Выделяют следующие типы транзакций: плоские или **классические** транзакции, цепочечные транзакции и вложенные транзакции.

Плоские, или традиционные, транзакции, характеризуются четырьмя классическими свойствами: атомарности, согласованности, изолированности, долговечности (прочности) — ACID (Atomicity, Consistency, Isolation, Durability). Иногда традиционные транзакции называют ACID-транзакциями. Упомянутые выше свойства означают следующее:

Свойство **атомарности** (Atomicity) выражается в том, что транзакция выполнена в целом или не выполнена вовсе.

Свойство **согласованности** (Consistency) гарантирует, что по мере выполнения транзакций данные переходят из одного согласованного состояния в другое — транзакция не разрушает взаимной согласованности данных.

Свойство **изолированности** (Isolation) означает, что конкурирующие за доступ к базе данных транзакции физически обрабатываются последовательно, но для пользователей это выглядит так, как будто они выполняются параллельно.

Свойство **долговечности** (Durability) трактуется следующим образом: если транзакция завершена успешно, то те изменения в данных, которые были ею произведены, не могут быть потеряны ни при каких обстоятельствах (даже в случае последующих ошибок).

Возможны два варианта завершения транзакции: если все операторы выполнены успешно, транзакция фиксируется. **Фиксация** транзакции — это действие, обеспечивающее запись на диск изменений в базе данных, которые были сделаны в процессе выполнения транзакции. Если в процессе выполнения транзакции случилось нечто такое, что делает невозможным ее нормальное завершение, база данных должна быть возвращена в исходное состояние. **Откат** транзакции — это действие, обеспечивающее аннулирование всех изменений данных, которые были сделаны операторами SQL в теле текущей незавершенной транзакции. Каждый оператор в транзакции выполняет свою часть работы, но для успешного завершения всей работы в целом требуется безусловное завершение всех их операторов. **Группирование** операторов в транзакции сообщает СУБД, что вся эта группа должна быть выполнена как единое целое, причем такое выполнение должно поддерживаться автоматически.

В стандарте ANSI/ISO SQL определены модель транзакций и функции операторов COMMIT и ROLLBACK. Транзакция начинается с первого SQL-оператора, инициируемого пользователем или содержащегося в программе. Все последующие SQL-операторы составляют тело транзакции. Транзакция завершается одним из четырех возможных путей:

оператор COMMIT означает успешное завершение транзакции; его использование делает постоянными изменения, внесенные в базу данных в рамках текущей транзакции;

оператор ROLLBACK прерывает транзакцию, отменяя изменения, сделанные в базе данных в рамках этой транзакции; новая транзакция начинается непосредственно после использования ROLLBACK;

успешное завершение программы, в которой была инициирована текущая транзакция, означает успешное завершение транзакции (как будто был использован оператор COMMIT);

ошибочное завершение программы прерывает транзакцию (как будто был использован оператор ROLLBACK).

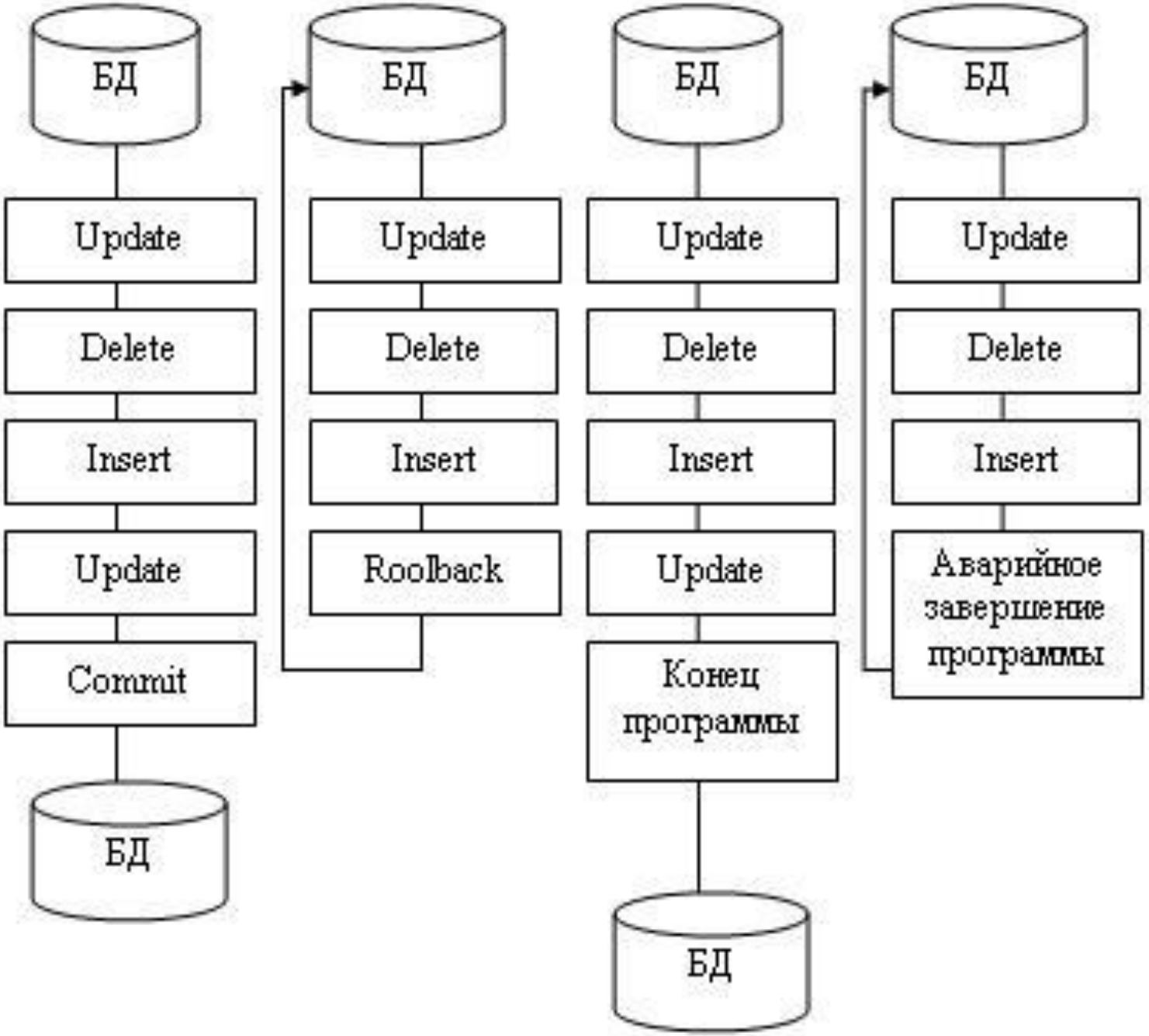
В дальнейшем в СУБД SYBASE была реализована расширенная модель транзакций, которая включает еще ряд дополнительных операций. В модели SYBASE используются следующие четыре оператора:

