Лекция 7. Реализация многопользовательского режима. Транзакции.

Важнейшей целью создания БД является организация параллельного доступа многих пользователей к общим данным, используемым ими совместно. Обеспечить параллельный доступ несложно, если все пользователи будут только читать данные, помещенные в базу. В этом случае работа каждого из них не оказывает никакого влияния на работу остальных пользователей. Но если два или больше пользователей одновременно обращаются к БД и хотя бы один из них имеет целью обновить хранимую в базе информацию, возможно взаимное влияние процессов друг на друга, способное привести к потере согласованности данных.

Существуют два подхода решения проблем совместного доступа к общим данным: **установка блокировок и управление параллельностью выполнения транзакций**.

В первом случае на время выполнения какой-либо операции в БД доступ к используемому объекту со стороны других пользователей временно запрещается или ограничивается.

Во втором случае при одновременном доступе к одним и тем же данным операции манипулирования данными, заключенные в рамки транзакции, чередуются, и таким образом достигается их параллельное выполнение. Однако, несмотря на то, что каждая из транзакций сама по себе может быть выполнена вполне корректно, подобное чередование операций способно приводить к неверным результатам, из-за чего целостность и согласованность базы данных будет нарушена. В большинстве случаев эти два подхода применяются совместно.

Различают **оптимистический и пессимистический подходы** в управлении **параллельностью выполнения транзакций**.

Пессимистический подход обеспечивает более высокий уровень безопасности, поскольку откладывается выполнение любых транзакций, потенциально способных войти в конфликт с другими транзакциями.

Оптимистический подход разрешает асинхронное выполнение транзакций в предположении, что вероятность конфликта невысока. Проверка на наличие конфликта откладывается до завершения транзакции и ее фиксации в БД.

Блокировки

Для **организации многопользовательского доступа** в СУБД применяется **механизм блокировок**. Суть **блокировки** состоит в том, что на время выполнения какой-либо операции в БД доступ к используемому объекту со стороны других пользователей **временно запрещается или ограничивается**. Например, при копировании таблицы она блокируется от изменения, хотя и разрешено просматривать ее содержимое.
Блокировки устанавливаются СУБД автоматически, но возможно и их явное определение.

Рассмотрим **четыре вида блокировок**, перечисленные в порядке убывания строгости ограничений на возможные действия.

1. **Полная блокировка**. Означает полное запрещение любых операций над объектами БД. Обычно применяется при изменении структуры объектов.
2. **Блокировка от записи**. Накладывается в случаях, когда можно читать данные, но не изменять их. Изменение структуры данных также запрещается.
3. **Предохраняющая блокировка от записи**. Предохраняет объект от наложения на него со стороны других операций полной блокировки либо блокировки от записи. Этот вид блокировки позволяет тому, кто раньше «захватил» объект, успешно завершить модификацию объекта. Такая блокировка обычно применяется при совместном изменении данных несколькими пользователями.
4. **Предохраняющая полная блокировка**. Предохраняет объект от наложения на него со стороны других операций полной блокировки. Обеспечивает максимальный уровень совместного использования объектов. Используется, например, для обеспечения одновременного просмотра несколькими пользователями одной таблицы. Тогда никому не будет позволено изменить структуру общей таблицы.

При **одновременном** выполнении различных операций над одним и тем же объектом производится попытка их **совмещения**.

Совмещение возможно тогда, когда совместимыми сказываются блокировки, накладываемые конкурирующими операциями. В отношении перечисленных выше блокировок действуют следующие **правила совмещения**:

• полная блокировка несовместима с другими блокировками, т. е. при наличии полной блокировки над объектом нельзя производить другие операции, накладывающие со своей стороны любую блокировку;
• блокировка от записи совместима с аналогичной блокировкой и предохраняющей полной блокировкой;
• предохраняющая блокировка от записи совместима с обоими видами предохраняющих блокировок;
• предохраняющая полная блокировка совместима со всеми видами блокировок, кроме полной.

Транзакции

Понятие **транзакции** имеет непосредственную связь с понятием **целостности БД**. В системах с развитыми средствами контроля целостности каждая транзакция **начинается при целостном** состоянии БД и должна оставить это состояние **целостным после своего завершения**. Несоблюдение этого условия приводит к тому, что вместо фиксации результатов транзакции происходит ее откат, и БД остается в таком состоянии, в котором находилась к моменту начала транзакции, т. е. в целостном состоянии.

