произвести откат транзакции T_4.

ДВУХВЕРСИОННЫЙ ВАРИАНТ 2PL

Версионный вариант двухфазного протокола синхронизационных блокировок

При описании *двухверсионного варианта протокола 2PL (**Two-Version Two-Phase Locking Protocol, 2V2PL**)* будем называть *текущими* версиями объектов базы данных версии, созданные зафиксированными транзакциями с наиболее поздним временем фиксации; *незафиксированными* версиями – версии, созданные еще незавершившимися транзакциями.

При следовании протоколу 2V2PL в каждый момент времени существует не более одной незафиксированной версии каждого объекта базы данных.

Операции любой транзакции T_i над объектом базы данных *o* обрабатываются следующим образом:

- операция $R_i(o)$ немедленно выполняется над текущей версией объекта *o*;
- операция $W_i(o)$, приводящая к созданию новой версии объекта *o*, выполняется только после завершения (фиксации или отката) транзакции, создавшей незафиксированную версию объекта *o*;
- выполнение операции COMMIT откладывается до тех пор, пока не завершатся все транзакции T_k, прочитавшие текущие версии объектов базы данных, которые должны замениться незафиксированными версиями этих объектов, созданными транзакцией T_i.

РЕАЛИЗАЦИЯ 2V2PL

Для реализации 2V2PL используются три типа блокировок:

- **RL (Read Lock)** – в этом режиме блокируется любой объект базы данных *o* перед выполнением операции чтения его текущей версии; удержание этой блокировки до конца транзакции гарантирует, что при повторном чтении объекта *o* будет прочитана та же версия этого объекта;
- **WL (Write Lock)** – в этом режиме блокируется любой объект базы данных *o* перед выполнением операции, приводящей к созданию новой (незафиксированной) версии этого объекта; удержание этой блокировки до конца транзакции гарантирует, что в любой момент времени будет существовать не более одной незафиксированной версии любого объекта базы данных;
- **CL (Commit Lock)** – блокировка устанавливается во время выполнения операции COMMIT транзакции и затрагивает любой объект базы данных, новую версию которого создала данная транзакция; удовлетворение этой блокировки для данной транзакции гарантирует, что завершились все транзакции, читавшие текущие версии объектов, новые версии которых были созданы при выполнении данной транзакции, и, следовательно, их можно заменить.

Совместимость блокировок

	RL(o)	WL(o)	CL(o)
RL(o)	да	да	нет
WL(o)	да	нет	нет
CL(o)	нет	нет	нет

- Операция чтения может блокироваться только на время фиксации транзакции, заменяющей текущую версию требуемого объекта базы данных.
- Для выполнения операции записи требуется долговременная монопольная блокировка соответствующего объекта базы данных, которая, совместима с блокировкой по чтению.
- Возможны синхронизационные тупики.

ГИБРИДНЫЙ ПРОТОКОЛ ROMV

Версионно-блокировочный протокол сериализации транзакций для поддержки только читающих транзакций (*Multiversion Protocol for Read-Only Transactions, ROMV*)

- При применении этого протокола при образовании **каждой транзакции явно указывается ее тип** – только читающая (read-only) или изменяющая (update) транзакция. В только читающих транзакциях допускается использование только операций чтения объектов базы данных, а в изменяющих транзакциях – операций и чтения, и записи.

- **Изменяющие транзакции** выполняются в соответствии с обычным протоколом 2PL, т.е. перед выполнением операции чтения или записи объекта базы данных о этот **объект должен быть заблокирован в режиме S или X соответственно**, и блокировки объектов удерживаются до конца изменяющей транзакции. Каждая операции записи объекта о создает его новую версию, которая при завершении транзакции помечается временной меткой, соответствующей моменту фиксации этой транзакции.

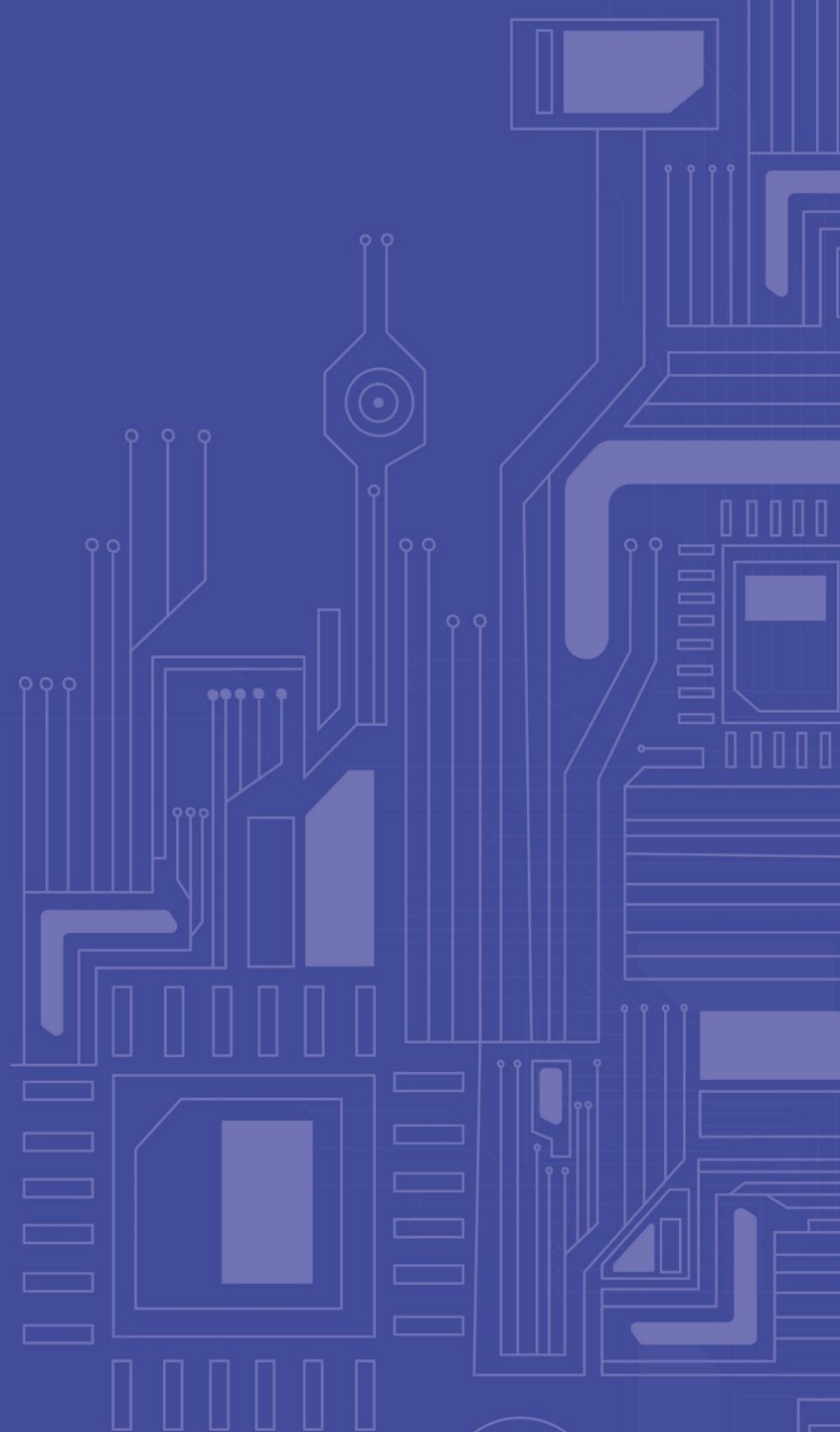
- Каждая **только читающая транзакция** при своем образовании получает соответствующую временную метку. При выполнении операции чтения объекта базы данных о транзакция получает доступ к версии объекта о, образованной изменяющей транзакцией, которая хронологически последней зафиксировалась к моменту образования данной читающей транзакции.

ПРЕИМУЩЕСТВА ROMV

Основным плюсом протокола ROMV по сравнению с ранее описанным протоколом 2V2PL является принципиальное отсутствие синхронизационных задержек при выполнении операций чтения только читающих транзакций. Если сравнивать ROMV с MVTO, то он выигрывает в принципиальном отсутствии откатов только читающих транзакций. Конечно, при работе изменяющих транзакций возможно возникновение синхронизационных тупиков и откатов, и здесь требуется использовать обычные методы распознавания и разрушения тупиков.

Кроме того, при использовании протокола ROMV в базе данных может возникать произвольное число версий объектов. Требуется создание специального сборщика мусора, который должен удалять ненужные версии данных. Простейший сборщик мусора удаляет все неиспользуемые версии, значения временных меток которых меньше значения временной метки старейшей активной только читающей транзакции.

ЖУРНАЛИЗАЦИЯ (ЛОГИ)



ЖУРНАЛИЗАЦИЯ

Общей целью журнализации изменений баз данных является обеспечение возможности восстановления согласованного состояния базы данных после любого сбоя.

Общими принципами восстановления являются следующие:

- результаты зафиксированных транзакций должны быть сохранены в восстановленном состоянии базы данных (т.е. должно поддерживаться свойство *долговечности (durability)* транзакций);
- результаты незафиксированных транзакций должны отсутствовать в восстановленном состоянии базы данных (в противном случае состояние базы данных могло бы оказаться не целостным).