Оператор BEGIN TRANSACTION сообщает о начале транзакции. В отличие от модели в стандарте ANSI/ISO, где начало транзакции неявно задается первым оператором модификации данных, в модели SYBASE начало транзакции задается явно с помощью оператора начала транзакции.

Оператор COMMIT TRANSACTION сообщает об успешном завершении транзакции. Он эквивалентен оператору COMMIT в модели стандарта ANSI/ISO. Этот оператор, как и оператор COMMIT, фиксирует все изменения, которые производились в БД в процессе выполнения транзакции. Оператор SAVE TRANSACTION создает внутри транзакции точку сохранения, которая соответствует промежуточному состоянию БД, сохраненному на момент выполнения этого оператора.

В операторе SAVE TRANSACTION может стоять имя точки сохранения. Поэтому в ходе выполнения транзакции может быть запомнено несколько точек сохранения, соответствующих нескольким промежуточным состояниям.

-Оператор ROLLBACK имеет две модификации. Если этот оператор используется без дополнительного параметра, то он интерпретируется как оператор отката всей транзакции, то есть в этом случае он эквивалентен оператору отката ROLLBACK в модели ANSI/ISO. Если же оператор отката имеет параметр и записан в виде ROLLBACK B, то он интерпретируется как оператор частичного отката транзакции в точку сохранения B.


БД
Update
Delete
Insert
Update
Commit
БД
БД
Update
Delete
Insert
Roolback
БД
Update
Delete
Insert
Update
Конец программы
БД
БД
Update
Delete
Insert
Аварийное завершение программы

Журнал транзакций

Реализация в СУБД принципа сохранения промежуточных состояний, подтверждения или отката транзакции обеспечивается специальным механизмом, для поддержки которого создается некоторая системная структура, называемая *Журналом транзакций*

Общими принципами восстановления являются следующие:

результаты зафиксированных транзакций должны быть сохранены в восстановленном состоянии базы данных;

результаты незафиксированных транзакций должны отсутствовать в восстановленном состоянии базы данных.

Это, собственно, и означает, что восстанавливается последнее по времени согласованное состояние базы данных.

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

Индивидуальный откат транзакции. Этот откат должен быть применен в следующих случаях:

- стандартной ситуацией отката транзакции является ее явное завершение оператором ROLLBACK;
- аварийное завершение работы прикладной программы, которое логически эквивалентно выполнению оператора ROLLBACK, но физически имеет иной механизм выполнения;

-принудительный откат транзакции в случае взаимной блокировки при параллельном выполнении транзакций. В подобном случае для выхода из тупика данная транзакция может быть выбрана в качестве «жертвы» и принудительно прекращено ее выполнение ядром СУБД.

Восстановление после внезапной потери содержимого оперативной памяти (мягкий сбой). Такая ситуация может возникнуть в следующих случаях:

- при аварийном выключении электрического питания;
- при возникновении неустранимого сбоя процессора (например, срабатывании контроля оперативной памяти) и т. д. Ситуация характеризуется потерей той части базы данных, которая к моменту сбоя содержалась в буферах оперативной памяти.

Восстановление после поломки основного внешнего носителя базы данных (жесткий сбой). Эта ситуация при достаточно высокой надежности современных устройств внешней памяти может возникать сравнительно редко, но СУБД должна восстановить базу данных и в этом случае. Основой восстановления является архивная копия и журнал изменений базы данных.

.

Для восстановления согласованного состояния базы данных при **индивидуальном откате** транзакции нужно устранить последствия операторов модификации базы данных, которые выполнялись в этой транзакции.

Для восстановления непротиворечивого состояния БД **при мягком сбое** необходимо восстановить содержимое БД по содержимому журналов транзакций, хранящихся на дисках, **при жестком сбое** надо восстановить содержимое БД по архивным копиям и журналам транзакций, которые хранятся на неповрежденных внешних носителях.

Во всех трех случаях основой восстановления является избыточное хранение данных. Возможны два основных варианта ведения журнальной информации: для каждой транзакции поддерживать **отдельный локальный журнал изменений базы** данных этой транзакцией. Такие журналы называются локальными журналами. Они используются для индивидуальных откатов транзакций и могут поддерживаться в оперативной (правильнее сказать, в виртуальной) памяти.

Кроме того, поддерживается общий журнал изменений базы данных, используемый для восстановления состояния базы данных после мягких и жестких сбоев.

