Лекция 4

Методы управления транзакциями

План лекции

- Общее понятие транзакции и основные характеристики транзакций
- Методы сериализации транзакции
  - синхронизационные блокировки
  - deadlock-и
  - методы временных меток
  - ~~версионные методы управления~~

Обработка транзакций

- Часто база данных должна обрабатывать события, которые приводят более чем к одному изменению в БД.

Обработка транзакций

Рассмотрим событие – прием нового заказа от клиента:

- Добавление нового заказа в таблицу ORDERS
- Обновление фактического объема продаж для служащего, принявшего заказ
- Обновление фактического объема продаж для офиса, в котором работает данный служащий
- Обновление количества товара, имеющегося в наличии

Обработка транзакций

Допустим, что произошел системный сбой или другая ошибка.

Тогда одна часть изменений была внесена, а другая нет. Это приводит к нарушению целостности хранимых данных, и при последующих вычислениях результаты окажутся неверными.

Изменения базы данных, которые были вызваны одним событием, необходимо вносить по принципу “Всё или ничего”.

SQL обеспечивает такое поведение посредством возможностей обработки транзакций.

Общее понятие транзакции

Транзакция – это последовательность операций над БД, рассматриваемых СУБД как единое целое.

Транзакция в SQL – это несколько последовательных инструкций SQL, которые вместе образуют логическую единицу работы (unit of work).

Общее понятие транзакции

- Поддержка механизма транзакций – показатель уровня развитости СУБД.
- Корректное поддержание транзакций является основой обеспечения целостности баз данных
- Механизм транзакций составляет базис изолированности пользователей в многопользовательских системах

Общее понятие транзакции

В современных СУБД поддерживается понятие транзакции, характеризуемое аббревиатурой ACID:

- Atomicity (атомарность)
- Consistency (непротиворечивость)
- Isolation (изолированность)
- Durability (устойчивость)

Атомарность

- Означает, что результаты всех операций, успешно выполненных в пределах транзакции, должны быть отражены в состоянии базы данных, либо в состоянии базы данных не должно быть отражено действие ни одной операции.
- Свойство атомарности, которое часто называют свойством “все или ничего”, позволяет относиться к транзакции, как к динамически образуемой составной операции над базой данных. В общем случае состав и порядок выполнения операций, выполняемых внутри транзакции, становится известным только на стадии выполнения.

«Все или ничего»:

- при успешном завершении транзакции оператором COMMIT
  - результаты гарантированно фиксируются во внешней памяти
  - смысл термина commit состоит в запросе «фиксации» результатов транзакции
- при завершении транзакции оператором ROLLBACK
  - результаты гарантированно отсутствуют во внешней памяти
  - смысл термина rollback состоит в запросе ликвидации результатов транзакции

Непротиворечивость (целостность)

- Означает, что транзакция может быть успешно завершена с фиксацией результатов своих операций только в том случае, когда действия операций не нарушают целостность базы данных, т.е. удовлетворяют набору ограничений целостности, определенных для этой базы данных.
- Это свойство расширяется тем, что во время выполнения транзакции разрешается устанавливать точки согласованности и явным образом проверять ограничения целостности.

Изолированность

- Требуется, чтобы две одновременно выполняемые транзакции никаким образом не действовали одна на другую.
- Результаты выполнения операций транзакции T_1 не должны быть видны никакой другой транзакции T_2 до тех пор, пока транзакция T_1 не завершится успешным образом.

Устойчивость (долговечность)

- После успешного завершения транзакции, все изменения, которые были внесены в состояние базы данных операциями этой транзакции, должны гарантированно сохраняться, даже в случае сбоев аппаратуры или программного обеспечения.

Транзакции и целостность баз данных

- Часто база данных обладает такими ограничениями целостности, которые просто невозможно не нарушить, выполняя только один оператор изменения базы данных.
- Для поддержки подобных ограничений целостности допускается их нарушение внутри транзакции с тем условием, чтобы к моменту завершения транзакции условия целостности были соблюдены.
- В системах, которые поддерживают ACID транзакции, каждая транзакция начинается при логически целостном состоянии базы данных и должна оставить это состояние целостным после своего завершения.

Транзакции и целостность баз данных

Начальная проверка целостности БД проверяется при первоначальной загрузке БД, что гарантирует её целостность на момент выполнения некоторой транзакции *T*. Теперь эта транзакция фиксирует изменения операцией COMMIT:

- Если вдруг транзакция нарушает логическую целостность базы данных, то вместо фиксации происходит откат изменений (ROLLBACK)
- Если транзакция не нарушает целостность базы данных, то происходит фиксация, и база данных всё равно в целостном состоянии

Ограничения целостности

Различаются два вида ограничений целостности: немедленно проверяемые и откладываемые.

- К немедленно проверяемым ограничениям целостности относятся такие ограничения, проверку которых бессмысленно или даже невозможно откладывать. При их нарушениях не производится откат транзакции, а лишь отвергается соответствующий оператор.
- Откладываемые ограничения целостности – это ограничения на базу данных, 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.

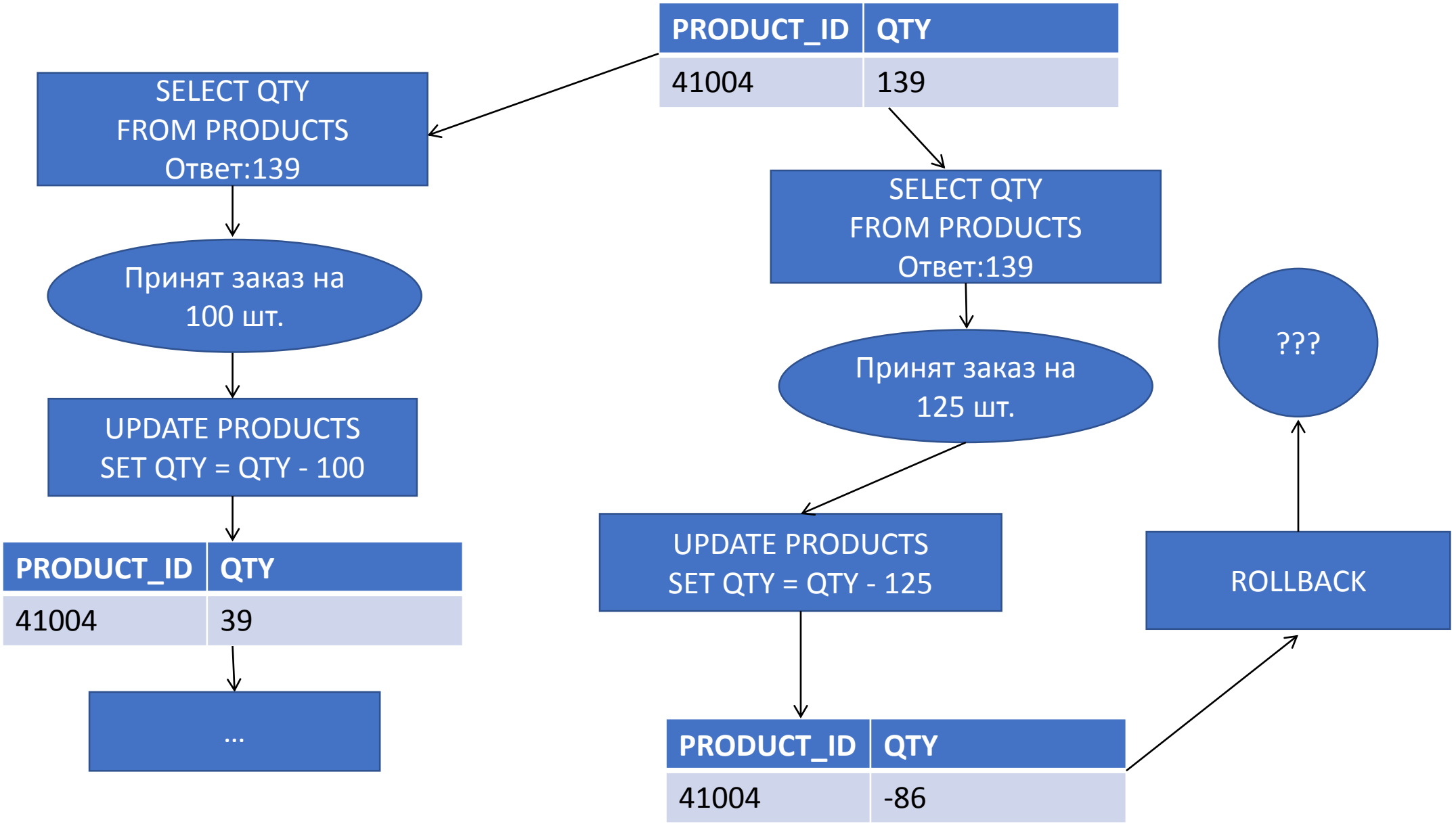
Изолированность транзакций

- В многопользовательских системах с одной базой данных одновременно может работать несколько пользователей или прикладных программ. Предельной задачей системы является обеспечение изолированности пользователей (приложений), т.е. создание достоверной и надежной иллюзии того, что каждый из пользователей работает с базой данных в одиночку.
- В связи со свойством сохранения целостности базы данных транзакции являются подходящими единицами изолированности пользователей.

