БАЗЫ ДАННЫХ И ИНФОРМАЦИОННЫЕ СИСТЕМЫ

Ст.преп. каф. ИТ Кеберле Наталья Геннадьевна

Лекция 15. Защита данных в реляционных СУБД: транзакции и блокировки, параллельное выполнение транзакций.

На этой лекции

Мы выясним,

- как СУБД восстанавливает данные после аппаратных сбоев
- как СУБД обрабатывает одновременные доступы пользователей к данным (параллельный доступ)

Транзакция

Все обновления в системе баз данных происходят под контролем **механизма транзакций**.

Транзакция – логическая единица работы с базой данных.

В одной транзакции может быть от одной до нескольких тысяч команд.

Пример транзакции

В БД «Расписание авиарейсов» обновить номер рейса с 83 на 1083.

рейсы (

рейс	пункт_ отправл	пункт_ прибытия	время_ вылета	время_ прибыт
83	Нью-Йорк	Чикаго	10:15	12:28
84	Чикаго	Нью-Йорк	14:15	16:30
109	Нью-Йорк	Лос-Анджелес	21:50	2:52
213	Нью-Йорк	Бостон	11:43	12:45
214	Бостон	Нью-Йорк	14:20	15:12
117	Атланта	Бостон	06:30	09:35

)

варианты (

рейс	тип_ самолёта	пилот
83	727	Симонс
83	727	Хилл
83	747	Барт
83	747	Хилл
84	727	Симонс
84	727	Хилл
84	747	Барт
84	747	Хилл
109	707	Симонс

)

пригодность (

рейс	тип_ самолёта
83	727
83	747
84	727
84	747
109	707

)

Пример транзакции

В БД «Расписание авиарейсов» обновить номер рейса с 83 на 1083.

Действие1. Обновление номера рейса в таблице “*рейсы*”

рейсы (

рейс	пункт_ отправл	пункт_ прибытия	время_ вылета	время_ прибыт
83	Нью-Йорк	Чикаго	10:15	12:28
84	Чикаго	Нью-Йорк	14:15	16:30
109	Нью-Йорк	Лос-Анджелес	21:50	2:52
213	Нью-Йорк	Бостон	11:43	12:45
214	Бостон	Нью-Йорк	14:20	15:12
117	Атланта	Бостон	06:30	09:35

)

рейсы (

рейс	пункт_ отправл	пункт_ прибытия	время_ вылета	время_ прибыт
1083	Нью-Йорк	Чикаго	10:15	12:28
84	Чикаго	Нью-Йорк	14:15	16:30
109	Нью-Йорк	Лос-Анджелес	21:50	2:52
213	Нью-Йорк	Бостон	11:43	12:45
214	Бостон	Нью-Йорк	14:20	15:12

)

Пример транзакции

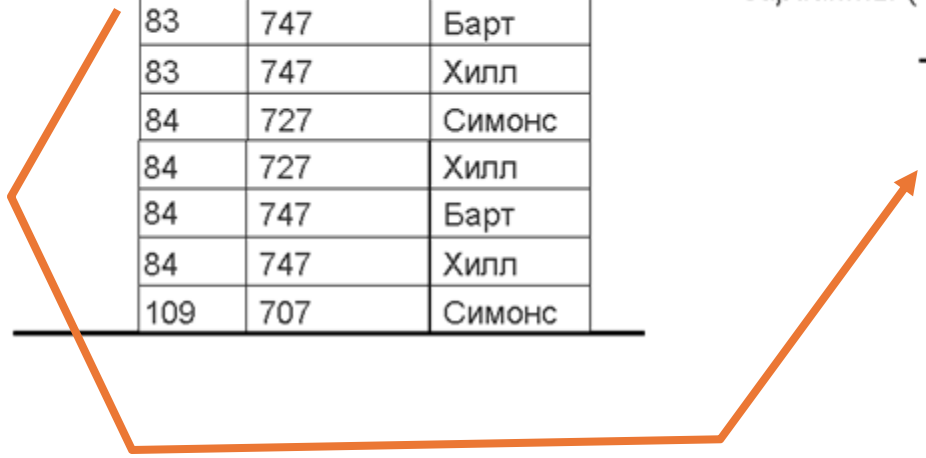
В БД «Расписание авиарейсов» обновить номер рейса с 83 на 1083.