Это приводит к дублированию информации в локальных и общем журналах. Поэтому чаще используется второй вариант — **поддержание только общего журнала изменений базы** данных, который используется и при выполнении индивидуальных откатов.

Все транзакции имеют свои внутренние номера, поэтому в едином журнале транзакций фиксируются все изменения, проводимые всеми транзакциями.Каждая запись в журнале транзакций помечается номером транзакции, к которой она относится, и значениями атрибутов, которые она меняет. Для каждой транзакции в журнале фиксируется команда начала и завершения транзакции Для большей надежности журнал транзакций часто дублируется системными средствами коммерческих СУБД, именно поэтому объем внешней памяти во много раз превышает реальный объем данных, которые хранятся в хранилище.

Имеются два альтернативных варианта ведения журнала транзакций: протокол с отложенными обновлениями и протокол с немедленными обновлениями.

Ведение журнала по принципу отложенных изменений предполагает следующий механизм выполнения транзакций:

1.Когда транзакция T1 начинается, в протокол заносится запись

<T1 Begin transaction>

2.На протяжении выполнения транзакции в протоколе для каждой изменяемой записи записывается новое значение: <T1, ID_RECORO. атрибут, новое значение ... >. Здесь ID_RECORD — уникальный номер записи.

3.Если все действия, из которых состоит транзакция T1, успешно выполнены, то транзакция частично фиксируется и в протокол заносится <T1 COMMIT>.

4.После того как транзакция фиксирована, записи протокола, относящиеся к T1, используются для внесения соответствующих изменений в БД.

5.Если происходит сбой, то СУБД просматривает протокол и выясняет, какие транзакции необходимо переделать. Транзакцию T1 необходимо переделать, если протокол содержит обе записи <T1 BEGIN TRANSACTION и <T1 COMMIT>. БД может находиться в несогласованном состоянии, однако все новые значения измененных элементов данных содержатся в протоколе, и это требует повторного выполнения транзакции. Для этого используется системная процедура REDOQ, которая заменяет все значения элементов данных на новые, просматривая протокол в прямом порядке.

6.Если в протоколе не содержится команда фиксации транзакции COMMIT, то никаких действий проводить не требуется, а транзакция запускается заново.

Альтернативный механизм с немедленным выполнением предусматривает внесение изменений сразу в БД, а в протокол заносятся не только новые, но и все старые значения изменяемых атрибутов, поэтому каждая запись выглядит <T1, ID_RECORD, атрибут новое значение старое значение ...>.

При этом запись в журнал предшествует непосредственному выполнению операции над БД. Когда транзакция фиксируется, то есть встречается команда <T1 COMMIT> и она выполняется, то все изменения оказываются уже внесенными в БД и не требуется никаких дальнейших действий по отношению к этой транзакции.

При откате транзакции выполняется системная процедура UNDO(), которая возвращает все старые значения в отмененной транзакции, последовательно проходя по протоколу начиная с команды BEGIN TRANSACTION. Для восстановления при сбое используется следующий механизм:

Если транзакция содержит команду начала транзакции, но не содержит команды фиксации с подтверждением ее выполнения, то выполняется последовательность действий как при откате транзакции, то есть восстанавливаются старые значения.

Если сбой произошел после выполнения последней команды изменения БД, но до выполнения команды фиксации, то команда фиксации выполняется, а с БД никаких изменений не происходит. Работа происходит только на уровне протокола.

Однако следует отметить, что проблемы восстановления выглядят гораздо сложнее приведенных ранее алгоритмов, с учетом того, что изменения как в журнал, так и в БД заносятся не сразу, а буферируются. Если бы запись об изменении базы данных, которая должна поступить в журнал при выполнении любой операции модификации базы данных, реально немедленно записывалась бы во внешнюю память, это привело бы к существенному замедлению работы системы. Поэтому записи в журнале тоже буферизуются: при нормальной работе очередная страница выталкивается во внешнюю память журнала только при полном заполнении записями. Проблема состоит в выработке некоторой общей политики выталкивания **Основным принципом согласованной политики выталкивания буфера журнала и буферов страниц базы данных является то, что запись об изменении объекта базы данных должна попадать во внешнюю память журнала раньше, чем измененный объект оказывается во внешней памяти базы данных.** Соответствующий протокол журнализации называется Write Ahead Log (WAL) — «пиши сначала в журнал»

Если во внешней памяти базы данных находится некоторый объект базы данных, по отношению к которому выполнена операция модификации, то во внешней памяти журнала обязательно находится запись, соответствующая этой операции.

Дополнительное условие на выталкивание буферов накладывается тем требованием, что каждая успешно завершившаяся транзакция должна быть реально зафиксирована во внешней памяти. Какой бы сбой не произошел, система должна восстановить состояние базы данных, содержащее результаты всех зафиксированных к моменту сбоя транзакций. 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.

Операции восстановления базы данных в различных ситуациях

Индивидуальный откат транзакции выполняется следующим образом:

Выбирается очередная запись из списка данной транзакции.

Выполняется противоположная по смыслу операция: вместо операции INSERT выполняется соответствующая операция DELETE, вместо операции DELETE вы полняется INSERT и вместо прямой операции UPDATE обратная операция UPDATE, восстанавливающая предыдущее состояние объекта базы данных.

Любая из этих обратных операций также заносится в журнал. Собственно, для индивидуального отката это не нужно, но при выполнении индивидуального отката транзакции может произойти мягкий сбой, при восстановлении после которого потребуется откатить такую транзакцию, для которой не полностью выполнен индивидуальный откат.