Потерянные изменения (*DIRTY WRITE)*

Транзакция T_1 модифицирует строку. Другая транзакция T_2 также модифицирует эту строку до COMMIT или ROLLBACK от T_1. Если T_1 или T_2 произведет ROLLBACK не ясно, какие данные должны быть корректны.

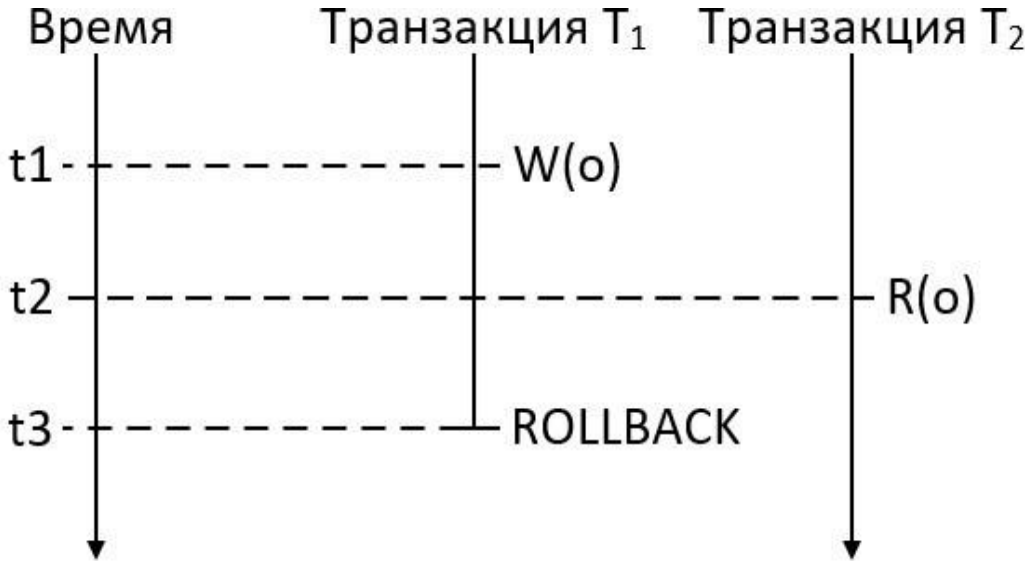
LOST UPDATE


PRODUCT_ID
QTY
41004
139
SELECT QTY
FROM PRODUCTS
Ответ:139
Принят заказ на
100 шт.
UPDATE PRODUCTS
SET QTY = QTY - 100
PRODUCT_ID
QTY
41004
39
...
SELECT QTY
FROM PRODUCTS
Ответ:139
Принят заказ на
125 шт.
UPDATE PRODUCTS
SET QTY = QTY - 125
PRODUCT_ID
QTY
41004
-86
???
ROLLBACK

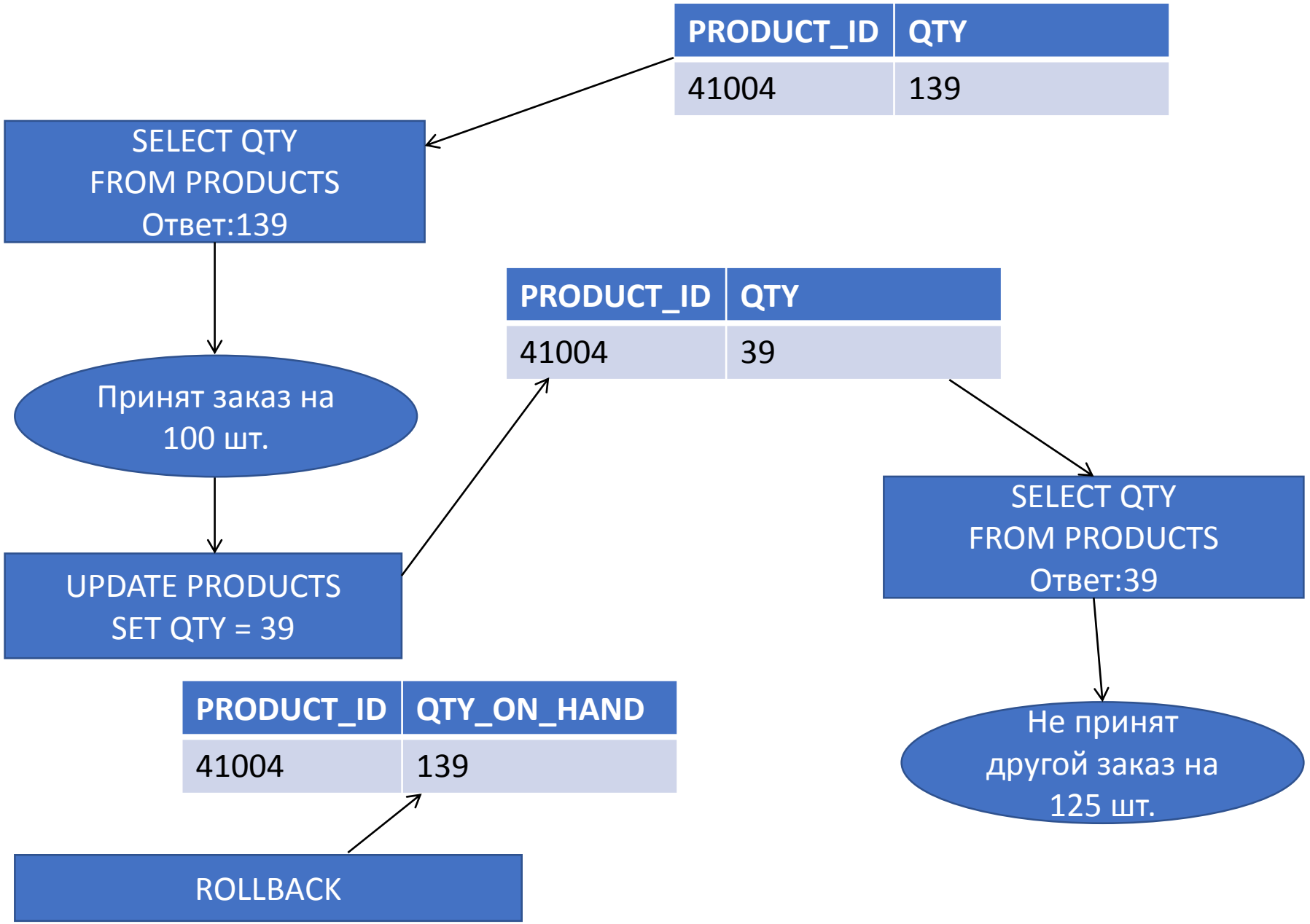
Чтение «грязных» данных (*DIRTY READ*)

Пусть в момент времени $t1$ транзакция T_1 изменяет объект базы данных *o.* В момент времени $t2 > t1$ транзакция T_2 читает объект *o*. Транзакция T_1 еще не завершена, поэтому транзакция T_2 видит несогласованные «грязные» данные, которых в базе данных не было.

Пусть в момент времени t3 > t2 транзакция T1 завершается откатом, тогда эти грязные данные ещё и не зафиксируются.

