# Лекция 15. Защита данных в реляционных СУБД: транзакции и блокировки.

15.1 Восстановление данных после аппаратных сбоев ..... 1
15.2 Управления транзакциями ..... 1
15.3 Параллельная обработка транзакций ..... 3
15.3.1 Проблемы, возникающие при параллельной обработке транзакций. ..... 3
15.3.2 Механизм блокировок для параллельного исполнения транзакций. ..... 4
15.3.2.1 Простая модель транзакции ..... 4

## *15.1 Восстановление данных после аппаратных сбоев*

Все обновления в системе баз данных происходят с помощью механизма транзакций.

> **Транзакция** – логическая единица работы с базой данных

В транзакцию может быть включено от одной до нескольких тысяч команд.

*Пример 1.*
*Предположим, необходимо обновить код поставщика (атрибут «n_номер») с S001 на S010. Для этого нужно выполнить два действия:*

1. *Обновить значение атрибута «n_номер» в отношении «ПОСТАВЩИКИ».*
2. *Обновить код поставщика в тех записях отношении «ПОСТАВКИ», в которых значение этого атрибута было равно S001, иначе нарушится общее правило целостности по связям.*

*Эти два действия над базой данных образуют одну транзакцию.*

В общем виде,

> **Транзакция** – последовательный набор команд SQL, образующих логически завершенный блок, который выполняется как единое целое.

В основе теории транзакций лежит такая модель транзакции:

Любая транзакция обладает четырьмя свойствами (их называют **свойствами АСИД**):

1. **Атомарность** – транзакция или выполняется целиком или не выполняется ничего.
2. **Согласованность** – транзакция переводит базу данных из согласованного в согласованное состояние.
3. **Изоляция** – транзакции отделены друг от друга. Любое обновление из любой транзакции будет скрыто до тех пор, пока эта транзакция еще выполняется.
4. **Долговечность** – если транзакция выполнена, то изменения, произошедшие в процессе ее работы, сохраняются, даже если в следующий момент происходит сбой системы.

Любая СУБД поддерживает так называемый **диспетчер транзакций**, который гарантирует выполнение свойств АСИД как при нормальной работе, так и после сбоев в системе.
Программист (и DBA) должен лишь выбрать нужный уровень изоляции транзакций и разработать эффективные алгоритмы обработки данных.

## *15.2 Управления транзакциями*

Язык SQL имеет операторы для управления ходом транзакций:

Оператор `BEGIN TRANSACTION` – начинает транзакцию.

Результатом выполнения транзакции может быть:

- оператор `COMMIT` – означает успешное окончание транзакции. Он сообщает диспетчеру о том, что база данных снова находится в согласованном состоянии, а все обновления могут стать постоянными, т.е. зафиксированными. Оператор `COMMIT` устанавливает т.н. «точку фиксации» (в коммерческих РСУБД - syncpoint), т.е. список обновлений признается зафиксированным.
- Оператор `ROLLBACK` – означает неудачное завершение транзакции. Он сообщает диспетчеру, что база данных находится в несогласованном состоянии, и все обновления могут быть аннулированы. `ROLLBACK` возвращает базу данных в предыдущую точку фиксации и последние обновления отменяются.

Список всех обновлений базы данных хранится в **журнале транзакций** (также его называют файлом регистрации). Следовательно, можно использовать журнальные записи для возвращения базы данных в последнее из согласованных состояний. Запись об операции в журнал осуществляется перед ее выполнением (т.н. **правило предварительной записи в журнал**). Это гарантирует, что даже если после выполнения операции `COMMIT` произойдет сбой системы, то после перезагрузки системы по записи в журнале все обновления восстанавливаются.

### 15.2.1 Протокол двухфазной фиксации для распределенных транзакций

> Транзакция называется **распределенной**, если она обращается к ресурсам, находящимся удаленно, и управляемым разными СУБД.

Менеджер транзакций при выполнении распределенной транзакции взаимодействует с несколькими администраторами ресурсов, например, разными СУБД; каждый из администраторов руководит собственным набором ресурсов и имеет свой журнал регистрации.

Протокол **двухфазной фиксации** позволяет гарантировать согласованные обновления данных во всех ресурсах.