При успешном завершении отката в журнал заносится запись о конце транзакции. С точки зрения журнала такая транзакция является зафиксированной

Восстановление после мягкого сбоя

К числу основных проблем восстановления после мягкого сбоя относится то, что одна логическая операция изменения базы данных может изменять несколько **физических блоков** базы данных, например, страницу данных и несколько страниц индексов. Страницы базы данных буферизуются в оперативной памяти и выталкиваются независимо. Несмотря на протокол WAL, после мягкого сбоя набор страниц внешней памяти базы данных может оказаться несогласованным, то есть часть страниц внешней памяти соответствует объекту до изменения, часть — после изменения

Состояние внешней памяти базы данных называется **физически согласованным,** если наборы страниц всех объектов согласованы, то есть соответствуют состоянию объекта либо до его изменения, либо после изменения. Точки физической согласованности базы данных — моменты времени, в которые во внешней памяти содержатся согласованные результаты операций, завершившихся до соответствующего момента времени

Назовем такие точки **tpc** (time of physical consistency) — **точками физического согласования.**

Тогда к моменту мягкого сбоя возможны следующие состояния транзакций:

транзакция успешно завершена, то есть выполнена операция подтверждения транзакции COMMIT и для всех операций транзакции получено подтверждение ее выполнения во внешней памяти;

транзакция успешно завершена, но для некоторых операций не получено подтверждение их выполнения во внешней памяти;

транзакция получила и выполнила команду отката ROLLBACK;

транзакция не завершена? Для восстановления базы данных в момент tpc используются два основных подхода: подход, основанный на использовании **теневого** механизма, и подход, в котором применяется **журнализация постраничных изменений** базы данных.

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 оперативной памяти) изменяется, а теневая — сохраняется 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 оперативную память **теневую таблицу** отображения. **Теневой механизм** используется следующим образом. Периодически выполняются операции установления точки физической согласованности базы данных (checkpoints). Для этого все логические операции завершаются, все буферы оперативной памяти, содержимое которых не соответствует содержимому соответствующих страниц внешней памяти, выталкиваются. Теневая таблица отображения файлов базы данных заменяется на текущую (правильнее сказать, текущая таблица отображения записывается на место теневой).

Восстановление к tpc происходит мгновенно: текущая таблица отображения заменяется на теневую (при восстановлении просто считывается 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.

Для выполнения более слабого требования наряду с логической журнализацией операций изменения базы данных производится журнализация постраничных изменений. Первый этап восстановления после мягкого сбоя состоит в постраничном откате незакончившихся 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

Имеются два метода решения проблемы. При использовании **первого метода 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 увеличивают журнал.

Поэтому все более популярным становится поддержание отдельного (короткого) журнала постраничных изменений. Такая техника применяется, например, в известном продукте Informix Online

Восстановление после жесткого сбоя

Понятно, что для восстановления последнего согласованного состояния базы данных после жесткого сбоя журнала изменений базы данных явно недостаточно. Основой восстановления в этом случае являются журнал и архивная копия базы данных.

Восстановление начинается с **обратного копирования** базы данных из архивной копии. Затем для всех закончившихся транзакций выполняется redo, то есть операции **повторно выполняются в прямом порядке.**

Более точно, происходит следующее:

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

При утрате журнала единственным способом восстановления базы данных является возврат к архивной копии.

Архивировать базу данных можно при переполнении журнала. В журнале вводится так называемая «желтая зона», при достижении которой образование новых транзакций временно блокируется. Когда все транзакции закончатся и, следовательно, база данных придет в согласованное состояние, можно производить ее архивацию, после чего начинать заполнять журнал заново.

Можно выполнять архивацию базы данных реже, чем переполняется журнал. При переполнении журнала и окончании всех начатых транзакций можно архивировать сам журнал. Поскольку такой архивированный журнал, по сути дела, требуется только для воссоздания архивной копии базы данных, журнальная информация при архивации может быть сжата.

Параллельное выполнение транзакций

СУБД должна не только корректно выполнять индивидуальные транзакции и восстанавливать согласованное состояние БД после сбоев, но и обеспечить корректную параллельную работу всех пользователей над одними и теми же данными.

Основные проблемы, которые возникают при параллельном выполнении транзакций, делятся условно на 4 типа:

Пропавшие изменения. Эта ситуация может возникать, если две транзакции одновременно изменяют одну и ту же запись в БД.

Проблемы промежуточных данных

Проблемы несогласованных данных. Эта ситуация возникла потому, что приложение первого оператора смогло изменить кортеж с данными, который уже прочитало приложение второго оператора.

Проблемы строк-призраков (строк-фантомов).

Для того чтобы избежать подобных проблем, требуется выработать некоторую процедуру согласованного выполнения параллельных транзакций. Эта процедура должна удовлетворять следующим правилам:

В ходе выполнения транзакции пользователь видит только согласованные данные. Пользователь не должен видеть несогласованных промежуточных данных.

Когда в БД две транзакции выполняются параллельно, то СУБД гарантированно поддерживает принцип независимого выполнения транзакций1.

Такая процедура называется сериализацией транзакций. Фактически она гарантирует, что каждый пользователь (программа), обращаясь к базе данных, работает с ней так, как будто не существует других пользователей (программ), одновременно с ним обращающихся к тем же данным.

Для поддержки параллельной работы транзакций строится специальный план.

- План выполнения набора транзакций называется сериальным, если результат совместного выполнения транзакций эквивалентен результату некоторого последовательного выполнения этих же транзакций. Наиболее распространенным механизмом, который используется коммерческими СУБД для реализации на практике сериализации транзакций является **механизм блокировок**. Самый простой вариант — это блокировка объекта на все время действия транзакции. В общем случае на момент выполнения транзакция получает монопольный доступ к объектам БД, с которыми она работает до момента окончания транзакции. Типы конфликтов между двумя параллельными транзакциями:
- W-W — транзакция 2 пытается изменять объект, измененный незакончившейся транзакцией 1;
- R-W — транзакция 2 пытается изменять объект, прочитанный незакончившейся транзакцией 1;
- W-R — транзакция 2 пытается читать объект, измененный незакончившейся транзакцией 1.