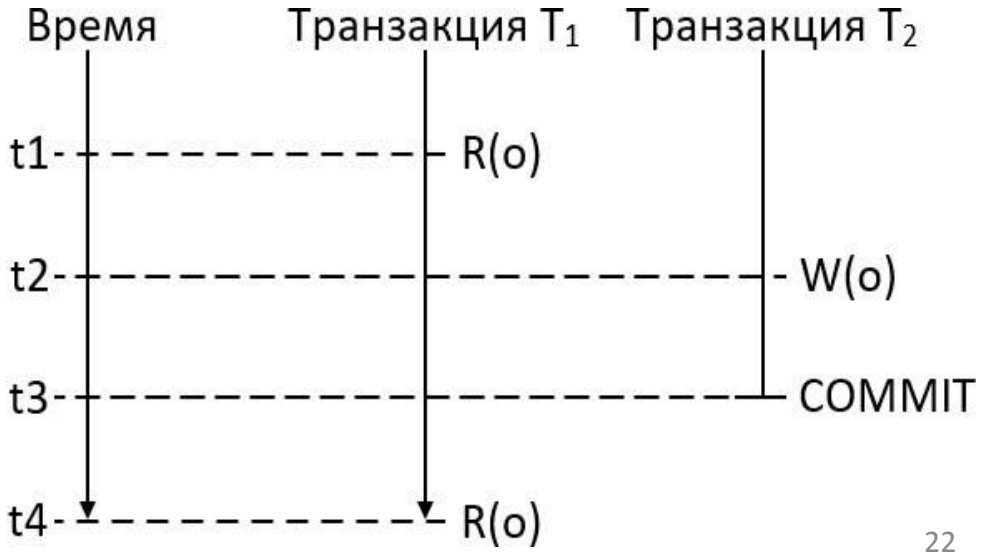


DIRTY READ

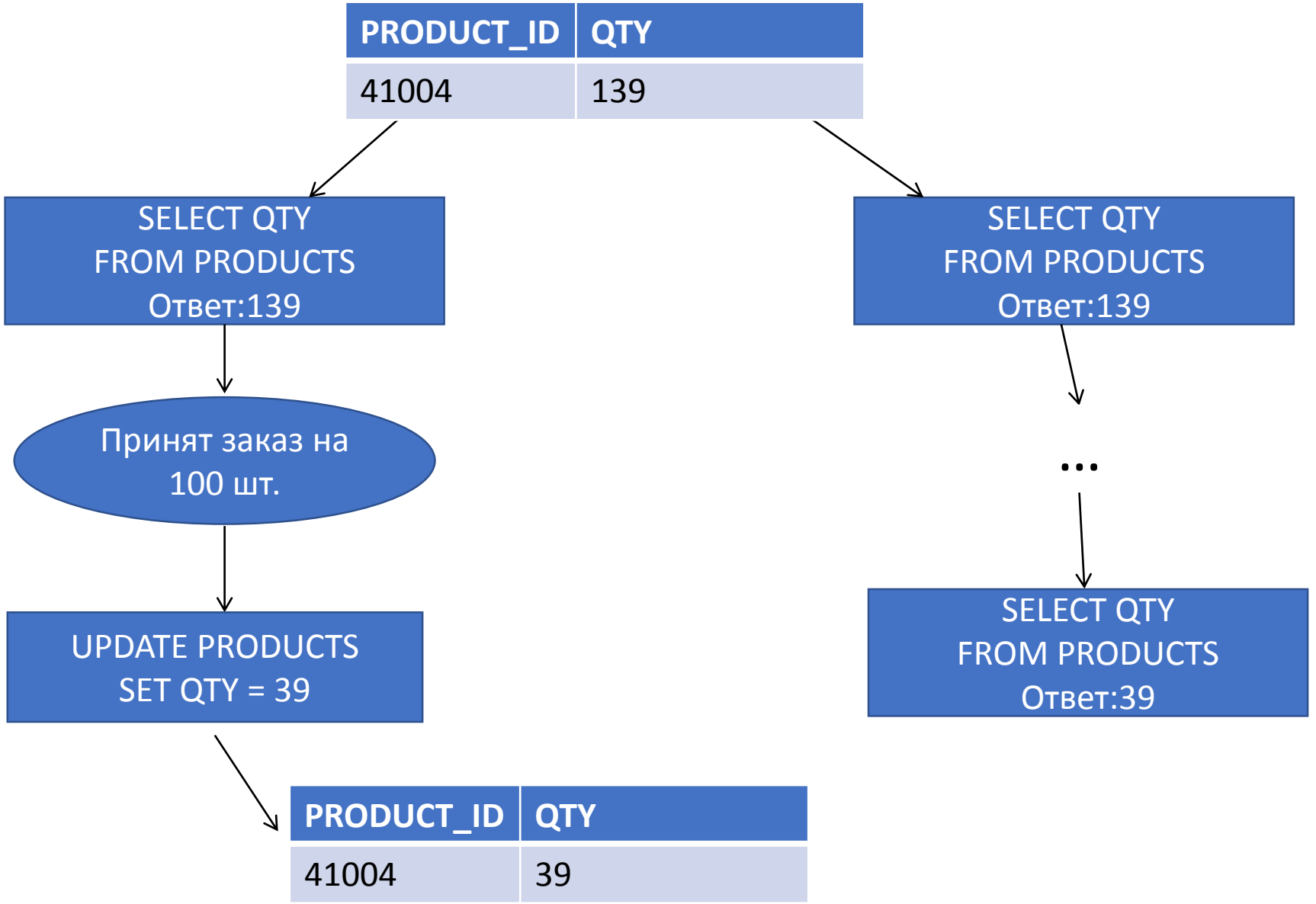

PRODUCT_ID
QTY
41004
139
SELECT QTY
FROM PRODUCTS
Ответ:139
Принят заказ на
100 шт.
PRODUCT_ID
QTY
41004
39
UPDATE PRODUCTS
SET QTY = 39
SELECT QTY
FROM PRODUCTS
Ответ:39
PRODUCT_ID
QTY_ON_HAND
41004
139
Не принят
другой заказ на
125 шт.
ROLLBACK

Неповторяющиеся чтения (*NONREPEATABLE READ*)

Пусть в момент времени $t1$ транзакция T_1 читает объект базы данных o. До завершения транзакции T_1 в момент времени $t2 > t1$ транзакция T_2 изменяет объект o и успешно завершается оператором COMMIT. В момент времени $t3 > t2$ транзакция T_1 повторно читает объект o и видит его изменённое состояние.



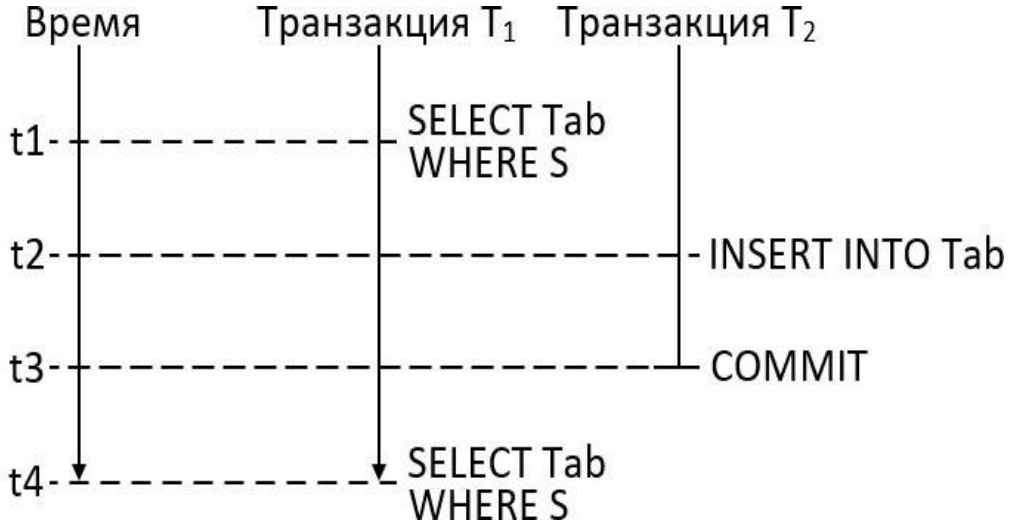
NONREPEATABLE READ



Фантомы (*PHANTOM*)

Пусть в момент времени $t1$ транзакция T_1 выполняет оператор выборки строк таблицы Tab с условием выборки S. До завершения транзакции T_1 в момент времени $t2$ транзакция T_2 вставляет в таблицу Tab новую строку r, удовлетворяющую условию S, и успешно завершается.