Возможны следующие ситуации, при которых требуется производить восстановление состояния базы данных:

- Индивидуальный откат транзакции. Тривиальной ситуацией отката транзакции является ее явное завершение оператором ROLLBACK.
- Восстановление после внезапной потери содержимого оперативной памяти (мягкий сбой). Такая ситуация может возникнуть при аварийном выключении электрического питания, при возникновении неустранимого сбоя процессора (например, срабатывании контроля основной памяти) и т.д. Ситуация характеризуется потерей буфера оперативной памяти СУБД.
- Восстановление после поломки основного внешнего носителя базы данных (жесткий сбой).

Во всех трех случаях основой восстановления является хранение избыточных данных. Эти избыточные данные хранятся в журнале, содержащем последовательность записей об изменении базы данных.

БУФЕРИЗАЦИЯ

Рассматриваем подход общего журнала СУБД, без локальных журналов транзакций (что тоже возможно, но более затратно). Рассматриваем старые классические простые принципы работы СУБД (System R).

По причинам объективно существующей разницы в скорости работы процессоров и основной памяти и устройств внешней памяти (эта разница в скорости существовала, существует, и будет существовать всегда) **буферизация блоков базы данных** в основной памяти является единственным реальным способом достижения приемлемой эффективности СУБД. Без поддержки буферизации базы данных СУБД работала бы со скоростью магнитных дисков, т.е. на несколько порядков медленнее, чем если бы обработка данных происходила в основной памяти.

Поэтому записи в журнал тоже буферизуются: при нормальной работе буфер выталкивается во внешнюю память журнала только при полном заполнении записями. Более точно, для буферизации записей журнала обычно используются два буфера.

- После полного заполнения **первый буфер** выталкивается на магнитный диск, и пока совершается этот обмен, журнальные записи размещаются во втором буфере.
- К моменту конца обмена заполняется **второй буфер**, он выталкивается во внешнюю память, а журнальные записи снова размещаются в первом буфере и т.д.

Используются буфера *(и базы данных, и журнала),* ***располагающиеся именно в физической основной памяти****, управляемой непосредственно СУБД, а не виртуальной памяти СУБД, управляемой операционной системой. Использование буферов виртуальной памяти является практически бессмысленным делом, поскольку в этом случае операционная система, руководствуясь своими собственными стратегиями управления основной памяти, в любой момент может удалить буферную страницу СУБД из основной памяти и перенести ее копию во внешнюю память.*

УПРАВЛЕНИЕ БУФЕРНЫМ ПУЛОМ

OS

Если некоторый процесс требует обеспечения доступа к странице виртуальной памяти, отсутствующей в основной памяти, и нет свободных страниц основной памяти, в соответствии с некоторым критерием выбирается некоторая занятая страница основной памяти, освобождается (т.е. изымается из виртуальной памяти какого-то процесса и, может быть, копируется на диск) и подключается к виртуальной памяти запросившего процесса с предварительным считыванием с диска нужных данных.

СУБД

Если при выполнении некоторой операции в некоторой транзакции требуется доступ к некоторому блоку базы данных, и копия этого блока отсутствует в буферном пуле, СУБД должна выделить какую-либо страницу буферного пула, считать в нее с диска требуемый блок базы данных и предоставить доступ к этой странице запросившей операции. Конечно, в буферном пуле может не оказаться свободных страниц, и тогда СУБД в соответствии с некоторым критерием находит некоторую занятую страницу, освобождает ее (возможно, выталкивает во внешнюю память).

Почти всегда ОС стремится заменить страницу, к которой предположительно дольше всего не будет обращений, но, поскольку предвидение будущего невозможно, оно аппроксимируется прошлым.

В частности, в одном из популярных алгоритмов замещения страниц **LRU (Least Recently Used)** принимается предположение, что дольше всего в будущем не потребуется та страница, к которой дольше всего не обращались в прошлом.

МЕТОД LRU ДЛЯ СУБД

В стратегии замещения страниц буферного пула СУБД тоже чаще всего используется некоторая разновидность алгоритма LRU. Но, как уже отмечалось выше, СУБД располагает большей информацией о страницах буферного пула, чем операционная система о страницах основной памяти.

ПРИМЕР

Если в некоторой транзакции выполняется сканирование некоторой таблицы без использования индекса, и при выполнении операции NEXT был затребован доступ к некоторому блоку базы данных (с соответствующим перемещением копии этого блока в некоторую страницу буферного пула), то подсистема управления буферным пулом «знает», что эта страница еще точно потребуется до тех пор, пока не будет прочитан последний кортеж сканируемой таблицы, располагающийся в данной странице. Более того, СУБД «знает», какой блок базы данных потребуется после завершения просмотра кортежей данного блока, и может заранее переместить его копию в некоторую страницу буферного пула.

КОРНЕВЫЕ БЛОКИ

При любом просмотре таблицы на основе некоторого индекса гарантированно потребуется доступ к корневому блоку соответствующего B-дерева. При вставке кортежа в любую таблицу или удалении из нее кортежа будет необходимо должным образом изменить все определенные для нее индексы, и для этого тоже гарантированно потребуется доступ к корневым блокам всех соответствующих B-деревьев.

В стратегии замещения страниц буферного пула базы данных обычно используется алгоритм LRU с приоритетами страниц В частности, страницы, содержащие копии корневых блоков индексов, являются настолько высокоприоритетными, что обычно никогда не замещаются. Кроме того, поддерживается предварительное считывание в буферную память копий блоков, доступ к которым вскоре понадобится.

ФИЗИЧЕСКАЯ СИНХРОНИЗАЦИЯ (1/2)

Поскольку в СУБД может одновременно («параллельно») выполняться несколько транзакций, вполне реальна ситуация, когда в двух одновременно выполняемых операциях требуется доступ к одному и тому же блоку базы данных (т.е. к одной и той же буферной странице, содержащей копию этого блока). Понятно, что в одновременном доступе для чтения содержимого блока ничего плохого нет, но параллельное изменение блока может привести к непредсказуемым результатам.

ПРИМЕР 1

Предположим, что в двух параллельно выполняемых транзакциях одновременно выполняются операции модификации кортежей, у одного из которых tid = (*n*, 1), а у другого tid = (*n*, 2). Если в СУБД используются блокировки на уровне кортежей, то система допустит параллельное выполнение этих двух операций, и они будут одновременно изменять страницу, содержащую копию блока базы данных с номером *n*. При выполнении обеих операций может потребоваться перемещение кортежей внутри этого блока, и понятно, что в результате ничего хорошего, скорее всего, не получится. Также, логическая синхронизация может легко допустить параллельное выполнение нескольких операций, требующих обновления одного и того же индекса. Некоординированное параллельное обновление B-дерева с большой вероятностью приводит к разрушению его структуры.

Поэтому **при выполнении операций уровня RSS необходимо поддерживать дополнительную «физическую» синхронизацию**, в которой единицами блокировки служат страницы буферного пула (или блоки) базы данных. В пределах операции перед чтением из страницы буферного пула (блока базы данных) требуется запросить у подсистемы управления буферным пулом блокировку соответствующей страницы (блока) в режиме S, а перед записью в страницу (в блок) – ее блокировку в режиме X.

ФИЗИЧЕСКАЯ СИНХРОНИЗАЦИЯ (2/2)

ПРИМЕР 2 При выполнении операций уровня RSS могут возникать ошибки, обнаруживаемые в середине операции, уже после того, как одна или несколько страниц буферного пула (блоков базы данных) было изменено. Например, может выполняться операция вставки кортежа в некоторую таблицу, нарушающая уникальность некоторого индекса, определенного над этой таблицей. Нарушение уникальности этого индекса будет обнаружено при попытке вставить в него новый ключ, но до этого новый кортеж уже мог быть размещен в блоке данных, и некоторые индексы уже могли быть успешно обновлены.

При обнаружении ошибки операции нужно ликвидировать все ее следы в базе данных и выдать соответствующий код ошибки на уровень RDS. Проще всего сделать это, произведя обратные изменения всех страниц (блоков базы данных), которые были изменены при прямом выполнении операции. Но для этого **требуется, чтобы все страницы (блоки базы данных), заблокированные при выполнении операции, оставались заблокированными до конца этой операции.**

Тем самым, для подсистемы управления буферным пулом операции уровня RSS являются (почти) тем же, чем являются транзакции для подсистемы управления транзакциями. **Достаточным условием корректного выполнения операций является соблюдение двухфазного протокола синхронизационных блокировок над страницами буферного пула в пределах операций.**

ДВА БУФЕРА

Имеются два вида буферов – буфера журнала и буферный пул страниц основной памяти, – которые содержат связанную информацию. И те, и другие буфера могут выталкиваться во внешнюю память.

- Основной причиной выталкивания буфера журнала является его полное заполнение журнальными записями.
- Страницы буферного пула базы данных чаще всего выталкиваются во внешнюю память, когда требуется переместить в основную память некоторый блок базы данных, а свободных страниц в буферном пуле нет. Тогда срабатывает алгоритм замещения страниц, выбирается страница, содержимое которой, вероятно, дольше всего не потребуется, и эта страница (если ее содержимое изменялось) выталкивается в соответствующий блок внешней памяти базы данных.

Проблема состоит в выработке некоторой общей политики выталкивания, которая обеспечивала бы возможность восстановления состояния базы данных после сбоев.

Эта проблема не возникает при индивидуальных откатах транзакций, поскольку в этих случаях содержимое основной памяти не утрачено, и при восстановлении можно пользоваться содержимым как буфера журнала, так и буферных страниц базы данных.

Но если произошел мягкий сбой, и содержимое буферов утрачено, то для проведения восстановления базы данных необходимо иметь некоторое согласованное состояние журнала и базы данных во внешней памяти.