Блокировки, называемые также синхронизационными захватами объектов, могут быть применены к разному типу объектов. Наибольшим объектом блокировки может быть **вся БД**, однако этот вид блокировки сделает БД недоступной для всех приложений, которые работают с данной БД. Следующий тип объекта блокировки — это **таблицы**.. Этот вид блокировки предпочтительнее предыдущего, потому что позволяет параллельно выполнять транзакции, которые работают с другими таблицами.

В ряде СУБД реализована блокировка на уровне страниц. В этом случае СУБД блокирует только отдельные страницы на диске, когда транзакция обращается к ним. Этот вид блокировки еще более мягок и позволяет разным транзакциям работать даже с одной и той же таблицей, если они обращаются к разным страницам данных.

В некоторых СУБД возможна блокировка на уровне строк, однако такой механизм блокировки требует дополнительных затрат на поддержку этого вида блокировки.

Для повышения параллельности выполнения транзакций используется комбинирование разных типов синхронизационных захватов.Рассматривают два типа блокировок (синхронизационных захватов):

совместный режим блокировки — нежесткая, или разделяемая, блокировка, обозначаемая как S (Shared). Этот режим обозначает разделяемый захват объекта и требуется для выполнения операции чтения объекта. Объекты, заблокированные таким образом, не изменяются в ходе выполнения транзакции и доступны другим транзакциям также, но только в режиме чтения;

монопольный режим блокировки — жесткая, или эксклюзивная, блокировка, обозначаемая как X (eXclusive). Данный режим блокировки предполагает монопольный захват объекта и требуется для выполнения операций занесения, удаления и модификации. Объекты, заблокированные данным типом блокировки, фактически остаются в монопольном режиме обработки и недоступны для других транзакций до момента окончания работы данной транзакции.

Захваты объектов несколькими транзакциями по чтению совместимы, то есть нескольким транзакциям допускается читать один и тот же объект, захват объекта одной транзакцией по чтению не совместим с захватом другой транзакцией того же объекта по записи, и захваты одного объекта разными транзакциями по записи не совместимы

		Транзакция B		
		Разблокирована	Нежесткая блокировка	Жесткая блокировка
Транзакция A	Разблокирована	Да	Да	Да
	Нежесткая блокировка	Да	Да	Нет
	Жесткая блокировка	Да	Нет	Нет

К сожалению, применения разных типов блокировок приводит к проблеме **тупиков**. Проблема тупиков возникла при выполнения параллельных процессов в операционных средах и связана с управлением разделяемыми (совместно используемыми) ресурсами.

В большинстве коммерческих СУБД существует механизм обнаружения таких тупиковых ситуаций.

Основой обнаружения тупиковых ситуаций является построение (или постоянное поддержание) графа ожидания транзакций.

Граф ожидания — это направленный граф, в вершинах которого расположены имена транзакций

Разрушение тупика начинается с выбора в цикле транзакций так называемой транзакции-жертвы, то есть транзакции, которой решено пожертвовать, чтобы обеспечить возможность продолжения работы других транзакций. Критерием выбора является **стоимость транзакции**; жертвой выбирается самая дешевая транзакция

Стоимость транзакции определяется на основе многофакторной оценки, в которую с разными весами входят время выполнения, число накопленных захватов, приоритет.

После выбора транзакции-жертвы выполняется откат транзакции, который может носить полный или частичный характер Такое насильственное устранение тупиковых ситуаций является нарушением принципа изолированности пользователей.

Заметим, что в централизованных системах стоимость построения графа ожидания сравнительно невелика, но она становится слишком большой в распределенных СУБД, Поэтому в таких системах обычно используются другие методы сериализации транзакций. - синхронизационные захваты объектов, произведенные по инициативе транзакции, можно снимать только при ее завершении. Это требование порождает двухфазный протокол синхронизационных захватов — 2PL(two phase lock) или 2PC (two phase commit). В соответствии с этим протоколом выполнение транзакции разбивается на две фазы:

-первая фаза транзакции — накопление захватов;

-вторая фаза (фиксация или откат) — освобождение захватов.

В языке SQL введен оператор явной блокировки таблицы, который позволяет точно задать тип блокировки для всей таблицы. Синтаксис операции блокировки имеет вид:

LOCK TABLE имя_таблицы IN {SHARED | EXCLUSIVE} MODE

Имеет смысл **блокировать таблицу полностью**, когда выполняется операция множественной модификации одной таблицы, то есть когда в ней изменяется большое количество строк. Эта операция иногда называется **пакетным** обновлением.

Конечно, у блокировки таблицы есть тот недостаток, что все остальные транзакции должны ждать окончания обновления таблицы. Но режим пакетного обновления одной таблицы работает достаточно быстро, и общая производительность выполнения множества транзакций может даже повыситься в этом случае.

Уровни изолированности пользователей

При соблюдении двухфазного протокола синхронизационных захватов действительно обеспечивается полная сериализация транзакций. Однако иногда приложению, которое выполняет транзакцию, не сколько важны точные данные, сколько скорость выполнения запросов

В системах поддержки принятия решении но электронным торгам важно просто иметь представление об общей картине торгов, на основании которого принимается решение об повышении или снижении ставок и т. д. Для смягчения требований сериализации транзакций вводится понятие уровня изолированности пользователя.