Затем транзакция T_1 повторно выполняет тот же оператор выборки, и в результате появляется строка, которая отсутствовала при первом выполнении оператора.



PHANTOM

ORDER_NUM	AMOUNT
112962	31500.00
113012	3745.00

SELECT *
FROM ORDERS

112962,31500

113012,3745

INSERT INTO VALUES
(118102, .., 5.000)

COMMIT

ORDER_NUM	AMOUNT
112962	31500.00
118102	5000.00
113012	3745.00

SELECT *
FROM ORDERS

112962,31500

118102,5000

113012,3745

Сериализация транзакций

Пусть в системе одновременно выполняется некоторое множество транзакций $S = \{T_1, T_2, ..., T_n\}$. План выполнения набора транзакций S, в котором, вообще говоря, чередуются или реально параллельно выполняются операции разных транзакций $T_1, T_2, ..., T_n$, называется ***сериальным***, если результат совместного выполнения транзакций эквивалентен результату некоторого последовательного выполнения этих же транзакций $(T_{i1}, T_{i2}, ..., T_{in})$.

Сериализация транзакций – это механизм одновременного выполнения транзакций $T_1, T_2, ..., T_n$ по некоторому сериальному плану.

Конфликты

Между транзакциями T_1 и T_2 могут существовать следующие виды конфликтов:

- W/W (Write/Write) – транзакция T_2 пытается изменять объект, изменённый не закончившейся транзакцией T_1. Может привести к ситуации потерянных изменений.
- R/W (Read/Write) – транзакция T_2 пытается изменять объект, прочитанный не закончившейся транзакцией T_1. Может привести к возникновению ситуации неповторяющихся чтений.
- W/R (Write/Read) – транзакция T_2 пытается читать объект, изменённый не закончившейся транзакцией T_1. Может привести к возникновению ситуации «грязного» чтения.

Методы сериализации транзакций

Существуют два базовых подхода к сериализации транзакций:

- основанный на синхронизационных блокировках объектов базы данных
- основанный на использовании временных меток

Для каждого из подходов имеются две разновидности: пессимистическая и оптимистическая.

- При применении пессимистических методов, ориентированных на ситуации, когда конфликты возникают часто, конфликты распознаются и разрешаются немедленно при их возникновении.
- Оптимистические методы хорошо работают, когда конфликты редки. Любая транзакция, изменяющая базу данных выполняется без синхронизации вообще, но все её результаты в базу данных не пишутся, а сохраняются в её локальной памяти (до операции COMMIT).

  Когда отрабатывает операция COMMIT система проигрывает транзакцию заново и смотрит не было ли конфликтов при выполнении рабочей фазы транзакции. Если конфликтов не было, то все её изменения реально записываются в БД. Если конфликты обнаруживаются, то транзакция откатывается.

Синхронизационные блокировки

- Это наиболее распространенный в централизованных СУБД подход, основанный на соблюдении двухфазного протокола синхронизационных захватов объектов баз данных (TwoPhase Locking Protocol, 2PL).
- В общих чертах подход состоит в том, что перед выполнением любой операции в транзакции *T* над объектом базы данных *o* от имени транзакции *T* запрашивается синхронизационная блокировка объекта *o* в соответствующем режиме (в зависимости от вида операции).

Синхронизационные блокировки

Основными режимами в базовом варианте 2PL являются следующие:

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

Синхронизационные блокировки

Блокировка объекта одной транзакцией по чтению не совместима с блокировкой другой транзакцией того же объекта по записи:

- никакая транзакция не может изменять объект, читаемый некоторой транзакцией (кроме самой этой транзакции)
- никакой транзакции нельзя читать объект, изменяемый некоторой транзакцией (кроме самой этой транзакции)

Правила совместимости режимов блокировок S и X:

	X	**S**
-	да	да
X	нет	нет
S	нет	да

Двухфазный протокол (2PL)

Для обеспечения сериализации транзакций синхронизационные блокировки объектов, произведённые по инициативе транзакции, можно снимать только при её завершении. Это требование порождает двухфазный протокол синхронизационных захватов – 2PL. В соответствии с этим протоколом выполнение транзакции разбивается на две фазы:

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

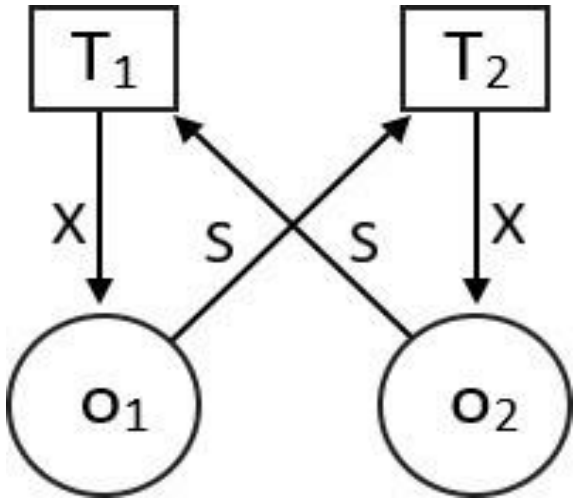
Уровни блокировки

- БД
- файл – физический (с точки зрения базы данных) объект, область хранения нескольких таблиц и, возможно, индексов
- таблица – логический объект, соответствующий множеству кортежей данной таблицы
- страница данных – физический объект, хранящий кортежи одной или нескольких таблиц, индексную или служебную информацию
- кортеж (строка) – элементарный физический объект базы данных

Синхронизационные тупики, их распознавание и разрушение

Методам блокировочных протоколов свойствена возможность возникновения тупиков (deadlocks) между транзакциями.

Справа показан простой сценарий возникновения синхронизационного тупика между транзакциями T_1 и T_2. Такой ориентированный граф называется графом ожидания транзакций. Циклы в таком графе обозначает синхронизационный тупик. Такие ситуации необходимо обнаруживать и искусственно устранять.

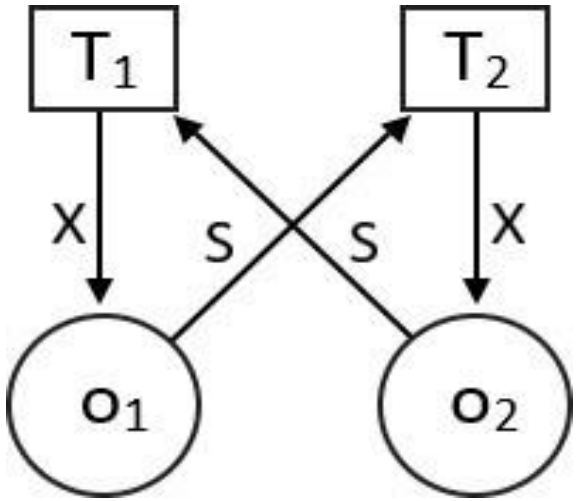


Обнаружение тупиковых ситуаций

Основой обнаружения тупиковых ситуаций является построение или постоянное поддержание графа ожидания транзакций. Граф ожидания транзакций – это двудольный ориентированный граф, вершины которого соответствуют либо транзакциям, либо объектам блокировок.