ПРОТОКОЛ УПРЕЖДАЮЩЕЙ ЗАПИСИ

Основным принципом согласованной политики выталкивания буфера журнала и буферных страниц базы данных является то, что запись об изменении объекта базы данных должна оказаться во внешней памяти журнала раньше, чем измененный объект окажется во внешней памяти базы данных. Соответствующий протокол журнализации (и управления буферизацией) называется **WAL (Write Ahead Log, «пиши сначала в журнал»)** и состоит в том, что если требуется вытолкнуть во внешнюю память буферную страницу, содержащую измененный объект базы данных, то перед этим нужно гарантировать выталкивание во внешнюю память журнала буферной страницы журнала, содержащей запись об изменении этого объекта.

- Дополнительное условие на выталкивание буферов накладывается тем требованием, что **каждая успешно завершенная транзакция должна быть реально зафиксирована во внешней памяти.** Какой бы сбой не произошел, система должна быть в состоянии восстановить состояние базы данных, содержащее результаты всех транзакций, зафиксированных до момента сбоя.

- Минимальным требованием, гарантирующим возможность восстановления последнего согласованного состояния базы данных, является **выталкивание при фиксации транзакции во внешнюю память журнала всех записей об изменении базы данных этой транзакцией.** При этом последней записью в журнал, производимой от имени данной транзакции, является специальная запись о конце транзакции.

ИНДИВИДУАЛЬНЫЙ ОТКАТ ТРАНЗАКЦИИ (1/2)

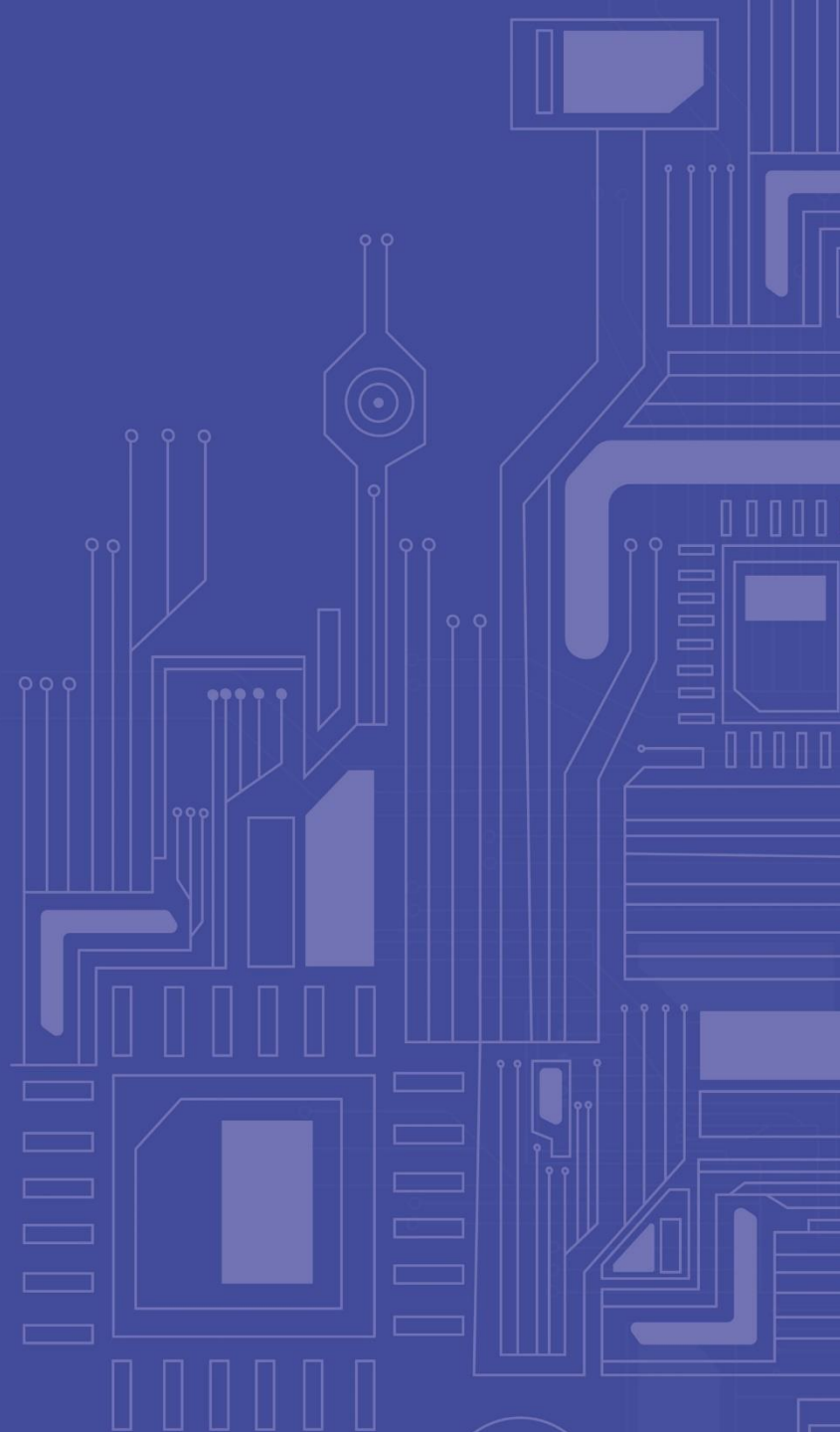
- Для обеспечения возможности индивидуального отката транзакции по общему журналу все записи в журнале от данной транзакции связываются в **обратный список**.
- **В начале списка** для незавершенных транзакций находится запись о последнем изменении базы данных, произведенном данной транзакцией. Заметим, что в этом случае хронологически последние записи могут быть еще не вытолкнуты во внешнюю память журнала и могут находиться в буфере основной памяти.
- **Для закончившихся транзакций** (индивидуальные откаты которых уже невозможны) началом списка является запись о конце транзакции, которая обязательно вытолкнута во внешнюю память журнала, т.е. весь список находится во внешней памяти.
- **Концом списка** всегда служит первая запись об изменении базы данных, произведенном данной транзакцией.
- Обычно в каждой записи **проставляется уникальный идентификатор транзакции**, чтобы можно было восстановить прямой список записей об изменениях базы данных данной транзакцией.

ИНДИВИДУАЛЬНЫЙ ОТКАТ ТРАНЗАКЦИИ (2/2)

Индивидуальный откат транзакции (это возможно только для незавершенных транзакций) выполняется следующим образом:

- Выбирается очередная журнальная запись из списка данной транзакции.
- Выполняется противоположная по смыслу операция: вместо операции INSERT выполняется соответствующая операция DELETE, вместо операции DELETE выполняется INSERT, и вместо прямой операции UPDATE – обратная операция UPDATE, восстанавливающая предыдущее состояние объекта базы данных.
- Любая из этих обратных операций также журнализуется. Собственно для индивидуального отката это не нужно, но при выполнении индивидуального отката транзакции может произойти мягкий сбой, при восстановлении после которого потребуется откатить транзакции, для которых не полностью выполнен индивидуальный откат.
- При успешном завершении отката в журнал заносится запись о конце транзакции. С точки зрения журнала такая транзакция является зафиксированной.

ВООСТАНОВЛЕНИЕ ПОСЛЕ СБОЕВ



МЯГКИЙ СБОЙ

Мягкий сбой характеризуется утратой оперативной памяти системы. При этом поражаются все выполняющиеся в момент **сбоя** транзакции, теряется содержимое всех буферов базы данных. Данные, хранящиеся на диске, остаются неповрежденными.

После мягкого сбоя набор блоков внешней памяти базы данных может оказаться несогласованным, т.е. часть блоков внешней памяти соответствует объекту до изменения, часть – после изменения. Например, в результате выполнения операции UPDATE соответствующий кортеж мог переместиться в другой блок. В этом случае изменяются два блока: в описатель кортежа в его исходном блоке записывается его новый tid, а в новом блоке размещается сам модифицированный кортеж. Очевидно, что если хотя бы один из этих блоков не попал во внешнюю память базы данных к моменту мягкого сбоя, то при восстановлении не удастся вернуть кортеж на его прежнее место. Другими словами, к такому состоянию внешней памяти базы данных не применимы операции логического уровня.