Часто БД могут обладать такими ограничениями целостности, которые просто невозможно не нарушить, выполняя только одну операцию изменения данных. Поэтому для поддержания подобных ограничений целостности допускается их нарушение внутри транзакции с тем условием, чтобы к моменту завершения транзакции условия целостности были соблюдены.

Различаются два вида **ограничений целостности: немедленно проверяемые и откладываемые**.

К **немедленно проверяемым** ограничениям целостности относятся такие ограничения, проверку которых бессмысленно или даже невозможно откладывать. Немедленно проверяемые ограничения целостности соответствуют уровню отдельных операторов языкового уровня СУБД. При их нарушениях не производится откат транзакции, а лишь отвергается соответствующий оператор.

Откладываемые ограничения целостности -это ограничения на БД, а не на какие-либо отдельные операции. По умолчанию такие ограничения проверяются при конце транзакции, и их нарушение вызывает автоматическую замену оператора COMMIT на оператор ROLLBACK. Однако в некоторых системах поддерживается специальный оператор принудительной проверки ограничений целостности внутри транзакции. Если после выполнения такого оператора обнаруживается, что условия целостности не выполнены, пользователь может сам выполнить оператор ROLLBACK. или постараться устранить причины нецелостного состояния БД внутри транзакции.

Уровни изолированности транзакций

В связи со свойством сохранения целостности БД транзакции являются подходящими единицами изолированности пользователей. Действительно, если с каждым сеансом работы с БД ассоциируется транзакция, то каждый пользователь начинает работу с согласованным состоянием БД, т. е. с таким состоянием, в котором БД могла бы находиться, даже если бы пользователь работал с ней в одиночку.

Следовательно, требуется, чтобы обновления, выполняемые некоторой транзакцией T1, не были доступны для любой другой транзакции T2 до тех и только до тех пор, пока не будет завершено выполнение транзакции T1.

Завершение выполнения транзакции открывает доступ ко всем обновлениям, выполненным данной транзакцией.

Существует несколько **уровней изолированности транзакций**.

По приоритету их можно расположить в следующем порядке: **READ UNCOMMITED (чтение незафиксированных данных), READ COMMITED (чтение зафиксированных данных), REPEATABLE READ (гарантия повторяемости считывания), SERIALIZABLE (способность к упорядочению).**

Далеко не каждая СУБД поддерживает все уровни изолированности транзакций. Последний уровень имеется лишь у наиболее развитых. Если все транзакции выполняются на уровне способности к упорядочению (принятом по умолчанию), то чередующееся выполнение любого множества параллельных транзакций может быть упорядочено.

Однако если любая транзакция выполняется на более низком уровне изоляции, то существует множество различных способов нарушения способности к упорядочению.

Существуют **четыре особых случая нарушения способности к упорядочению**.

1. **Потерянные изменения**. Результаты успешно завершенной операции обновления одной транзакции могут быть перекрыты результатами выполнения другой транзакции. Допустим, транзакция Т1 изменяет объект базы данных А. До завершения транзакции Т1 транзакция Т2 также изменяет объект А, перекрывая таким образом результат предыдущей транзакции. Чтобы избежать такой ситуации, явно противоречащей требованию изолированности пользователей, требуется до завершения транзакции Т1 запретить любой другой транзакции изменять объект А. Отсутствие потерянных изменений является минимальным требованием к многопользовательской СУБД, поскольку в этом случае обеспечивается требование целостности БД при завершении любой транзакции.

2. **Неаккуратное считывание** или чтение «грязных данных». Допустим, что транзакция T1 выполняет вставку новой строки, затем транзакция T2 извлекает эту строку, после чего выполнение транзакции T1 отменяется. В результате транзакция T2 обнаружит, что данной строки больше не существует в том смысле, что она никогда не существовала (поскольку транзакция T1 действительно не была выполнена). Можно рассмотреть еще один сценарий совместного выполнения транзакций T1 и T2. Транзакция T1 изменяет объект базы данных A, а транзакция T2 читает объект A. Поскольку операция изменения еще не завершена, транзакция T2 видит несогласованные «грязные» данные, поскольку в следующий момент времени транзакция T1 может быть отвергнута вследствие нарушения немедленно проверяемого ограничения целостности. Чтобы избежать ситуации чтения «грязных» данных, до завершения транзакции T1, изменившей объект A, никакая другая транзакция не должна иметь возможности читать объект A.