*Пример 2.*
*Пусть транзакция запущена на MS SQL Server и модифицирует базу данных под управлением MS SQL Server и базу данных под управлением MS ACCESS. Если транзакция завершается успешно, то **все** ее обновления, для обоих СУБД, могут быть выполнены (если не произойдет сбоя после выполнения оператора `COMMIT`), иначе – все ее обновления могут быть отменены (если не произойдет сбоя после выполнения оператора `ROLLBACK`).*

Руководит процессом такой распределенной транзакции специальный координатор, который гарантирует выполнение свойств АСИД. Этот координатор запускается на том сервере, который инициализирует выполнения распределенной транзакции.

1-я фаза. **Подготовка**: каждый участник транзакции должен насильно сохранить все записи журнала для своих ресурсов.

Если насильственная запись прошла успешно, то участник возвращает координатору результат «OK», иначе возвращается результат «Not OK».

2-я фаза. **Выполнение**: когда координатор получает ответы от **всех** участников, он насильно заносит записи в соответствующий физический файл регистрации, указав свое решение относительно транзакции. Если **все** ответили «OK», то принимается решение «выполнить транзакцию», иначе (есть хотя бы один участник ответил «Not OK») – «вернуться назад». Затем

координатор сообщает свое решение участникам, и каждый участник согласно инструкции либо зафиксирует, либо аннулирует транзакцию.

*Замечание: т.к. участники и координатор транзакции работают в общем случае на разных компьютерах, то ошибка в работе координатора может привести к блокировке работы некоторого участника.*

## *15.3 Параллельная обработка транзакций*

### 15.3.1 Проблемы, возникающие при параллельной обработке транзакций.

Термин «параллелизм» означает возможность одновременного выполнения в СУБД нескольких транзакций с доступом к одним и тем же данным. При этом возникает масса проблем, связанных с попытками одновременного изменения или удаления данных.

*Пример 5. проблема последнего изменения (the lost update problem)*
*( p - отдельный кортеж таблицы )*

| | Транзакция A | Транзакция B | Транзакция C |
|---|---|---|---|
| T1 | Взять *p* | Взять *p* | Взять *p* |
| T2 | Изменить *p* | - | - |
| T3 | - | Изменить *p* | - |
| T4 | - | - | Изменить *p* |

Изменения, выполненные транзакциями A и B будут утеряны. Причинами «простоев» при выполнении транзакций B и C может быть, например, их старт с удаленных компьютеров сети, или ожидание в транзакции ввода пользователя.

*Пример 6. Проблема «грязного» чтения (the uncommitted dependency problem)*

| | Транзакция A | Транзакция B |
|---|---|---|
| T1 | Взять *p* | - |
| T2 | Изменить *p* | Взять *p* |
| T3 | Изменить *p* | - |

Транзакция B будет считывать данные, возможно, находящиеся в несогласованном состоянии, вызванном длительной транзакцией A и большим количеством изменений в транзакции A.

*Пример 7. Проблема неповторяемого чтения (the inconsistent analysis problem)*

| | Транзакция A | Транзакция B |
|---|---|---|
| T1 | Взять *p* | Взять *p* |
| T2 | - | Изменить *p* |
| T3 | Взять *p* | - |

Данные, считанные транзакцией A в первый раз, могут отличаться от данных, считанных ею во второй раз, т.к. в это время транзакция B успела изменить эти данные.

*Пример 8. Проблема чтения фантомов (the phantom read problem)*

| | Транзакция A | Транзакция B |
|---|---|---|
| T1 | Взять множество *p* | Взять множество *p* |
| T2 | - | Добавить *p'* к множеству *p* |
| T3 | Изменить множество *p* | - |

Для решения указанных проблем был разработан механизм блокировок.

### 15.3.2 Механизм блокировок для параллельного исполнения транзакций.

> **Блокировка (lock)** – временно накладываемое ограничение на выполнение некоторых операций обработки данных.

**Идея:** если для выполнения транзакции нужно, чтобы некоторый объект базы данных (кортеж) не изменялся без санкции этой транзакции, то такой объект блокируется.

Для каждой транзакции РСУБД накладывает на данные *блокировки*, которые обеспечивают выполнение требований ACID. В состав СУБД, как правило, входит **менеджер блокировок**.

С учетом механизма блокирования были разработаны несколько моделей транзакции.
Каждая из известных моделей транзакции должна обеспечивать «правильное» параллельное исполнение транзакций.

> **Расписание транзакций** – порядок, в котором выполняются элементарные шаги этих транзакций (как то: блокирование, разблокирование, чтение, запись).