В этом графе дуги соединяют только вершины-транзакции с вершинами-объектами:

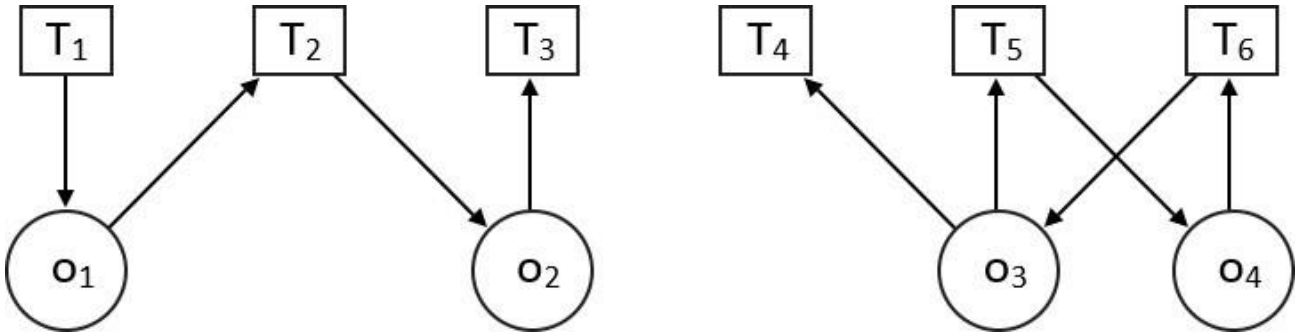
- Дуга из вершины-транзакции к вершине-объекту существует в том и только в том случае, если для этой транзакции имеется удовлетворенная блокировка данного объекта.
- Дуга из вершины-объекта к вершине-транзакции существует тогда и только тогда, когда эта транзакция ожидает удовлетворения запроса блокировки данного объекта.



Обнаружение тупиковых ситуаций

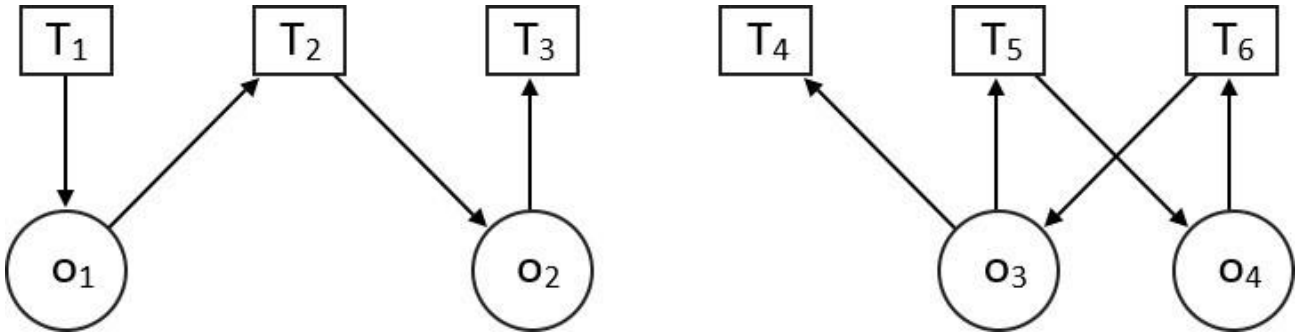
В системе существует тупиковая ситуация в том и только в том случае, когда в графе ожидания транзакций имеется хотя бы один цикл. Для распознавания тупиковых ситуаций периодически производится построение графа ожидания транзакций, и в этом графе ищутся циклы.

Алгоритм редукции графа (на примере):



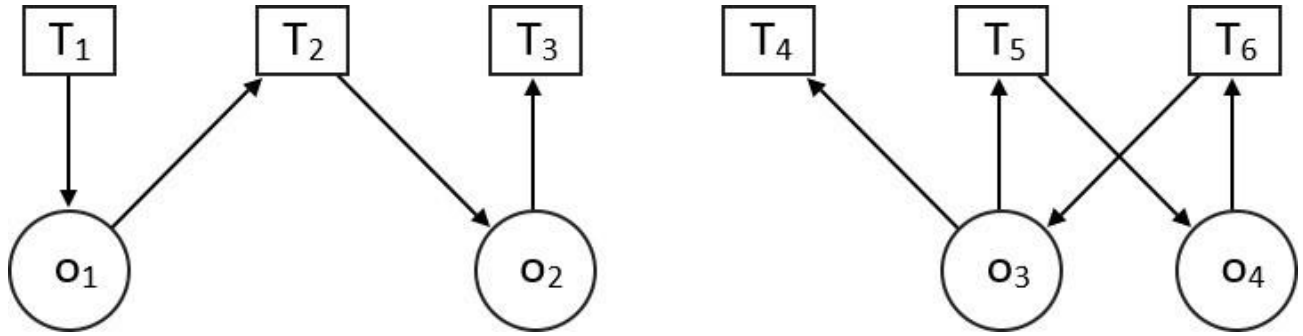
Алгоритм редукции графа ожидания:

- Удаляются все дуги от транзакций к объектам такие, что данная транзакция не ожидает блокировки каких-либо объектов. Если транзакции не ожидают удовлетворения запроса блокировок, то они не могут создавать тупика.
- Удаляются все дуги от объектов к транзакциям такие, что у транзакции нет заблокированных объектов. Если транзакции ожидают удовлетворения блокировок, но не удерживают заблокированных объектов, то они не могут создавать тупика.



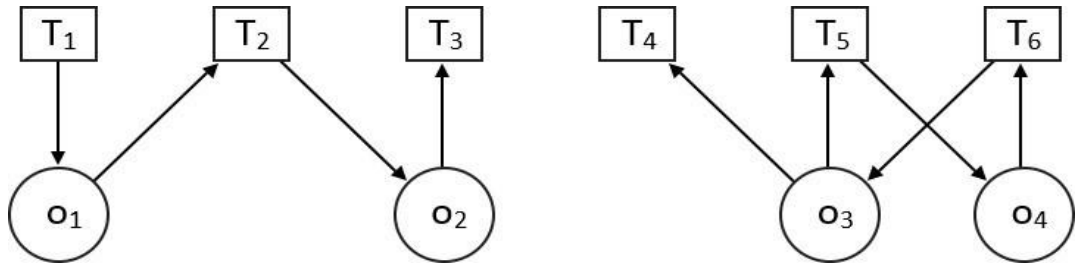
Алгоритм редукции графа ожидания:

- Для объектов без входящих дуг, но с исходящими дугами, ориентация одной произвольной исходящей дуги изменяется на противоположную - это моделирует удовлетворение запроса блокировки.
- Снова повторяются описанные действия с шага 1 до тех пор, пока не прекратится удаление дуг.
- Если в результате работы этого алгоритма в графе останутся дуги, то они обязательно образуют цикл.



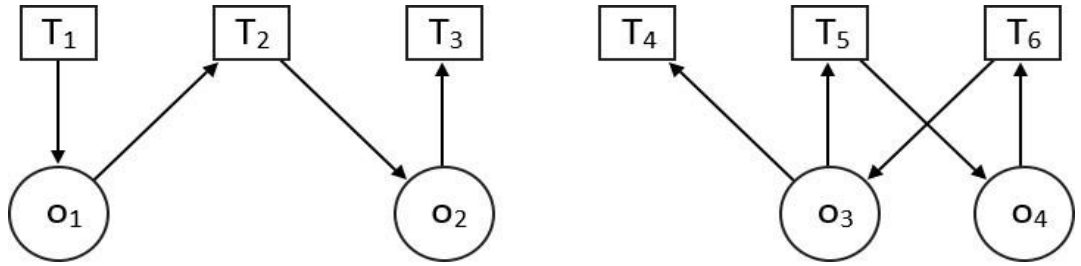
Алгоритм редукции графа ожидания:

1)

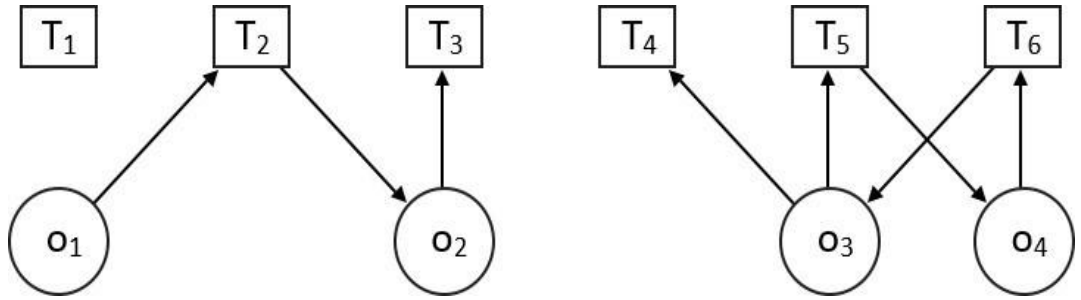


Алгоритм редукции графа ожидания:

1)