3. **Неповторяемое считывание.** Допустим, транзакция T1 читает объем A. До завершения транзакции T1 транзакция T2 изменяет объект A и успешно завершается. Транзакция T1 повторно читает объект A и видит его измененное состояние. Чтобы избежать неповторяющихся чтений, до завершения транзакции T1 никакая другая транзакция не должна иметь возможность изменять объект A. Отсутствие неповторяющихся чтений часто является максимальным требованием изолированности транзакций, хотя это еще не гарантирует реальной изолированности пользователей.

4. **Наличие фиктивных элементов (фантомов)**. Допустим, что транзакция T1 извлекает множество всех записей, которые удовлетворяют некоторому условию (например, записи всех поставщиков из Москвы). Затем транзакция T2 вставляет новую строку, которая удовлетворяет тому же условию и успешно завершается. Если транзакция T1 вновь повторит ту же операцию извлечения, что и раньше, то ею будет обнаружена ранее отсутствовавшая - «фиктивная» строка. Данная ситуация также противоречит идее изолированности пользователей и может возникнуть даже на третьем уровне изолированности транзакций. Для ее устранения требуется более высокий уровень синхронизации транзакций.

Возможности возникновения этих нарушений при различных уровнях изоляции транзакций приведены в табл.

Уровни изоляции транзакций

Уровень изоляции	Потерянные изменения	Неаккуратное считывание	Неповторяемое считывание	Фиктивные элементы
READ UNCOMMITTED	Нет	Да	Да	Да
READ COMMITTED	Нет	Нет	Да	Да
REPEATABLE READ	Нет	Нет	Нет	Да
SERIALIZABLE	Нет	Нет	Нет	Нет

Для обеспечения **реальной изолированности** транзакций в СУБД применяется метод **сериализации транзакций**.

Сериализиция транзакций – это определение последовательности выполнения транзакций, когда результат совместного выполнения транзакций эквивалентен результату последовательного выполнения этих же транзакций. Система, в которой поддерживается сериализация транзакций, обеспечивает реальную изолированность пользователей. Основная проблема состоит в выборе такого метода сериализации, который не слишком ограничивал бы их параллельность

Тривиальным решением является действительно последовательное выполнение транзакций. Но существуют ситуации, в которых можно выполнять операторы разных транзакций в любом порядке с сохранением сериальности. Примерами могут служить только читающие транзакции, а также транзакции, не конфликтующие по доступу к объектам БД.

Методы сериализации транзакций

Существуют два базовых подхода к сериализации транзакций: подход, основанный на **синхронизационных захватах (блокировках)** объектов БД, и подход, основанный на использовании **временных меток**. Суть обоих подходов состоит в **обнаружении конфликтов транзакций и их устранении**. Практические методы сериализации транзакций основываются на учете этих конфликтов. 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 (реальном или подразумеваемом). **Оптимистические методы** основываются на том, что результаты всех операций модификации БД сохраняются в рабочей памяти транзакций. Реальная модификация БД производится только на стадии фиксации транзакции. Тогда же проверяется, не возникают ли конфликты с другими транзакциями.

Наиболее распространенным в централизованных СУБД (включающих системы, основанные на архитектуре "клиент-сервер") является подход, основанный на соблюдении **двухфазного протокола синхронизационных захватов объектов БД**. В общих чертах протокол состоит в том, что перед выполнением любой операции в транзакции T над объектом базы данных r от имени транзакции T запрашивается синхронизационный захват объекта r в соответствующем режиме (в зависимости от вида операции).

Основными режимами синхронизационных захватов являются:

-совместный режим - S (Shared), означающий разделяемый захват объекта и требуемый для выполнения операции чтения объекта;

-монопольный режим - X (eXclusive), означающий монопольный захват объекта и требуемый для выполнения операций занесения, удаления и модификации.

	X	S
-	да	да
X	нет	нет
S	нет	да

Захваты объектов несколькими транзакциями **по чтению** совместимы, т.е. 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**.
Для обеспечения сериализации транзакций (третьего уровня изолированности) синхронизационные захваты 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 - **накопление захватов**;
вторая фаза (фиксация или откат) - **освобождение захватов**.