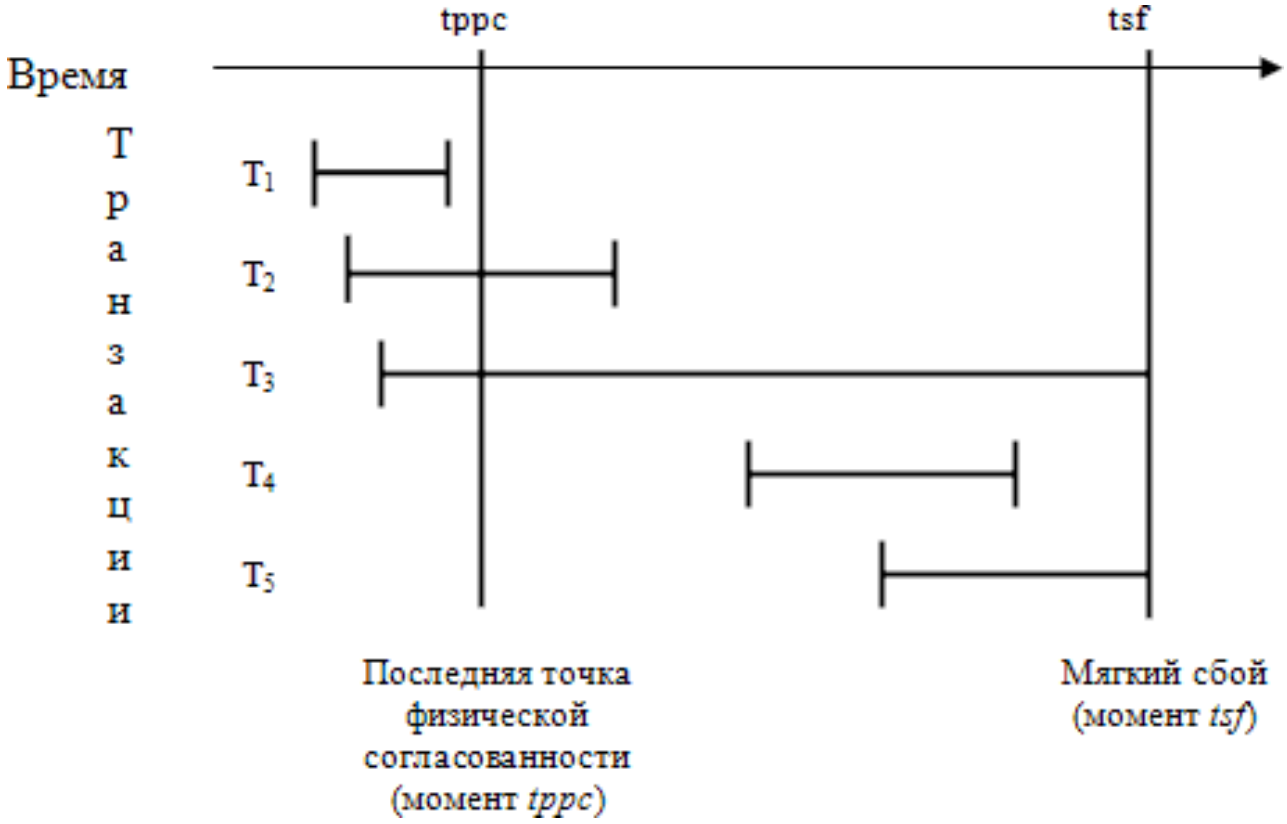
Состояние внешней памяти базы данных называется ***физически согласованным***, если наборы страниц всех объектов согласованы, т.е. соответствуют состоянию любого объекта либо после его изменения, либо до изменения.

Будем считать, что в журнале отмечаются точки физической согласованности базы данных – моменты времени, в которые во внешней памяти содержатся согласованные результаты операций, завершившихся до соответствующего момента времени, и отсутствуют результаты операций, которые не завершились, а буфер журнала вытолкнут во внешнюю память. Назовем такие точки ppc (point of physical consistency).

ОБРАБОТКА ТРАНЗАКЦИЙ ПРИ МЯГКОМ СБОЕ (1/2)

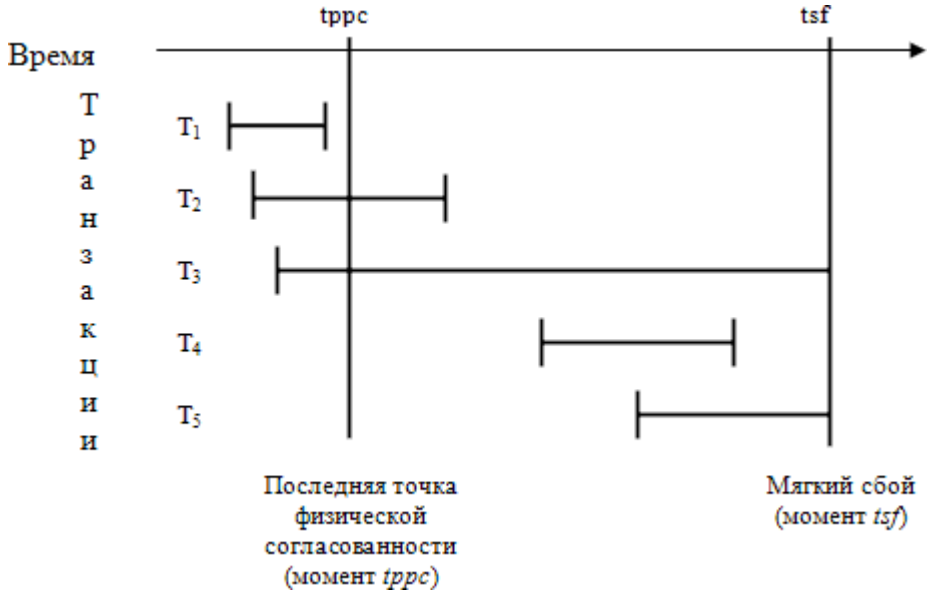
Для транзакции T_1 никаких действий производить не требуется. Она закончилась до момента *tppc*, и все ее результаты гарантированно отражены во внешней памяти базы данных.

Для транзакции T_2 нужно повторно выполнить (*redo*) последовательность операций, которые выполнялись после установки точки физически согласованного состояния в момент *tppc*. Действительно, во внешней памяти полностью отсутствуют следы операций, которые выполнялись в транзакции T_2 после момента *tppc*. Следовательно, повторное прямое (по смыслу и хронологии) выполнение операций транзакции T_2 корректно и приведет к логически согласованному состоянию базы данных. (Поскольку транзакция T_2 успешно завершилась до момента мягкого сбоя *tfs*, в журнале содержатся записи обо всех изменениях базы данных, произведенных этой транзакцией.)



ОБРАБОТКА ТРАНЗАКЦИЙ ПРИ МЯГКОМ СБОЕ (2/2)

Для транзакции T_3 нужно выполнить в обратном направлении (undo) ту часть операций, которую она успела выполнить до момента *tppc*. Действительно, во внешней памяти базы данных полностью отсутствуют результаты операций T_3, которые были выполнены после момента *tppc*. С другой стороны, во внешней памяти гарантированно присутствуют результаты операций T_3, которые были выполнены до момента *tppc*. Следовательно, обратное выполнение (по смыслу и хронологии) операций T_3 корректно и приведет к согласованному состоянию базы данных. (Поскольку транзакция T_3 не завершилась к моменту мягкого сбоя *tfs*, при восстановлении необходимо устранить все последствия ее выполнения.)



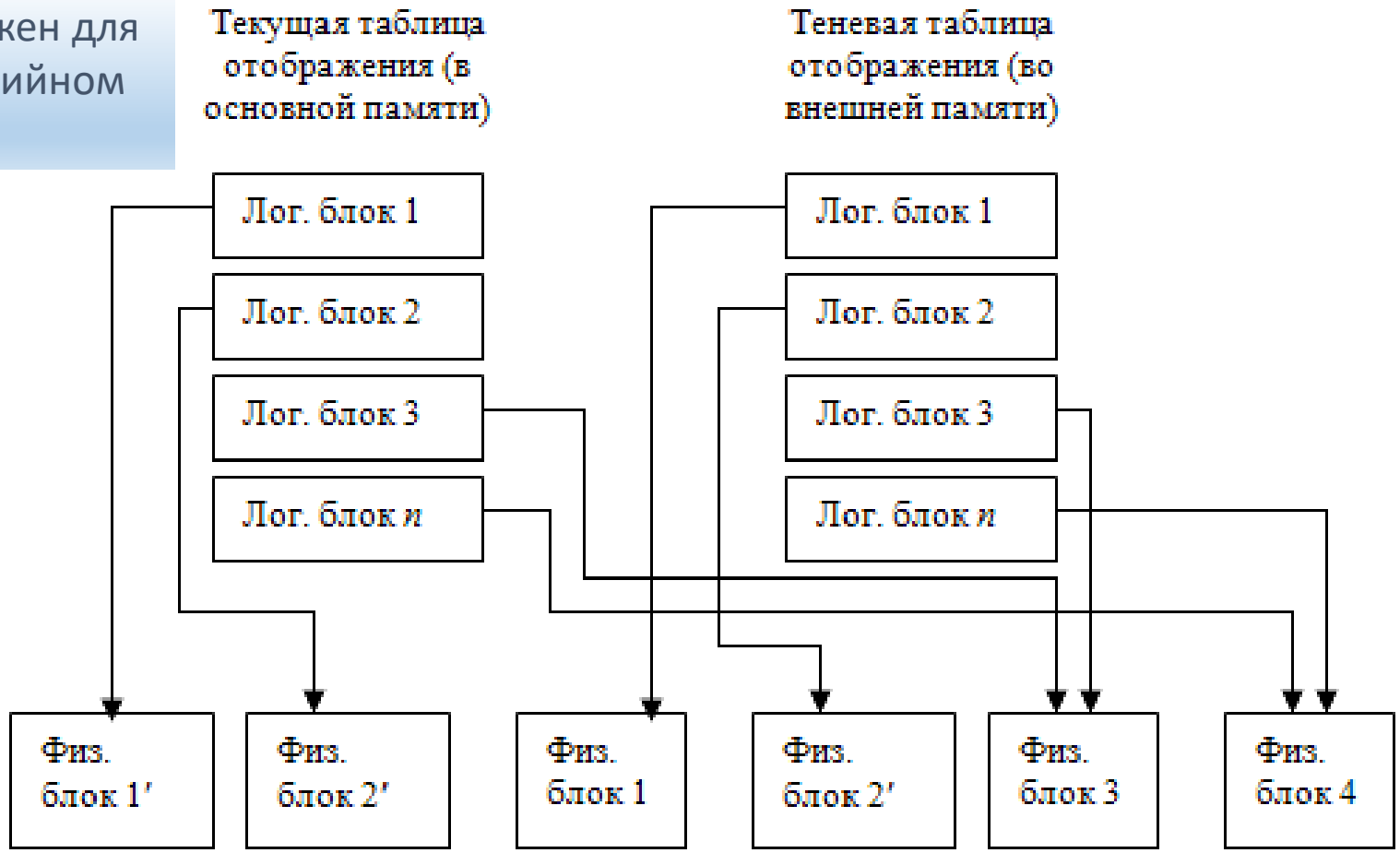
Для транзакции T_4, которая успела начаться после момента *tppc* и закончиться до момента мягкого сбоя *tfs*, нужно произвести полное повторное выполнение операций в прямом направлении. (Поскольку транзакция T_4 успешно завершилась до момента мягкого сбоя *tfs*, в журнале содержатся записи обо всех изменениях базы данных, произведенных этой транзакцией).