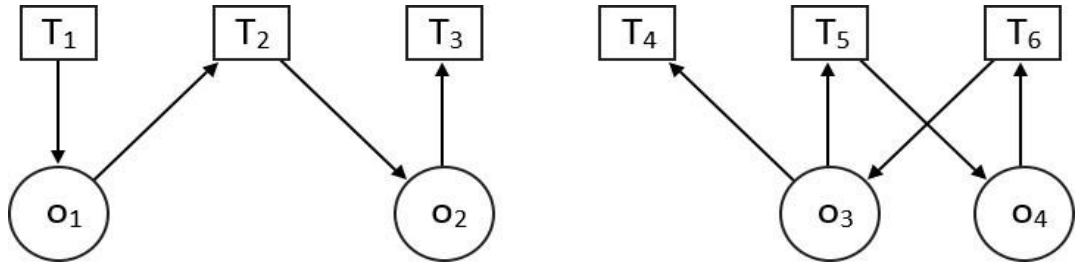


2)

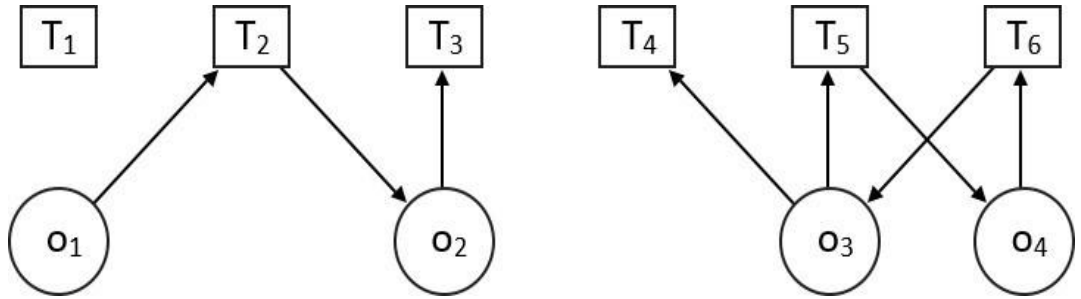


Алгоритм редукции графа ожидания:

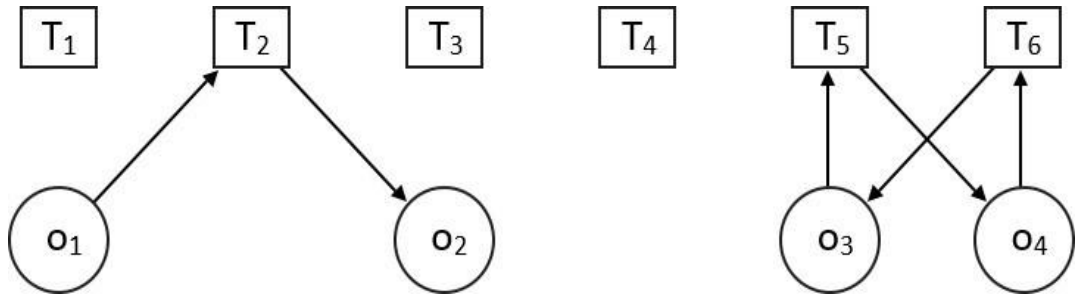
1)



2)

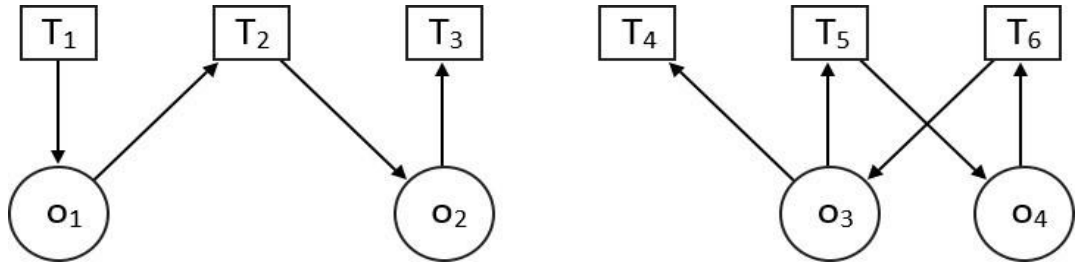


3)

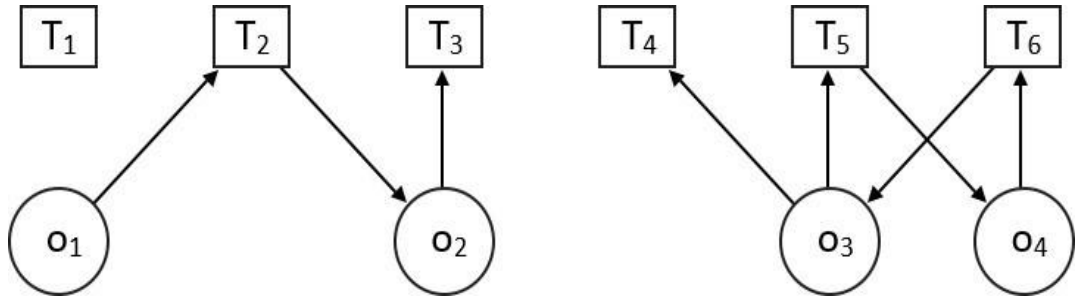


Алгоритм редукции графа ожидания:

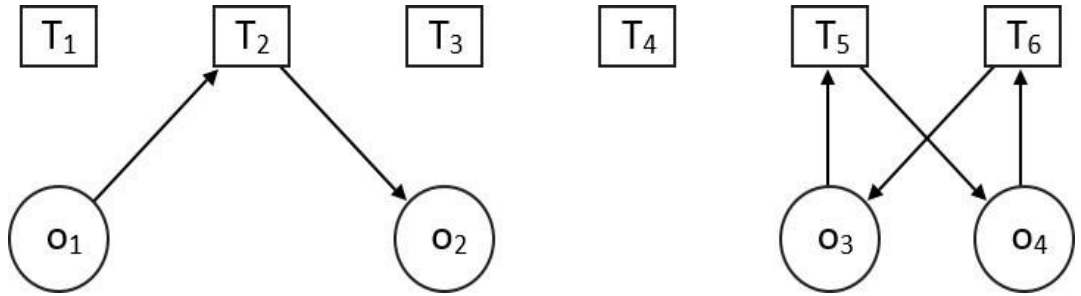
1)



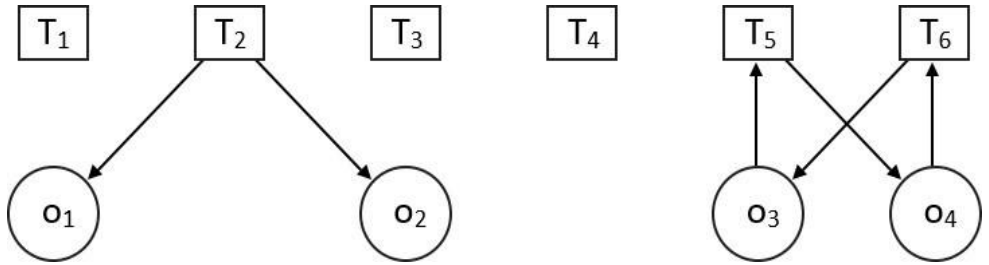
2)



3)

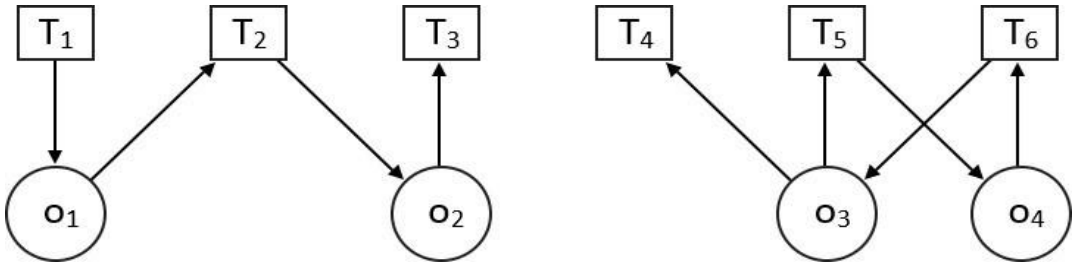


4)