Всего введено 4 уровня изолированности пользователей. Самый **высокий** уровень изолированности соответствует протоколу сериализации транзакций, это уровень SERIALIZABLE. Этот уровень обеспечивает полную изоляцию транзакций и полную корректную обработку параллельных транзакций.

Следующий уровень изолированности называется **уровнем подтвержденного чтения** — REPEATABLE READ. На этом уровне транзакция не имеет доступа к промежуточным или окончательным результатам других транзакций, поэтому такие проблемы, как пропавшие обновления, промежуточные или несогласованные данные, возникнуть не могут. **Следующий уровень** называется READ COMMITED. На этом уровне изолированности транзакция не имеет доступа к промежуточным результатам других транзакций. При этом уровне изолированности транзакция не может обновлять строку, уже обновленную другой транзакцией Самый **низкий** уровень изолированности называется уровнем неподтвержденного, или грязного, чтения.

Он обозначается как READ UNCOMMITED

В стандарте SQL2 существует оператор задания уровня изолированности выполнения транзакции. Он имеет следующий синтаксис:

SET TRANSACTION IZGLATION LEVEL [{SERIALIZABLE |

REPEATABLE READ |

READ COMMITED |

READ UNCOMMITED}] [{READ WRITE |

READ ONLY }]

Дополнительно в этом операторе может быть указано, операции какого типа выполняются в транзакции. По умолчанию предполагается уровень SERIALIZABLE. Если задан уровень READ UNCOMMITED, то допустимы только операции чтения в транзакции, поэтому в этом случае нельзя установить операции

READ WRITE

Соответствие уровней изолированности транзакций и проблем, возникающих при параллельном выполнении транзакций

L1\L2	X	S	IX	IS	SIX
Нет блокировки	Да	Да	Да	Да	Да
X	Нет	Нет	Нет	Нет	Нет
S	Нет	Да	Нет	Да	Нет
IX	Нет	Нет	Да	Да	Мет
IS	Нет	Да	Да	Да	Да
SIX	Нет	Нет	Нет	Да	Нет

В разных коммерческих СУБД могут быть реализованы не все уровни изолированности, это необходимо выяснить в технической документации

Гранулированные синхронизационные захваты

Мы уже говорили, что объектами блокирования могут быть объекты разного уровня, начиная с целой БД и заканчивая кортежем.

Понятно, что чем крупнее объект синхронизационного захвата (неважно, какой природы этот объект — логический или физический), тем меньше синхронизационных захватов будет поддерживаться в системе, и при этом, соответственно, будут меньшие накладные расходы. При использовании для захватов крупных объектов возрастает вероятность конфликтов транзакций и тем самым уменьшается допускаемая степень их параллельного выполнения. Фактически при укрупнении объекта синхронизационного захвата мы умышленно огрубляем ситуацию и видим конфликты в тех ситуациях, когда на самом деле конфликтов нет

В большинстве современных систем используются покортежные, то есть построковые синхронизационные захваты.

Синхронизационные захваты могут запрашиваться по отношению к объектам разного уровня: файлам, отношениям и кортежам. Требуемый уровень объекта определяется тем, какая операция выполняется. Объект любого уровня может быть захвачен в режиме S (разделяемом) или X (монопольном). Вводится специальный протокол гранулированных захватов и определены новые типы захватов: перед захватом объекта в режиме S или X соответствующий объект более высокого уровня должен быть захвачен в режиме IS, IX или SIX.

IS (Intented for Shared lock, предваряющий разделяемую блокировку) по отношению к некоторому составному объекту 0 означает намерение захватить некоторый входящий в 0 объект в совместном режиме. При намерении читать кортежи из отношения R это отношение должно быть захвачено в режиме IS

IX (Intented for exclusive lock, предваряющий жесткую блокировку) - означает намерение захватить некоторый входящий в 0 объект в монопольном режиме. При удалении кортежа из отношения R это отношение должно быть захвачено Б режиме IX (а до этого в таком же режиме должен быть захвачен файл).

SIX (Shared, Intented for eXclusive lock, разделяемая блокировка объекта, предваряющая дальнейшие жесткие блокировки его составляющих) по отношению к некоторому составному объекту О означает совместный захват всего этого объекта с намерением впоследствии захватывать какие-либо входящие в него объекты в монопольном режиме.