Наконец, для транзакции T_5, начавшейся после момента *tppc* и не успевшей завершиться к моменту мягкого сбоя *tfs*, никаких действий предпринимать не требуется. Результаты операций этой транзакции полностью отсутствуют во внешней памяти базы данных.

ТЕНЕВОЙ МЕХАНИЗМ ВОССТАНОВЛЕНИЯ ФАЙЛОВ

Теневой механизм был изначально предложен для поддержания целостности файлов при аварийном отключении питания компьютера.

При открытии файла таблица отображения номеров его логических блоков в адреса физических блоков внешней памяти считывается в оперативную память. При модификации любого блока файла во внешней памяти выделяется новый блок. При этом текущая таблица отображения (в основной памяти) изменяется, а теневая остается неизменной. Если во время работы с открытым файлом происходит сбой, во внешней памяти автоматически сохраняется состояние файла до его открытия.



Для явного восстановления файла достаточно повторно считать в основную память теневую таблицу отображения.

ТЕНЕВОЙ МЕХАНИЗМ ВОССТАНОВЛЕНИЯ БД

В контексте базы данных теневой механизм используется следующим образом. Периодически выполняются операции установки точки физической согласованности базы данных. При выполнении этой операции все логические операции завершаются, все страницы буферного пула базы данных, содержимое которых отличается от содержимого соответствующих блоков внешней памяти, выталкиваются. Теневая таблица отображения файлов (сегментов) базы данных заменяется текущей таблицей отображения (правильнее сказать, текущая таблица отображения записывается на место теневой).

ПРОБЛЕМА С ТЕНЕВОЙ ТАБЛИЦЕЙ ОТОБРАЖЕНИЯ

Проблема состоит в том, что в любой момент времени теневая таблица отображения должна быть корректной, т.е. соответствовать некоторому ранее зафиксированному физически целостному состоянию базы данных.

Для этого **необходимо обеспечить атомарность операции замены теневой таблицы отображения**. В общем случае таблица отображения может занимать несколько блоков внешней памяти, и для записи текущей таблицы отображения на место теневой таблицы в этом случае потребуется несколько обменов с дисками. Если в промежутке между этими обменами возникнет мягкий сбой, то будет благополучно утрачена текущая таблица отображения и безнадежно испорчена теневая таблица, т.е. мы просто лишимся возможности восстанавливаться за счет использования последнего физически согласованного состояния базы данных.

ХРАНЕНИЕ ТЕНЕВОЙ ТАБЛИЦЫ

- Во внешней памяти поддерживаются две области хранения таблицы отображения файлов (будем называть их областями A и B).
- Кроме того, в отдельном блоке внешней памяти хранится флаг F, показывающий, какая из этих областей в данный момент содержит действующую теневую таблицу отображения (назовем соответствующие значения флага F_A и F_B).
- Тогда, если сохраненным во внешней памяти значением флага является F_A, то текущая таблица отображения записывается в область B. Если эта операция выполняется успешно, то в блок флага записывается значение F_B. Считается, что операция записи одного блока на диск является атомарной. Если эта операция заканчивается успешно, это означает, что новая теневая таблица отображения хранится в области B. Если же запись текущей таблицы отображения в область B не удалась, или если не выполнилась операция записи блока с флагом F, то продолжает действовать старая теневая таблица отображения.
- Восстановление хронологически последнего сохраненного физически согласованного состояния базы данных происходит мгновенно: текущая таблица отображения заменяет теневой таблицей (при восстановлении просто считывается действующая теневая таблица отображения). Все проблемы восстановления решаются, но за счет слишком большого перерасхода внешней памяти. В пределе может потребоваться вдвое больше внешней памяти, чем реально нужно для хранения базы данных.

ЖУРНАЛИЗАЦИЯ ПОСТРАНИЧНЫХ ИЗМЕНЕНИЙ

Возможен другой подход, при использовании которого наряду с логической журнализацией операций изменения базы данных производится журнализация постраничных изменений.

- Первый этап восстановления после мягкого сбоя состоит в постраничном откате недовыполненных логических операций. Подобно тому, как это делается с логическими записями по отношению к транзакциям, последней записью о постраничных изменениях от одной логической операции является запись о конце операции.

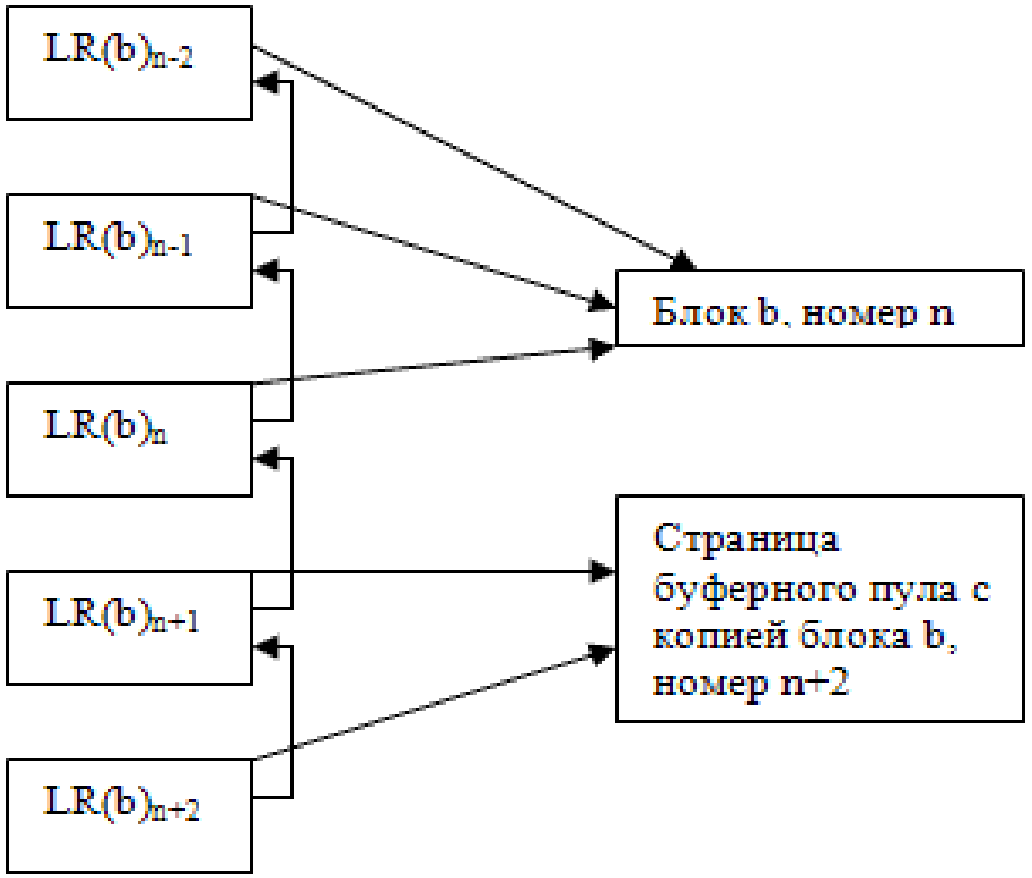
- При выполнении логической операции обновления базы данных может изменяется несколько блоков базы данных. Для обеспечения возможности отката отдельной операции приходится до конца операции монопольно блокировать все страницы буферного пула базы данных, содержащие копии изменяемых этой операцией блоков базы данных.

Чтобы распознать, нуждается ли страница внешней памяти базы данных в восстановлении, при выталкивании любой страницы из буферного пула основной памяти в нее помещается номер последней записи о постраничном изменении этой страницы. Этот же номер запоминается в самой записи. Тогда, чтобы понять, нужно ли применить данную запись о постраничном изменении соответствующего блока внешней памяти для восстановления состояния этого блока, требуется всего лишь сравнить номер, содержащийся в этом блоке, с номером, содержащимся в журнальной записи. Если в блоке содержится номер, меньший номера журнальной записи, то это означает, что буферная страница, в которой выполнялось соответствующее изменение, не была к моменту мягкого сбоя вытолкнута во внешнюю память, и применять данную запись для восстановления соответствующего блока внешней памяти не требуется.