«Правильность» параллельного исполнения транзакций определяется следующим образом:

> Параллельное исполнение транзакций называется **правильным** тогда и только тогда когда их совместный результат будет тем же самым, что и при исполнении этих транзакций последовательно, в некотором порядке.

> Расписание считается **последовательным**, если все шаги каждой транзакции выполняются последовательно.

> Расписание называется **сериализуемым**, если его результат эквивалентен результату некоторого последовательного расписания.

#### 15.3.2.1 Простая модель транзакции

В этой модели транзакция рассматривается как последовательность операций блокирования (`LOCK A`) и разблокирования (`UNLOCK A`) элементов. Каждый блокируемый элемент впоследствии должен быть разблокирован. Транзакция не пытается заблокировать уже заблокированный ею элемент, и также не пытается разблокировать элемент, уже ею разблокированный ранее. Предполагается, что после разблокирования элемента A этот элемент будет иметь уникальное значение, отличное от первоначального.

Сопоставляем каждой паре `LOCK A – UNLOCK A` однозначную функцию f, тогда
Если A0 – начальное значение элемента A, то

> f1(A0) – значение A после применения к нему 1-й пары `LOCK A – UNLOCK A`,
> fn(fn-1(…f1(A0)…))(n скобок) – значение A после применения к нему последовательно n пар `LOCK A – UNLOCK A`

Для проверки сериализуемости расписания транзакций, удовлетворяющих простой модели транзакции существует такой алгоритм:

*Вход*: расписание транзакций S вида

$a_1; a_2; \ldots a_n$

где:

$a_i$: $T_j$: LOCK $A_m$

$a_i$: $T_j$:UNLOCK $A_m$

$i$ – номер шага в расписании

$j$ – номер транзакции в расписании

*Выход*: ответ на вопрос, является ли расписание сериализуемым, и если да – то эквивалентное ему последовательное расписание.

*Метод:*

Построим ориентированный граф G, вершинами которого являются транзакции $T_1,\ldots$, $T_j,\ldots T_s,\ldots T_k$.

Дуга определяется так: для шага $a_i$ вида: $T_j$:UNLOCK $A_m$ ищем следующий за ним шаг $a_p$: $T_s$: LOCK $A_m$. Если такой шаг существует, то из $T_j$ в $T_s$ строится дуга.

Если в полученном графе обнаруживается цикл, то расписание S является несериализуемым.

При ацикличности графа можно найти порядок следования транзакций в эквивалентном последовательном расписании. Для этого используется алгоритм топологической сортировки: т.к. граф ацикличен, то в нем существует вершина $T_j$, у которой нет входящих дуг. Эта вершина помещается в список, и вычеркивается из графа. Процесс повторяется до тез пор, пока не будут перебраны все вершины без входящих дуг. В результате в полученном списке порядок перечисления вершин – транзакций даст последовательное расписание транзакций.

Можно сформулировать протокол, гарантирующий сериализуемость расписания транзакций, удовлетворяющих простой модели транзакции. Этот протокол называется **двухфазным**: 1-я фаза – блокирование, 2-я – разблокирование элемента.

**Протокол, гарантирующий сериализуемость, для простой модели транзакции:**

*Если потребовать, чтобы в транзакции всем операциям разблокирования предшествовали все операции блокирования элементов, причем в последовательности шагов разблокирования элементов не должно появиться шага, блокирующего элемент, и наоборот, то такой протокол гарантирует сериализуемость расписания транзакций.*

Усовершенствования моделей транзакции и соответствующие протоколы позволяют обеспечить более реалистичное отображение транзакций и также гарантировать сериализуемость расписания транзакций, удовлетворяющих новым моделям.

### 15.3.2.2 Модель транзакции с блокировками на чтение и на запись

Назовем **X-блокировкой** (eXclusive lock) блокировку без взаимного доступа (монопольную, или блокировку записи).

Назовем **S-блокировкой** (Shared lock) блокировку со взаимным доступом (блокировку чтения).

Причем:

1) Если транзакция A выполняет X-блокировку кортежа p, то запрос транзакции B на блокировку (любого типа) кортежа p будет отменен.
2) Если транзакция A выполняет S-блокировку кортежа p, то
   а) запрос транзакции B на X-блокировку отвергается
   б) запрос транзакции C на S-блокировку принимается.