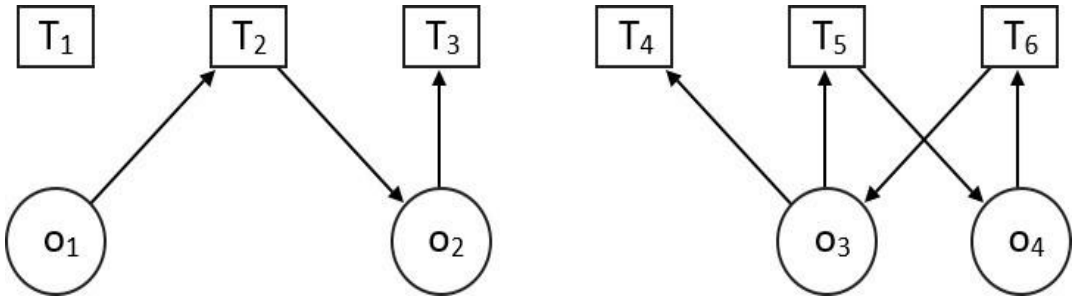


Алгоритм редукции графа ожидания:

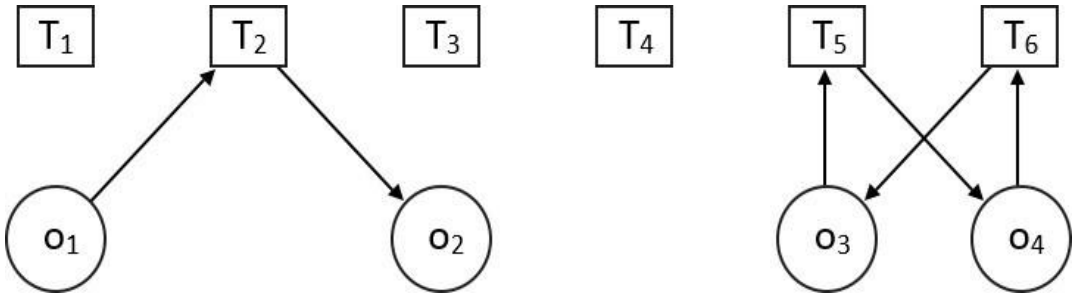
1)



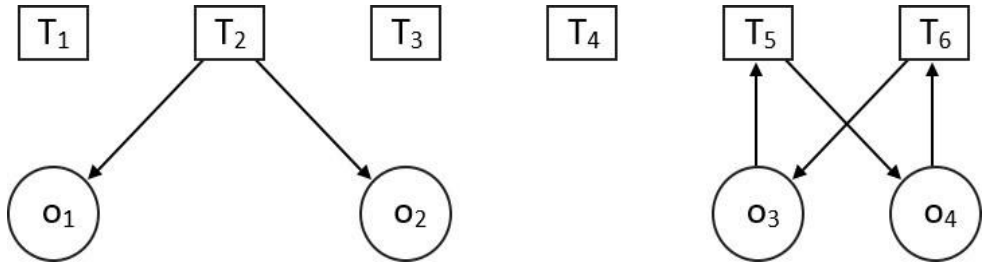
2)



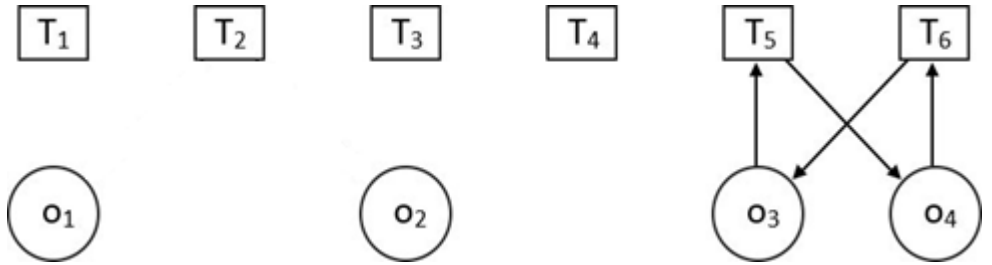
3)



4)

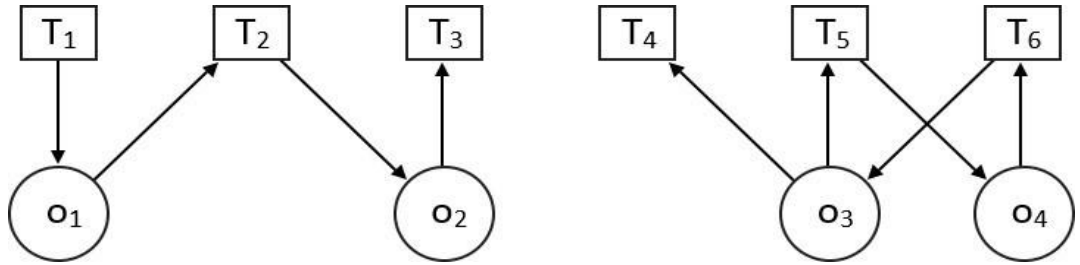


5)

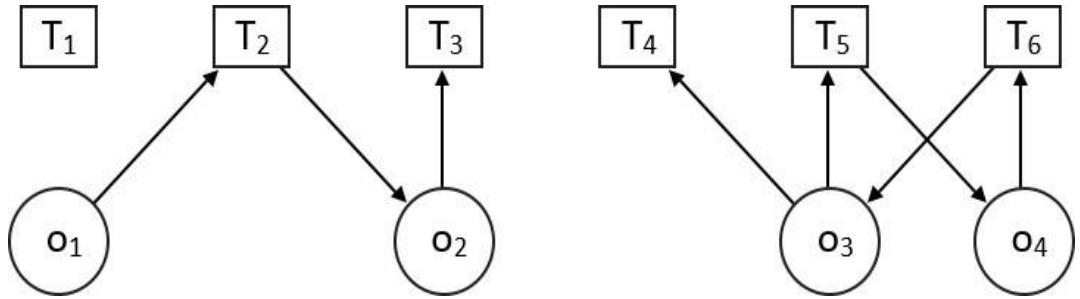


Алгоритм редукции графа ожидания:

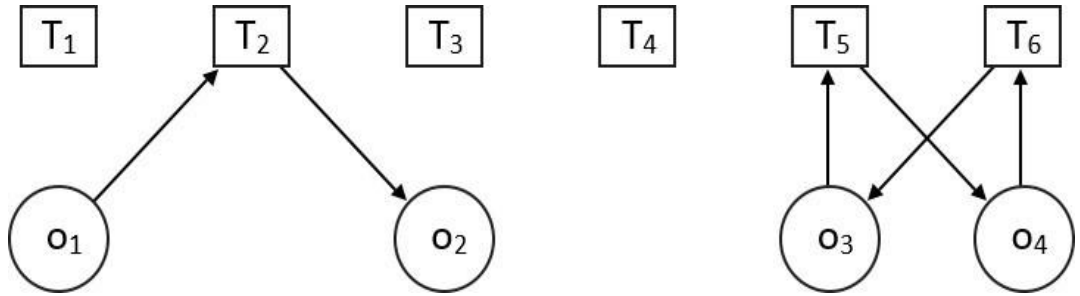
1)



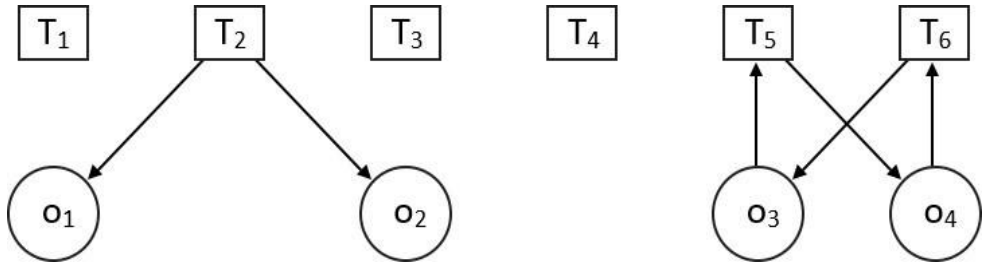
2)