Полная таблица совместимости захватов

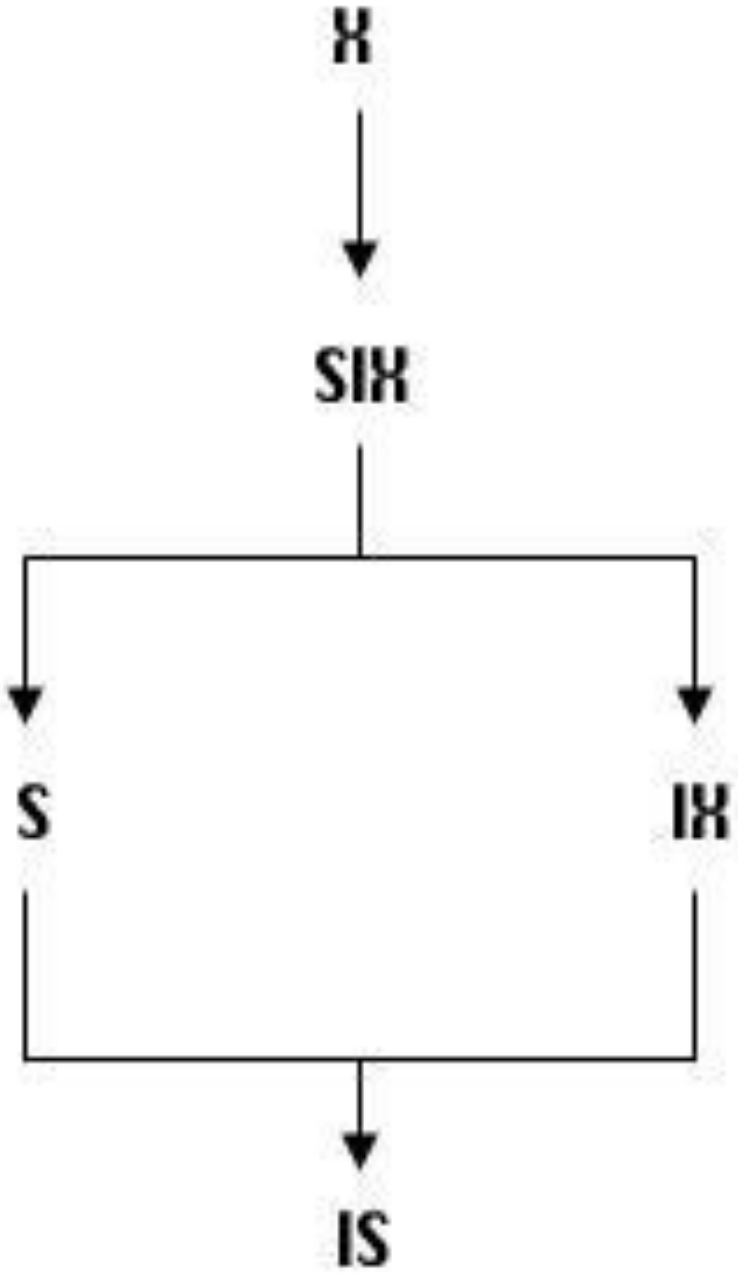
	L1\L2	X	S	IX	IS	SIX	
	Нет блокировки	Да	Да	Да	Да	Да	
	X	Нет	Нет	Нет	Нет	Нет	
	S	Нет	Да	Нет	Да	Нет	
	IX	Нет	Нет	Да	Да	Мет	
	IS	Нет	Да	Да	Да	Да	
	SIX	Нет	Нет	Нет	Да	Нет	

Протокол гранулированных захватов требует соблюдения следующих правил:

Прежде чем транзакция установит S-блокировку на данный кортеж, она должна установить блокировку IS или другую, более сильную блокировку на отношение, в котором содержится данный кортеж.

Прежде чем транзакция установит X-блокировку на данный кортеж, она должна установить IX-блокировку или другую более сильную блокировку на отношение, в которое входит кортеж

Диаграмма приоритета блокировок различных типов



Предикатные синхронизационные захваты

Несмотря на привлекательность метода гранулированных синхронизационных захватов, следует отметить, что он не решает проблему фантомов (если, конечно, не ограничиться использованием захватов отношений в режимах S и X).

Известно, что проблема фантомов не возникает, если объектом блокировки является целое отношение. Именно это свойство и послужило основой разработки метода предикатных синхронизационных захватов. В этом случае мы рассматриваем захват отношения — простой и частный случай предикатного захвата.

Суть этого метода — оценить множество кортежей, которое связано с той или иной транзакций, и если эти два множества, относящиеся к одному отношению, не пересекаются, то две транзакции могут оперировать ими параллельно без взаимной блокировки, а результаты выполнения обеих транзакций будут корректными.

Поскольку любая операция над реляционной базой данных задается некоторым условием (то есть в ней указывается не конкретный набор объектов базы данных, над которыми нужно выполнить операцию, а условие, которому должны удовлетворять объекты этого набора), идеальным выбором было бы требовать синхронизационный захват в режиме S или X именно этого условия. Но если посмотреть на общий вид условий, допускаемых, например, в языке SQL, то становится абсолютно непонятно, как определить совместимость двух предикатных захватов. Ясно, что без этого использовать предикатные захваты для синхронизации транзакций невозможно, а в общей форме проблема неразрешима.

Эта проблема сравнительно легко решается для случая простых условий. Будем называть простым условием конъюнкцию простых предикатов, имеющих вид:

имя-атрибута { операция сравнения } значение

Здесь операция сравнения: =, >, <

В типичных СУБД, поддерживающих двухуровневую организацию (языковой уровень и уровень управления внешней памяти), в интерфейсе подсистем управления памятью (которая обычно заведует и сериализацией транзакций) допускаются только просты условия. Подсистема языкового уровня производит компиляцию исходного оператора со сложным условием в последовательность обращений к ядру СУБД, в каждом из которых содержатся только простые условия. Следовательно, в случае типовой организации реляционной СУБД простые условия можно использовать как основу предикатных захватов.

Для простых условий совместимость предикатных захватов легко определяется на основе следующей геометрической интерпретации Пусть R — отношение с атрибутами a1, a2, ..., an, а m1,m2, ..., mn – множества допустимых значений a1, a2, ..., an соответственно (все эти множества — конечные). Тогда можно сопоставить R конечное n-мерное пространство возможных значений кортежей R. Любое простое условие «вырезает» m-мерный прямоугольник в этом пространстве (m <= n).