Это дает возможность построить такую **матрицу совместимости блокировок**:

| **Тр.A \ Тр.B** | **X** | **S** | **-** |
|---|---|---|---|
| **X** | N | N | Y |
| **S** | N | Y | Y |
| **-** | Y | Y | Y |

где N – отмена запроса, Y – выполнение запроса

Эти блокировки используют таким образом:
1. Транзакция на извлечение кортежа использует S-блокировку.
2. Транзакция на обновления кортежа использует X-блокировку.
3. Если запрос на блокировку со стороны транзакции B отвергается из-за конфликта с блокировкой от транзакции A, то транзакция B переходит в **состояние ожидания**.
4. X-, S-блокировки сохраняются до конца транзакции или до тех пор, пока транзакция не снимет блокировку.

Тогда для такой модели транзакции также применяется двухфазный протокол, гарантирующий сериализуемость расписания транзакций:

**Протокол, гарантирующий сериализуемость, для модели транзакции с блокировками на чтение и на запись:**

*Установление блокировок для чтения и для записи предшествует всем операторам разблокирования.*

Американским институтом стандартов ANSI был разработан стандарт на уровни блокирования, состоящий из четырех уровней:

Уровень 0 – запрет «загрязнения» данных (no trashing of data)

Изменять данные может только одна транзакция. Если другой транзакции необходимо изменить эти же данные, она должна дождаться завершения первой транзакции.

Уровень 1 – запрет на «грязное» чтение (no dirty reads)

Если транзакция начала изменение данных, то никакая другая транзакция не сможет прочитать эти данные до тех пор, пока первая транзакция не завершится.

Уровень 2 – запрет неповторяемого чтения (no nonrepeatable reads)

Если транзакция считывает данные, то никакая другая транзакция не сможет их изменить. Таким образом, при повторном чтении данных они будут находиться в исходном состоянии.

Уровень 3 – запрет фантомов (no phantom)

Если транзакция обращается к данным, то никакая другая транзакция не сможет добавить/удалить строки, которые могут быть считаны при выполнении транзакции. Реализация этого уровня достигается блокированием *диапазона ключей*

Уровень 0 является самым слабым ограничением на одновременное обращение к данным. Уровень 3 – самый жесткий, но, соответственно, и гарантирующий максимальную надежность. Решение о том, какой уровень блокирования применить, принимает менеджер блокировок конкретной СУБД, однако программист имеет возможность задавать нужный уровень блокирования для конкретной транзакции, и даже для конкретной команды SQL (см. ниже).

Для управления работой менеджера блокировок существует четыре уровня изоляции: READ UNCOMMITTED (уровень 0), READ COMMITTED(уровень 1), REPEATABLE READ(уровень 2), SERIALIZABLE(уровень 3).

Для декларативной установки уровня изоляции используется команда

SET TRANSACTION ISOLATION LEVEL уровень изоляции

*Пример 9.*
*Рассмотрим ситуацию из примера 5.*
*Вот как ее можно разрешить с использованием аппарата блокировок:*

| Транзакция A | *Время* | *Транзакция B* |
|---|---|---|

| | | |
|---|---|---|
| *Взять кортеж p (S-блок.)* | *t1* | *-* |
| *-* | *t2* | *Взять кортеж p (S-блок.)* |
| *Обновить кортеж p (X-блок.)* | *t3* | *-* |
| *Ожидание* | *t4* | *Обновить кортеж p (X-блок.)* |
| *Ожидание* | *t5* | *Ожидание* |

*В момент времени t4 транзакция A переходит в состояние ожидания, а в следующий момент времени и транзакция B тоже переходит в состояние ожидания.*

> Ситуация, при которой каждая из множества двух или более транзакций ожидает, когда ей будет предоставлена возможность заблокировать элемент, уже заблокированный какой-либо другой транзакцией из данного множества, называется **тупиком**.

Способ разрешения тупиковой ситуации - выбор **транзакции-жертвы** и отмена ее выполнения.
Способы предотвращения тупиковых ситуаций сводятся к таким:

1) Потребовать, чтобы каждая транзакция единовременно запрашивала все нужные ей блокировки. Тогда менеджер транзакций либо удовлетворяет все запросы транзакции и устанавливает все блокировки, либо заставляет транзакцию ждать разблокирования нужных элементов.
2) Ввести произвольное линейное упорядочивание элементов и потребовать, чтобы все транзакции запрашивали блокировки в этом порядке.