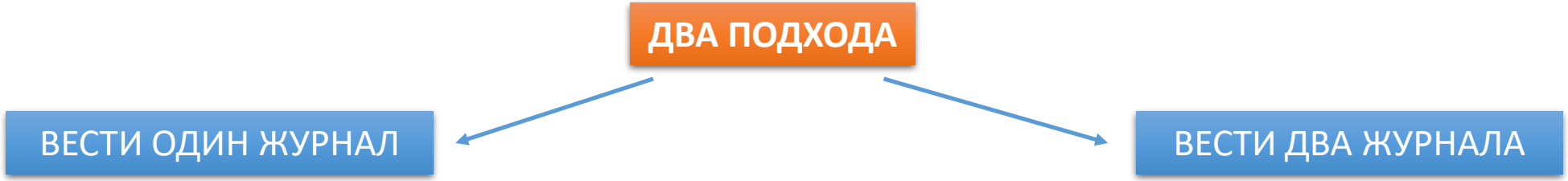
ЖУРНАЛИЗАЦИЯ ПОСТРАНИЧНЫХ ИЗМЕНЕНИЙ: ПРИМЕР

Пусть в рамках какой-то транзакции с блоком b последовательно происходят изменения LR(b)n-2 ,…., LR(b)n+2 . И тут случился сбой.

В блоке b содержится номер *n*. Это означает, что в состоянии блока отражены результаты операций изменения блока, соответствующих журнальным записям $LR(b)_n$, $LR(b)_{n-1}$ и $LR(b)_{n-2}$. Изменения блока, произведенные операциями, которым соответствуют две хронологически последние журнальные записи $LR(b)_{n+1}$ и $LR(b)_{n+2}$, в его состоянии во внешней памяти не отражены, поскольку не было выполнено выталкивание во внешнюю память страницы буферного пула, содержащей копию блока b. Поэтому при восстановлении состояния блока требуется выполнить обратные операции изменения блока b, соответствующие журнальным записям $LR(b)_n$, $LR(b)_{n-1}$ и $LR(b)_{n-2}$.



ЖУРНАЛИЗАЦИЯ ПОСТРАНИЧНЫХ ИЗМЕНЕНИЙ: ВАРИАЦИИ



Поддерживается общий журнал логических и страничных операций. Естественно, наличие двух видов записей, интерпретируемых абсолютно по-разному, усложняет структуру журнала. Кроме того, записи о постраничных изменениях, актуальность которых носит локальный характер, существенно (и не очень осмысленно) увеличивают журнал.

Распространено поддержание отдельного (короткого) журнала постраничных изменений. Такой журнал обычно называют ***физическим* журналом**, поскольку он содержит записи об изменении физических объектов – блоков внешней памяти. В отличие от этого, журнал логических операций принято называть ***логическим* журналом**, поскольку в нем содержатся записи об операциях над логическими объектами – кортежами.

Обычно ведут два журнала. Далее более подробно именно об этом подходе.

ЛОГИЧЕСКИЙ ЖУРНАЛ

СВОЙСТВА ЛОГИЧЕСКОГО ЖУРНАЛА

Логический журнал должен поддерживать как обратное выполнение журнализованных операций (*undo*), так и их повторное прямое выполнение (*redo*). В отличие от этого, от физического журнала требуется только поддержка обратного выполнения постраничных операций.

Логический журнал обычно начинает заполняться заново только после выполнения операций резервного копирования базы данных или архивирования самого журнала. До этого времени он линейно растет. Понятно, что в любом случае для размещения журнала выделяется внешняя память ограниченного размера. Предельный размер журнала определяется администратором базы данных и должен согласовываться с размером интервала времени, через которое производится резервное копирование базы данных.

ОБРАБОТКА ПЕРЕПОЛНЕНИЯ ЛОГИЧЕСКОГО ЖУРНАЛА

На пути к достижению максимально возможного размера журнала устанавливаются «желтая» и «красная» зоны. Когда записи в журнал достигают «желтой» зоны, выдается предупреждение администратору базы данных и прекращается образование новых транзакций. Если все существующие транзакции завершаются до достижения «красной» зоны, автоматически выполняется архивация базы данных или логического журнала. Если какие-то транзакции не успевают завершиться до достижения «красной» зоны журнала, выполняется их аварийный откат, после чего производится архивация базы данных или журнала. Естественно, размер «желтой» и «красной» зон логического журнала должен устанавливаться администратором базы данных с учетом максимально допустимого числа одновременно существующих транзакций и их возможной протяженности.

ФИЗИЧЕСКИЙ ЖУРНАЛ

Физический журнал существует сравнительно недолгое время (интервал времени между соседними операциями установки точки физической согласованности базы данных) и, как правило, занимает существенно меньшее дисковое пространство, чем логический журнал.

ВЫПОЛНЕНИЕ ОПЕРАЦИИ УСТАНОВКИ ТОЧКИ ФИЗИЧЕСКОЙ СОГЛАСОВАННОСТИ

1. Прекращают инициироваться новые логические операции;
2. после завершения всех выполняемых логических операций происходит выталкивание во внешнюю память всех модифицированных страниц буферного пула;
3. формируется и выталкивается во внешнюю память логического журнала специальная запись о точке физически согласованного состояния;
4. в случае успешного предыдущего действия разрешается инициация новых логических операций, и физический журнал пишется заново.

Предпоследняя операция является атомарной (это опять же запись одного блока на диск): если она успешно выполняется, то при следующем восстановлении после мягкого сбоя будет использоваться новая точка физически согласованного состояния, иначе ситуация воспринимается как мягкий сбой с восстановлением логически согласованного состояния базы данных от предыдущей точки физически согласованного состояния (с оповещением об этом администратора базы данных).

ЖЕСТКИЙ СБОЙ

Жесткий сбой – утрата внешней памяти, при этом буфер оперативной памяти может не пострадать.

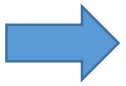
Для **восстановления последнего согласованного состояния базы данных** после жесткого сбоя журнала изменений базы данных явно недостаточно.

Самый простой способ восстановления основывается на использовании логического журнала и архивной копии базы данных.

Восстановление начинается с обратного копирования (на исправный носитель) базы данных из архивной копии. Затем для всех закончившихся транзакций выполняется *redo*, т.е. операции повторно выполняются в прямом смысле.

Более точно, происходит следующее:

- по журналу в прямом направлении выполняются все операции;
- для транзакций, которые не закончились к моменту сбоя, выполняется откат.



Поскольку жесткий сбой не сопровождается утратой буферов оперативной памяти, можно восстановить базу данных до такого уровня, чтобы можно было продолжить даже выполнение незавершенных транзакций. Но обычно это не делается, потому что восстановление после жесткого сбоя – это достаточно длительный процесс.

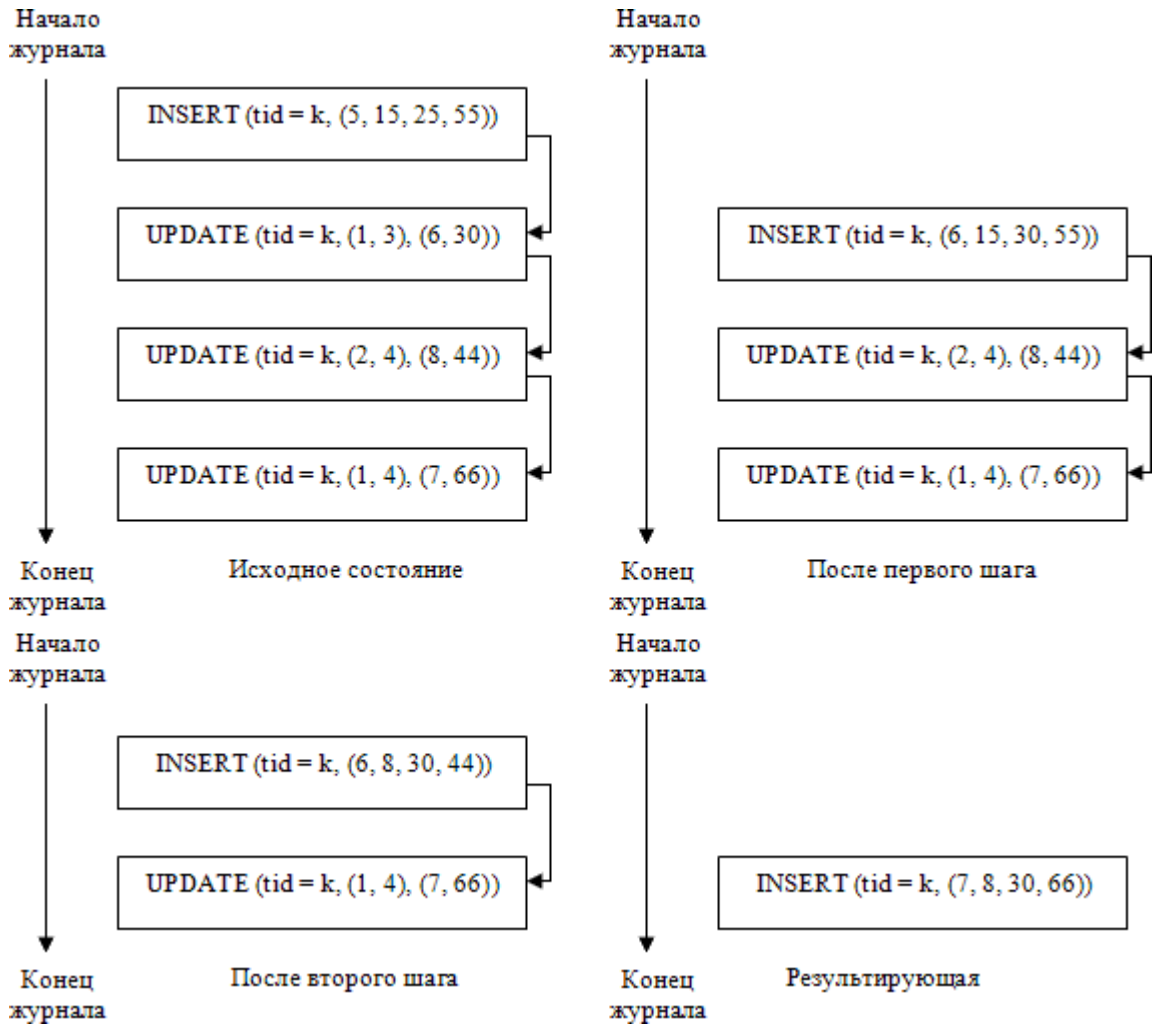
Хотя к ведению журнала предъявляются особые требования по части надежности, в принципе возможна и его утрата. Тогда единственным способом восстановления базы данных является возврат к архивной копии. Конечно, в этом случае не удастся получить последнее согласованное состояние базы данных, но это лучше, чем ничего.