Достаточно легко убедиться, что при соблюдении двухфазного протокола синхронизационных захватов действительно обеспечивается сериализация транзакций на третьем уровне изолированности. Основная проблема состоит в том, что следует считать **объектом для синхронизационного захвата**?
В контексте реляционных баз данных возможны следующие альтернативы:
файл - физический (с точки зрения базы данных) объект, область хранения нескольких отношений и, возможно, индексов;
отношение - логический объект, соответствующий множеству кортежей данного отношения;
страница данных - физический объект, хранящий кортежи одного или нескольких отношений, индексную или служебную информацию;
кортеж - элементарный физический объект базы данных.

На самом деле, когда мы говорим про операции над объектами базы данных, то любая операция над кортежем, фактически, является и операцией над страницей, в которой этот кортеж хранится, и над соответствующим отношением, и над файлом, содержащем отношение. Поэтому действительно имеется выбор уровня объекта захвата.

Понятно, что чем **крупнее объект** синхронизационного захвата (неважно, какой природы этот объект - логический или физический), тем **меньше синхронизационных захватов будет поддерживаться в системе, и на это, соответственно, будут тратиться меньшие накладные расходы.**

Но вся беда в том, что 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.

Разработчики многих систем начинали с использования **страничных захватов**, полагая это некоторым компромиссом между стремлениями сократить накладные расходы и сохранить достаточно высокий уровень параллельности транзакций. Но это не очень хороший выбор. Мы не будем останавливаться на деталях, но заметим, что использование страничных захватов в двухфазном протоколе иногда вызывает очень неприятные синхронизационные проблемы, усложняющие организацию СУБД. В большинстве современных систем используются **покортежные синхронизационные захваты**.

Но при этом возникает очередной вопрос. Если единицей захвата является кортеж, то какие синхронизационные захваты потребуются при выполнении таких операций как уничтожение отношения? Было бы довольно нелепо перед выполнением такой операции потребовать захвата всех существующих кортежей отношения. Кроме того, это не предотвратило бы возможности параллельной вставки в другой транзакции нового кортежа в уничтожаемое отношение.

Гранулированные синхронизационные захваты

Подобные рассуждения привели к разработки аппарата гранулированных синхронизационных захватов. При применении этого подхода синхронизационные захваты могут запрашиваться по отношению к объектам разного уровня: **файлам, отношениям и кортежам.** Требуемый уровень объекта определяется тем, какая операция выполняется (например, для выполнения операции уничтожения отношения объектом синхронизационного захвата должно быть все отношение, а для выполнения операции удаления кортежа - этот кортеж). Объект любого уровня может быть захвачен в режиме **S или X**.

Теперь наиболее важное отличие, на котором, собственно, держится соответствие захватов разного уровня. Вводится специальные протокол гранулированных захватов и новые типы захватов: перед захватом объекта в режиме S или X соответствующий объект более верхнего уровня должен быть захвачен в режиме **IS, IX или SIX**.

IS (Intented for Shared lock) по отношению к некоторому составному объекту O означает намерение захватить некоторый входящий в O объект в совместном режиме. Например, при намерении читать кортежи из отношения R это отношение должно быть захвачено в режиме IS (а до этого в таком же режиме должен быть захвачен файл).
IX (Intented for eXclusive lock) по отношению к некоторому составному объекту O означает намерение захватить некоторый входящий в O объект в монопольном режиме. Например, при намерении удалять кортежи из отношения R это отношение должно быть захвачено в режиме IX (а до этого в таком же режиме должен быть захвачен файл).
SIX (Shared, Intented for eXclusive lock) по отношению к некоторому составному объекту O означает совместный захват всего этого объекта с намерением впоследствии захватывать какие-либо входящие в него объекты в монопольном режиме. Например, если выполняется длинная операция просмотра отношения с возможностью удаления некоторых просматриваемых кортежей, то экономичнее всего захватить это отношение в режиме SIX (а до этого захватить файл в режиме IS).

Довольно трудно описать словами все возможные ситуации. Мы ограничимся приведением полной таблицы совместимости захватов, анализируя которую можно выявить все случаи:

	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

Предикатные синхронизационные захваты

Несмотря на привлекательность метода **гранулированных синхронизационных захватов**, следует отметить что он не решает **проблему фиктивной информации** (если, конечно, не ограничиться использованием захватов отношений в режимах S и X). Давно известно, что для решения этой проблемы необходимо перейти от захватов индивидуальных объектов базы данных, к захвату **условий (предикатов),** которым удовлетворяют эти объекты.