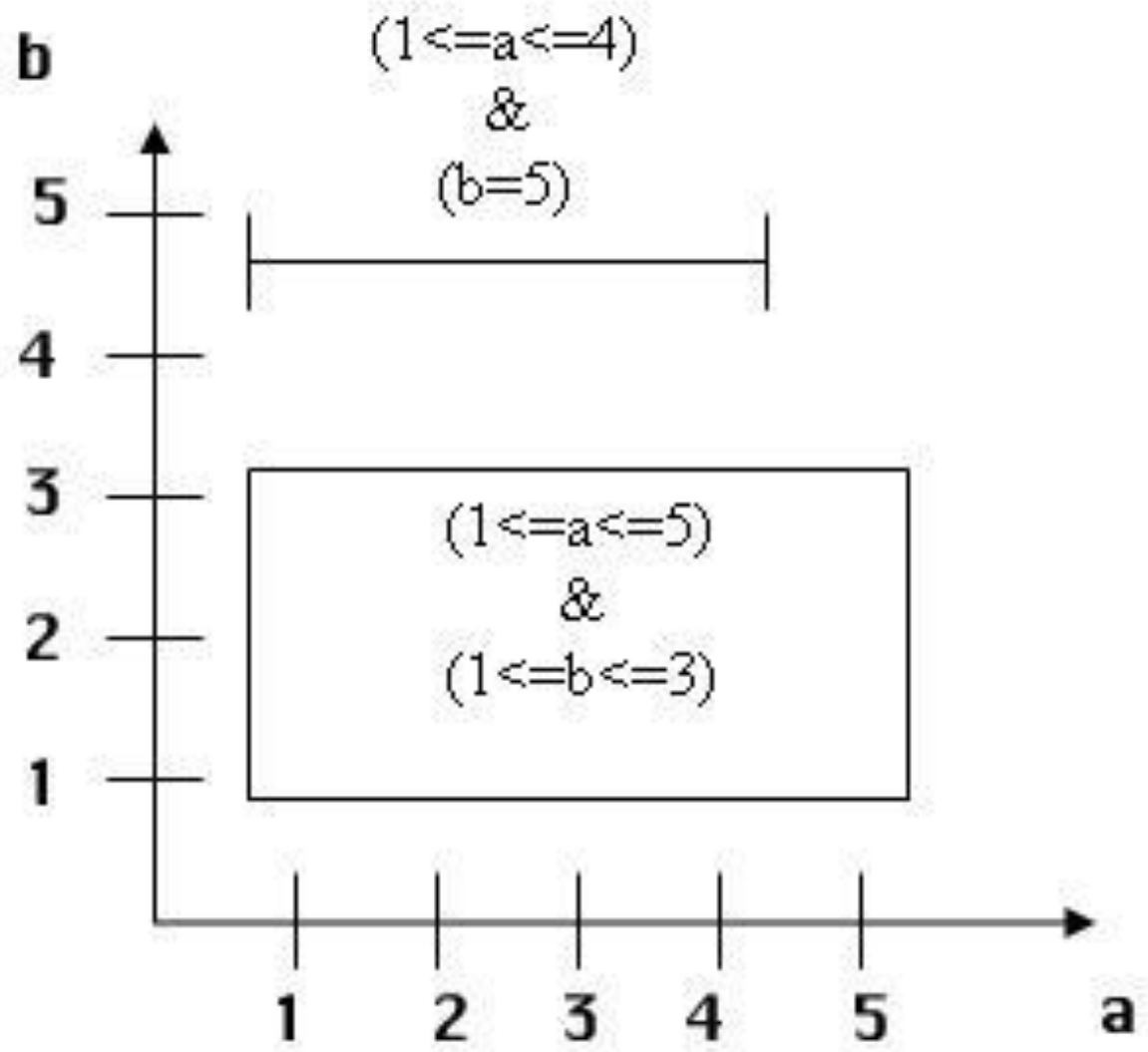
Тогда S-X, X-S, X-X предикатные захваты от разных транзакций совместимы, если соответствующие прямоугольники не пересекаются.

Это иллюстрируется следующим примером, показывающим, что в каких бы режимах не требовала транзакция 1 захвата условия (1<=a<=4) & (b=5), а транзакция 2 — условия (1<=a<=5) & (1<=b<=3), эти захваты всегда совместимы.

Пример: (n = 2)

Заметим, что предикатные захваты простых условий описываются таблицами, немногим отличающимися от таблиц традиционных синхронизаторов.

Области действия предикатных захватов



Метод временных меток

Альтернативный метод сериализации транзакций, хорошо работающий в условиях редких конфликтов транзакций и не требующий построения графа ожидания транзакций, основан на использовании временных меток.

Основная идея метода состоит в следующем: если транзакция T1 началась раньше транзакции T2, то система обеспечивает такой режим выполнения, как если бы T1 была целиком выполнена до начала T2.

Для этого каждой транзакции T предписывается временная метка t, соответствующая времени начала T. При выполнении операции над объектом r транзакция T помечает его своей временной меткой и типом операции (чтение или изменение).

- Перед выполнением операции над объектом г транзакция T1 выполняет следующие действия:
- Проверяет, не закончилась ли транзакция T, пометившая этот объект. Если T закончилась, T1 помечает объект и выполняет свою операцию.

Если транзакция T не завершилась, то T1 проверяет конфликтность операций. Если операции неконфликтны, при объекте r остается или проставляется временная метка с меньшим значением, и транзакция T1 выполняет свою операцию. Если операции T1 и T конфликтуют,,то если t(T) > t(T1) (то есть транзакция T является более «молодой», чем T1), производится откат T и T1 продолжает работу.

Если же t(T) < t(T1) (T «старше» T1), то T1 получает новую временную метку и начинается заново.

К недостаткам метода временных меток относятся потенциально более частые откаты транзакций, чем в случае использования синхронизационных захватов. Это связано с тем, что конфликтность транзакций определяется более грубо.

Кроме того, в распределенных системах не очень просто вырабатывать глобальные временные метки с отношением полного порядка. Но в распределенных системах эти недостатки окупаются тем, что не нужно распознавать тупики, а как мы уже отмечали, построение графа ожидания в распределенных системах стоит очень дорого.