АРХИВАЦИЯ

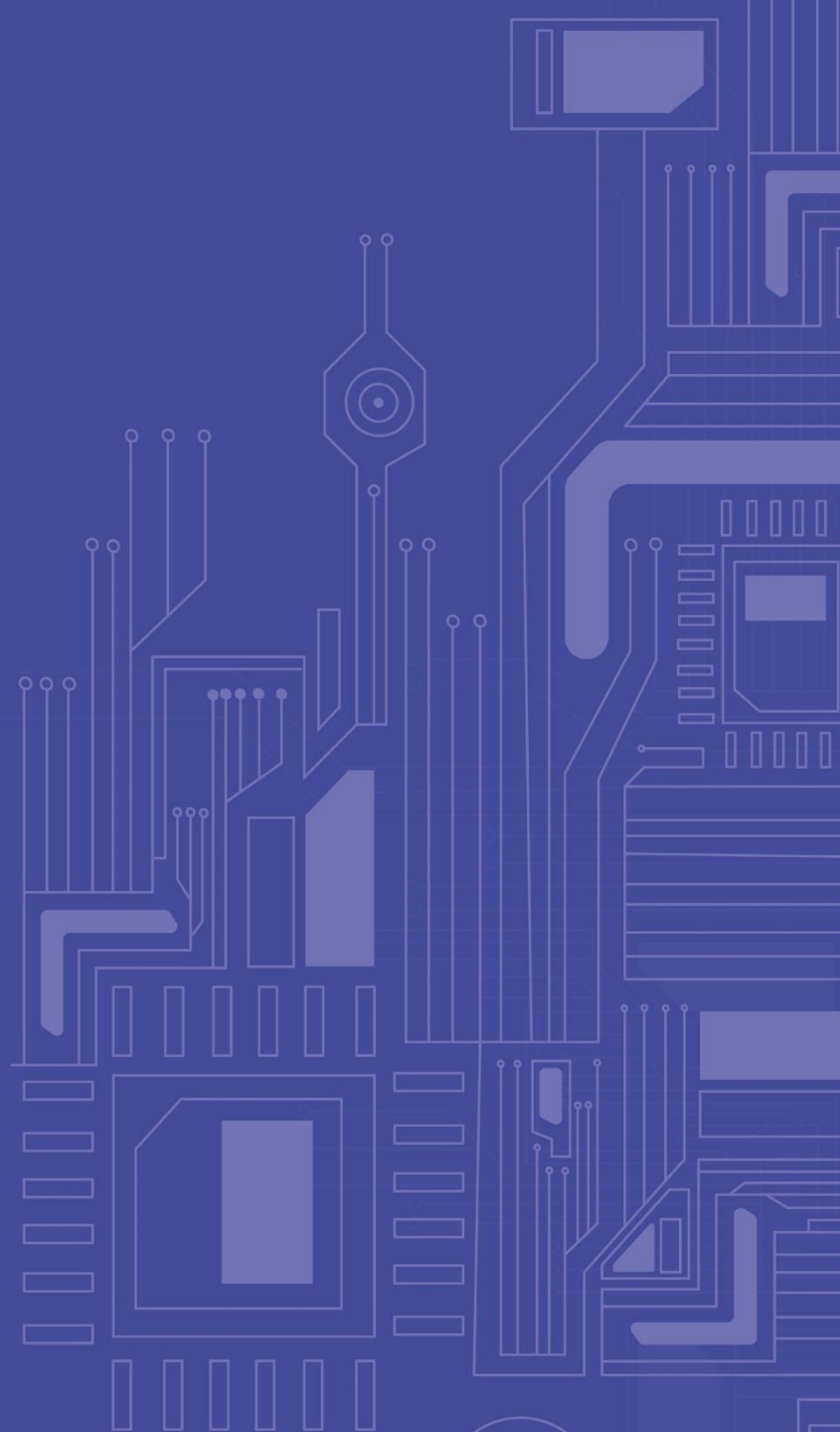
Вместо базы данных можно архивировать сам журнал.
В пределе для полного восстановления базы данных после жесткого сбоя достаточно иметь исходную архивную копию базы данных, последовательность архивных копий журналов и последний логический журнал.

Архивированный логический журнал можно сжимать.
Для этого для каждого объекта базы данных нужно найти последовательность журнальных записей, относящихся к этому объекту, в хронологическом порядке и заменить их одной записью, соответствующей операции над объектом, результат которой эквивалентен результату последовательного выполнения журнализованных операций из построенной последовательности.

Таким образом, для восстановления после жесткого сбоя можно воспользоваться исходной архивной копией, одним сжатым архивным журналом и последним логическим журналом.



ВВЕДЕНИЕ В SQL



ИСТОРИЯ SQL

Язык SQL, предназначенный для взаимодействия с базами данных, появился в середине 70-х гг. (первые публикации датируются 1974 г.) и был разработан в компании IBM в рамках проекта экспериментальной реляционной СУБД System R. Исходное название языка SEQUEL (Structured English Query Language) только частично отражало суть этого языка. Конечно, язык был ориентирован главным образом на удобную и понятную пользователям формулировку запросов к реляционным БД.

Возможности первого SEQUEL

- средства определения и манипулирования схемой БД;
- средства определения ограничений целостности и триггеров;
- средства определения представлений БД;
- средства определения структур физического уровня, поддерживающих эффективное выполнение запросов;
- средства авторизации доступа к отношениям и их полям;
- средства определения точек сохранения транзакции и выполнения фиксации и откатов транзакций.

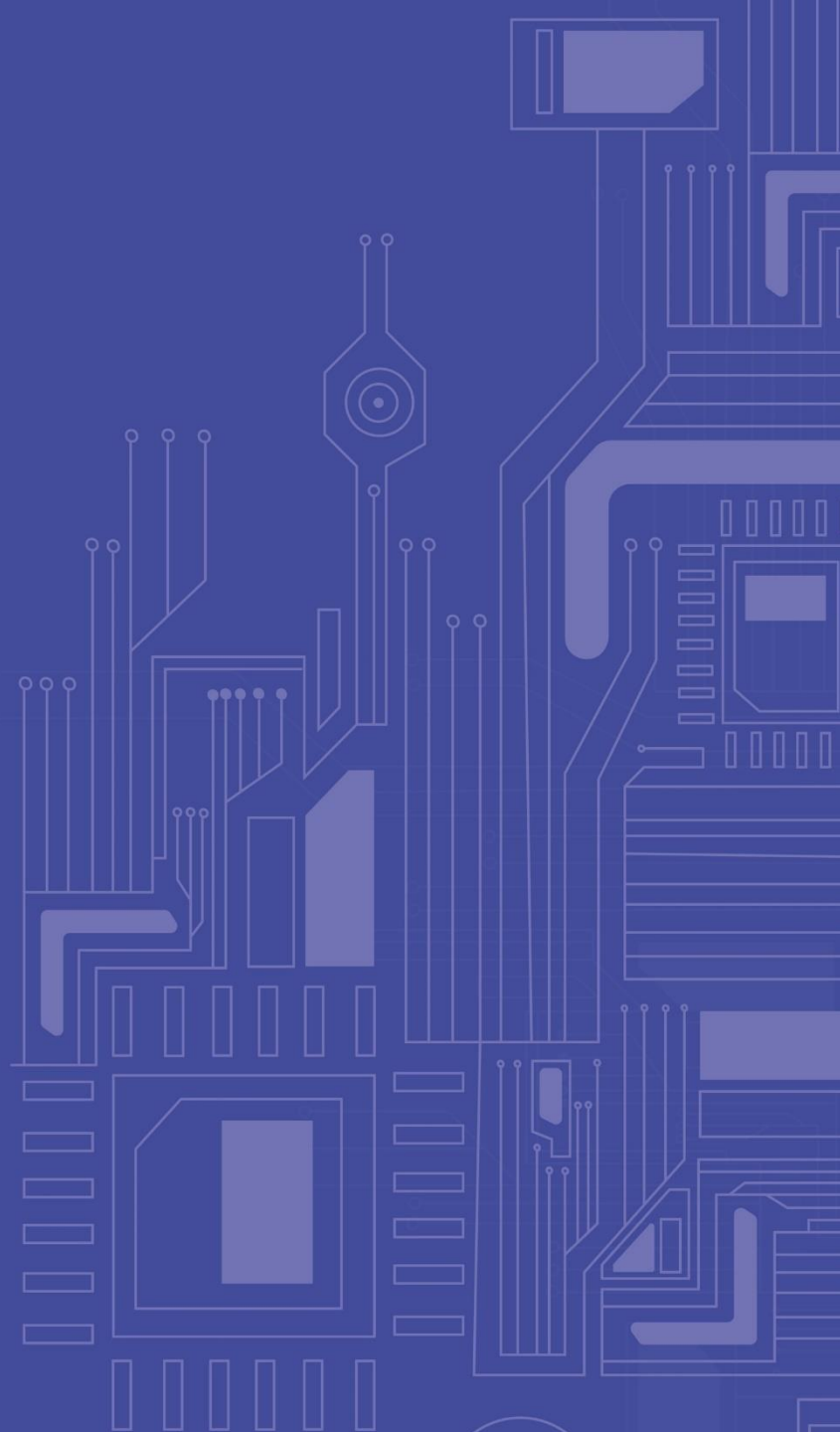
В языке отсутствовали средства явной синхронизации доступа к объектам БД со стороны параллельно выполняемых транзакций: с самого начала предполагалось, что необходимую синхронизацию неявно выполняет СУБД.