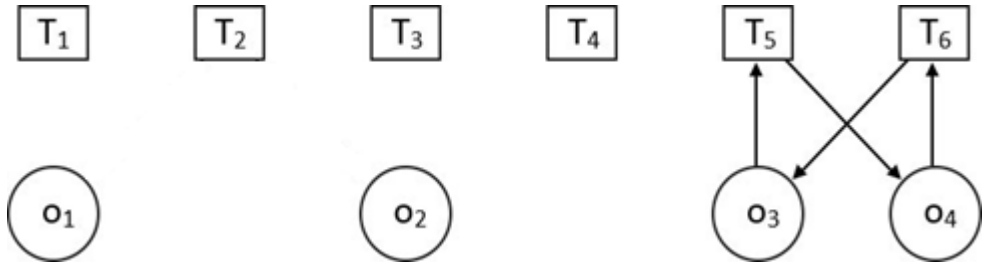
3)



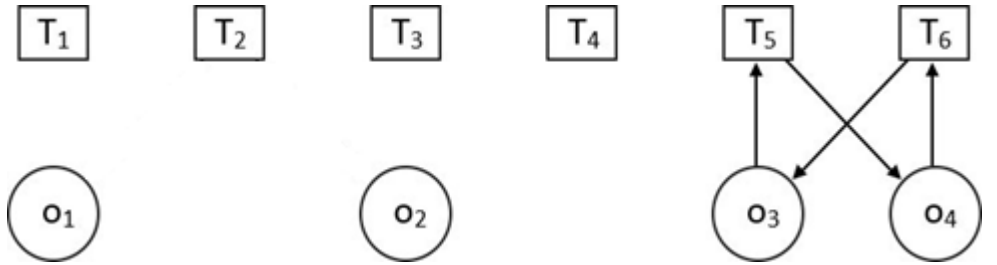
4)



5)



6)



Разрушение тупиков

- Нужно каким-то образом обеспечить возможность продолжения работы хотя бы для части транзакций, попавших в тупик.
- Разрушение тупика начинается с выбора в цикле транзакции-жертвы, т.е. транзакции, которой решено пожертвовать, чтобы обеспечить возможность продолжения работы других транзакций. Для этого могут использоваться различные, зачастую противоречивые критерии.
  - транзакция, которая удерживает наибольшее число блокировок объектов
  - транзакция, которая существует в системе в течение наименьшего времени
  - можно выбрать транзакцию-жертву случайным образом

Разрушение тупиков

- Обычно при выборе транзакции-жертвы используется многофакторная оценка её стоимости, в которую с разными весами входят время выполнения, число накопленных блокировок, приоритет и т.д. В качестве «жертвы» выбирается транзакция, для которой эта оценка выдает наиболее подходящий результат.
- После выбора транзакции-жертвы выполняется откат этой транзакции, который может носить полный или частичный (до некоторой точки сохранения) характер. При этом, естественно, освобождаются блокировки, и может быть продолжено выполнение других транзакций.

Метод временных меток

Основная идея метода временных меток (Timestamp Ordering, TO), состоит в следующем:

- Если транзакция T_1 началась раньше транзакции T_2, то система обеспечивает такой сериальный план, как если бы транзакция T_1 была целиком выполнена до начала T_2. Для этого каждой транзакции T предписывается временная метка $t(T)$, соответствующая времени начала выполнения транзакции T.
- При выполнении операции над объектом o транзакция T помечает его своими идентификатором, временной меткой и типом операции (чтение или изменение).

Метод временных меток

Перед выполнением операции над объектом *o* транзакция T_2 выполняет следующие действия:

- Проверяет, помечен ли объект *o* какой-либо транзакцией T_1. Если не помечен, то помечает этот объект своей временной меткой и типом операции и выполняет операцию.
- Иначе (если *o* помечен T_1) транзакция T_2 проверяет, не завершилась ли транзакция T_1, пометившая этот объект. Если транзакция T_1 закончилась, то T_2 помечает объект *o* и выполняет свою операцию.
- Иначе (если T_1 не завершилась) T_2 проверяет конфликтность операций. Если операции неконфликтны, то при объекте *o* запоминается идентификатор транзакции T_2, остается или проставляется временная метка с меньшим значением, и транзакция T_2 выполняет свою операцию.

Метод временных меток

- Иначе (если операции транзакций T_2 и T_1 конфликтуют), то если $t(T_2) > t(T_1)$ (т.е. транзакция T_1 «моложе» T_2), то производится откат T_1 и всех других транзакций, идентификаторы которых сохранены при объекте o, и T_2 выполняет свою операцию.
- Если же $t(T_2) \leq t(T_1)$ (T_1 «старше» T_2), то производится откат T_2, и T_2 получает новую временную метку и начинается заново.

К недостаткам метода ТО относятся потенциально более частые откаты транзакций, чем в случае использования синхронизационных захватов. Это связано с тем, что конфликтность транзакций определяется более грубо. Но в распределенных системах эти недостатки окупаются тем, что не нужно распознавать тупики, а построение графа ожидания в распределенных системах стоит очень дорого.

Версионные методы

Версионные методы

- Основная идея версионных 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, свойственных механизмам синхронизационных блокировок, а также без некоторых откатов, возможных при применении метода временных меток.

Версионный вариант метода временных меток

- Multiversion Timestamp Ordering, 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 её записала.
- Алгоритм не только следит за порядком выполнения операций транзакций, но также отвечает за преобразование 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*.

Версионный вариант метода временных меток

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 $t(T_k) \leq t(T_i)$.
2) Операция $W_i(o)$ обрабатывается следующим образом:
   - Если уже обработана операция $R_j(o_k)$, такая что $t(T_k) \leq t(T_i) < t(T_j)$, то операция $W_i(o)$ отменяется, а транзакция T_i откатывается.
   - В противном случае $W_i(o)$ преобразуется в $W_i(o_i)$.
3) Завершение (COMMIT) любой транзакции T откладывается до завершения всех транзакций, записавших версии объектов, которые прочитала T.

Версионный вариант метода временных меток

- При откате любой транзакции уничтожаются все созданные ею версии объектов базы данных и откатываются все транзакции, прочитавшие хотя бы одну из этих версий («каскадные» откаты).
- Основные преимущества алгоритма MVTO - это отсутствие задержек и откатов при выполнении операций чтения, а основной недостаток – 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.

Версионный вариант протокола 2PL

- Two-Version Two-Phase Locking Protocol, 2V2PL
- Поддерживается до двух версий объектов базы данных
- Текущей версией объекта базы данных будем называть версию, созданную зафиксированной транзакцией с наиболее поздним временем фиксации. Незафиксированной версией объекта БД – версию, созданную еще незавершившейся транзакцией.
- В соответствии с протоколом 2V2PL, в каждый момент времени существует не более одной незафиксированной версии каждого объекта базы данных.

Версионный вариант протокола 2PL

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

1) Операция $R_i(o)$ немедленно выполняется над текущей версией объекта o
2) Операция $W_i(o)$, приводящая к созданию новой версии объекта o, выполняется только после завершения транзакции, создавшей незафиксированную версию объекта o
3) 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

Версионный вариант протокола 2PL

Для реализации такого поведения используются специальные виды синхронизационных 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.

Версионный вариант протокола 2PL

- 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.

	RL	**WL**	**CL**
RL	да	да	нет
WL	да	нет	нет
CL	нет	нет	нет

Версионно-блокировочный протокол сериализации транзакций для поддержки только читающих транзакций

- Multiversion Protocol for Read-Only Transactions, ROMV
- Гибридный протокол, поддерживающий эффективное выполнение транзакций, не изменяющих состояние базы данных
- При применении этого протокола при образовании каждой транзакции явно указывается её 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.

ROMV

- Изменяющие транзакции выполняются в соответствии с обычным протоколом 2PL, т.е. перед выполнением операции над объектом базы данных *o* этот объект должен быть заблокирован до конца изменяющей транзакции. Каждая операции записи объекта *o* создает его новую версию, которая при завершении транзакции помечается временной меткой, соответствующей моменту фиксации этой транзакции.

ROMV

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

ROMV

- Основным плюсом по сравнению с протоколом 2V2PL является принципиальное отсутствие синхронизационных задержек при выполнении операций чтения только читающих транзакций.
- В сравнении с MVTO, ROMV выигрывает в принципиальном отсутствии откатов только читающих транзакций.
- При использовании протокола ROMV в базе данных может возникать произвольное число версий объектов. Требуется создание специального сборщика мусора, который должен удалять ненужные версии данных.