Проблема **фантомов** не возникает при использовании для синхронизации **уровня отношений** именно потому, что отношение как логический объект представляет собой неявное условие для входящих в него кортежей.

Захват отношения - это простой и частный случай предикатного захвата.

Поскольку 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), 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.

К счастью, эта проблема сравнительно легко решается для случая **простых условий**. Будем называть простым условием конъюнкцию простых предикатов, имеющих вид

имя-атрибута { = > < } значение

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

Тупики, распознавание и разрушение

Одним из наиболее чувствительных недостатков метода сериализации транзакций на основе синхронизационных захватов является возможность возникновение **тупиков (deadlocks)** между транзакциями. Тупики возможны при применении любого из рассмотренных нами вариантов.

Вот простой пример возникновения тупика между транзакциями T1 и T2:

транзакции T1 и T2 установили монопольные захваты объектов r1 и r2 соответственно;

после этого T1 требуется совместный захват r2, а T2 - совместный захват r1;

ни одна из транзакций не может продолжаться, следовательно, монопольные захваты не будут сняты, а совместные - не будут удовлетворены.

Поскольку тупики возможны, и никакого естественного выхода из тупиковой ситуации не существует, то эти ситуации необходимо обнаруживать и искусственно устранять.

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, и вершины, соответствующие объектам захвата. В этом графе существует дуга, ведущая из вершины-транзакции к вершине-объекту, если для этой транзакции существует удовлетворенный захват объекта. В графе существует дуга из вершины-объекта к вершине-транзакции, если транзакция ожидает удовлетворения захвата объекта.

Легко показать, что в системе существует ситуация **тупика**, если в графе ожидания транзакций имеется хотя бы **один цикл**. Традиционной техникой (для которой существует множество разновидностей) нахождения циклов в ориентированном графе является **редукция графа**.

Не вдаваясь в детали, редукция состоит в том, что прежде всего из графа ожидания **удаляются все дуги, исходящие из вершин-транзакций, в которые не входят дуги из вершин-объектов.** (Это как бы соответствует той ситуации, что транзакции, не ожидающие удовлетворения захватов, успешно завершились и освободили захваты). Для тех вершин-объектов, для которых не осталось входящих дуг, но существуют исходящие, **ориентация исходящих дуг изменяется на противоположную** (это моделирует удовлетворение захватов). После этого снова срабатывает первый шаг и так до тех пор, пока на первом шаге не прекратится удаление дуг. Если в графе остались дуги, то они обязательно образуют цикл.

Предположим, что нам удалось найти цикл в графе ожидания транзакций. Что делать теперь? 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.

Грубо говоря, 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** этой транзакции, который может носить полный или частичный характер. При этом, естественно, освобождаются захваты и может быть продолжено выполнение других транзакций. Естественно, такое насильственное устранение тупиковых ситуаций является нарушением принципа изолированности пользователей, которого невозможно избежать.

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

Чтобы минимизировать число конфликтов между транзакциями, в некоторых СУБД (например, в Oracle) используется следующее развитие подхода. **Монопольный захват объекта блокирует только изменяющие транзакции.** После выполнении операции модификации предыдущая версия объекта остается доступной для чтения в других транзакциях. Кратковременная блокировка чтения требуется только на период фиксации изменяющей транзакции, когда обновленные объекты становятся текущими.

Альтернативный метод сериализации транзакций, хорошо работающий в условиях редких конфликтов транзакций, основан на использовании **временных меток**. Основная идея метода (у которого существует множество разновидностей) состоит в следующем: **если транзакция Т1 началась раньше транзакции Т2, то система обеспечивает такой режим выполнения, как если бы Т1 была целиком выполнена до начала Т2.**

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 r транзакция T1 проверяет, не закончилась ли транзакция T, пометившая этот объект. Если T закончилась, T1 помечает объект r 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 при t(T) > t(T1) (т. е. транзакция T является более «молодой», чем T1), производится откат T, и T1 продолжает работу. Если же t(T) < t (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. Но все эти недостатки окупаются тем, что **не нужно распознавать и устранять тупики**, реализация чего в распределенных системах стоит очень дорого.