ИЗМЕНЕНИЯ В SQL

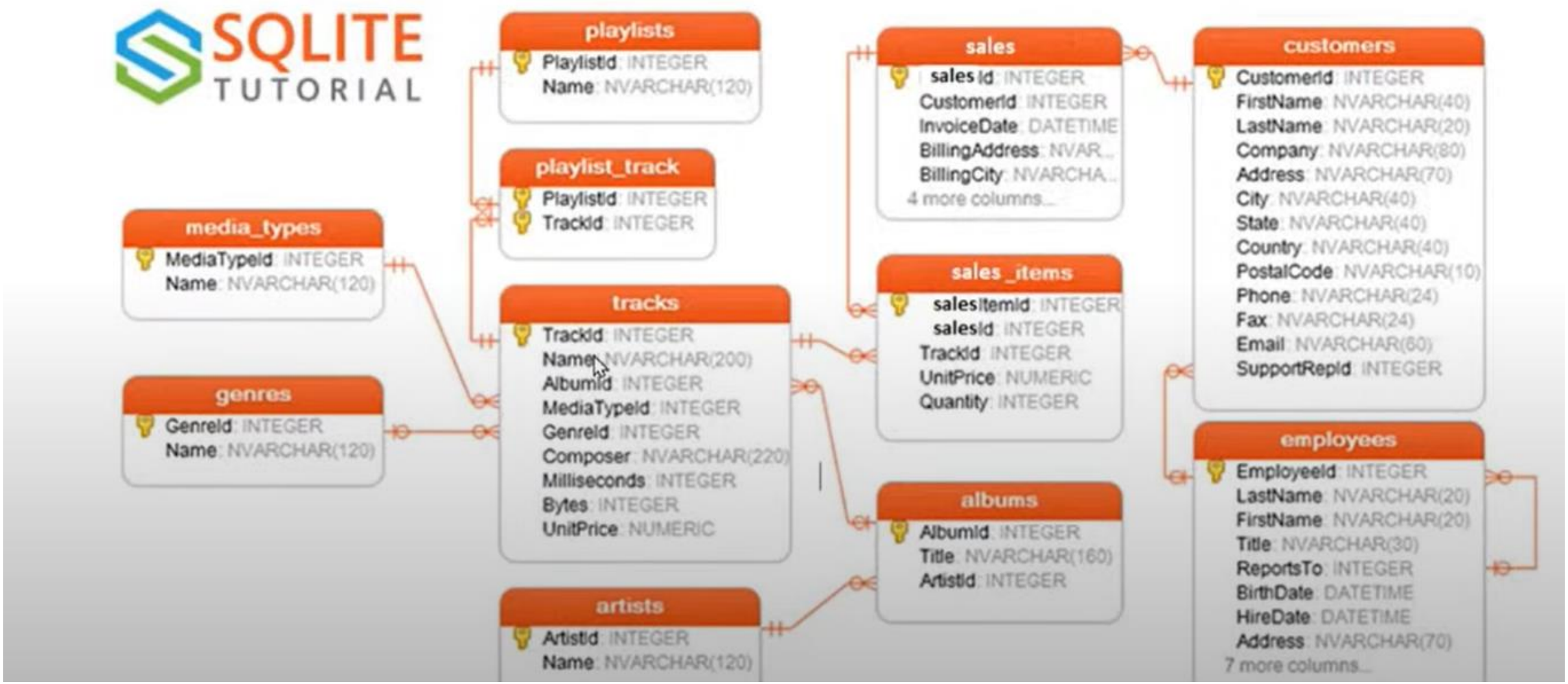
Год	Название	Другое название	Изменения
1999	SQL:1999	SQL3	Добавлена поддержка регулярных выражений, рекурсивных запросов, поддержка триггеров, базовые процедурные расширения, нескалярные типы данных и некоторые объектно-ориентированные возможности.
2003	SQL:2003		Введены расширения для работы с XML-данными, оконные функции (применяемые для работы с OLAP-базами данных), генераторы последовательностей и основанные на них типы данных.
2006	SQL:2006		Функциональность работы с XML-данными значительно расширена. Появилась возможность совместно использовать в запросах SQL и XQuery.
2008	SQL:2008		Улучшены возможности оконных функций, устранены некоторые неоднозначности стандарта SQL:2003


ВСЁ!

СЕМИНАР



СТРУКТУРА ДАННЫХ УЧЕБНОЙ БАЗЫ



ОБЗОР КОМАНД SQL ДЛЯ SQLITE3

```
DML:--Data Manipulation Language Язык опредения данных
        SELECT DCL
        INSERT
        UPDATE
        DELETE
        MERGE
DDL:--Data Defenition Language  Язык манипулирования данными
        CREATE
        ALTER
        DROP

DCL:--Data Control Language (DCL)  язык управления данными
        GRANT
        REVOKE
        DENY
TCL     --transaction Control Language  язык управления транзакциями
        BEGIN
        COMMIT
        ROLLBACK
```

ТИПЫ ЗАПРОСОВ (1/2)

```
SELECT *
from tracks
where UnitPrice>1 and UnitPrice<2 and name <> 'Torn'
```

Комментарии

```
--1.05 Отобрать все музыкальные треки  с ценой меньше 1
/*dfg
dfg
fbvbnagd
*/
```

Конкатенация

```
select LastName,FirstName,LastName || ', ' ||FirstName as fio
from employees
```

Обработка NULL

```
select    FirstName ||', '||   LastName as FIO,
         ifnull(Address,'') ||', '|| ifnull(City,'')||', '|| ifnull(State,'')||', '|| ifnull(Country,'')   as  full_adress
        ,FAX

from  customers
```

Поиск по слову

```
-- 1.09 Отобрать все песни в которых названии содержится слово night.

select *
from tracks
where   name like '%night%'
```

ТИПЫ ЗАПРОСОВ (2/2)

Ограничение на значение

```
select *
from tracks
where  TrackId in (1,2,3,4,5)
```

```
select *
from tracks
where   Composer='AC/DC'and unitPrice < 0.5  or  name like 'A%'
```

Помним о приоритетах, у AND он выше, чем у OR

Работа с датами

```
select *
from sales
where SalesDate >= date('2010-01-01')  and SalesDate < date('2011-01-01')
```

Группировка с выбором верхних двух

```
SELECT  date(salesDate,'start of year')  as yyyy , count(*)
  FROM sales
group by  date(salesDate,'start of year')
having count(*)>8
order by  yyyy limit 2
```

Округление дат

```
select SalesDate
    ,date(SalesDate)
    ,date(SalesDate,'start of month' )
    ,date(SalesDate,'start of year' )
    ,time(salesDate)
    ,SalesDate - date(SalesDate,'start of month' )
    ,strftime('%d', salesDate) as DD
    ,strftime('%m', salesDate) as MM
    ,strftime('%Y', salesDate) as YYYY

from sales
where --SalesDate >= date('2010-01-01')  and SalesDate < date('2011-01-01')
date(SalesDate,'start of year' ) = date('2010-01-01')
```

Группировка с отображением количества строк и среднего по каждой группе

```
select  country, count(*), avg(age)
from customers
group by country
```

ЗАДАЧИ

1. Покажите фамилию и имя клиентов из города Прага ?
2. Покажите фамилию и имя клиентов у которых имя начинается букву M ? Содержит символ "ch"?
3. Покажите название и размер музыкальных треков в Мегабайтах ?
4. Покажите фамилию и имя сотрудников кампании нанятых в 2002 году из города Калгари ?
5. Покажите фамилию и имя сотрудников кампании нанятых в возрасте 40 лет и выше?
6. Покажите покупателей-американцев без факса ?
7. Покажите канадские города в которые сделаны продажи в августе и сентябре месяце?
8. Покажите почтовые адреса клиентов из домена gmail.com ?
9. Покажите сотрудников которые работают в кампании уже 18 лет и более ?
10. Покажите в алфавитном порядке все должности в кампании ?
11. Покажите в алфавитном порядке Фамилию, Имя и год рождения покупателей ?
12. Сколько секунд длится самая короткая песня ?
13. Покажите название и длительность в секундах самой короткой песни ?
14. Покажите средний возраст клиента для каждой страны ?
15. Покажите Фамилии работников нанятых в октябре?
16. Покажите фамилию самого старого по стажу сотрудника в кампании ?

Примечание: Вам поможет документация https://www.sqlitetutorial.net/sqlite-date-functions/sqlite-date-function/