Действие2. Обновление номера рейса в дочерних таблицах: обновление таблицы “*варианты*”

варианты (

рейс	тип_ самолёта	пилот
83	727	Симонс
83	727	Хилл
83	747	Барт
83	747	Хилл
84	727	Симонс
84	727	Хилл
84	747	Барт
84	747	Хилл
109	707	Симонс

)



варианты (

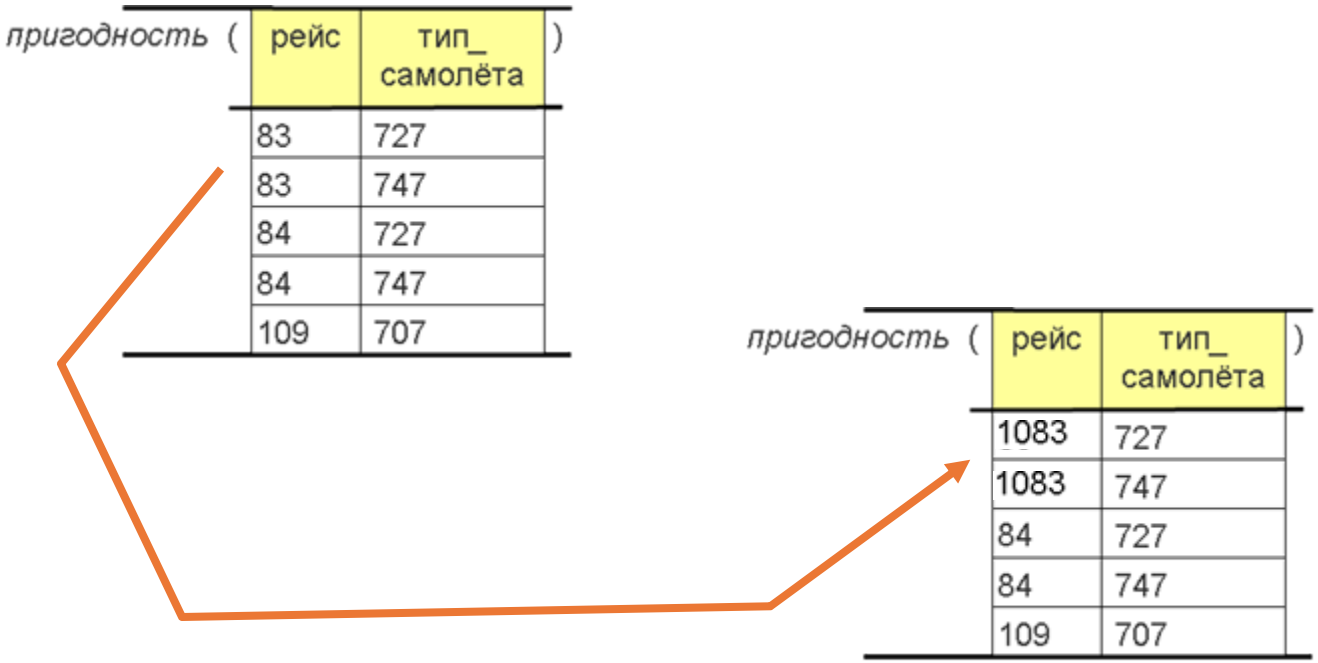
рейс	тип_ самолёта	пилот
1083	727	Симонс
1083	727	Хилл
1083	747	Барт
1083	747	Хилл
84	727	Симонс
84	727	Хилл
84	747	Барт
84	747	Хилл
109	707	Симонс

)

Пример транзакции

В БД «Расписание авиарейсов» обновить номер рейса с 83 на 1083.

Действие3. Обновление номера рейса в дочерних таблицах: обновление таблицы “*пригодность*”



Пример транзакции

В БД «Расписание авиарейсов» обновить номер рейса с 83 на 1083.

Действие1. Обновление номера рейса в таблице “*рейсы*”

Действие2. Обновление номера рейса в дочерних таблицах: обновление таблицы “*варианты*”

Действие3. Обновление номера рейса в дочерних таблицах: обновление таблицы “*пригодность*”

Эти три действия над базой данных образуют **одну транзакцию**. Если хотя бы одно из действий не будет выполнено – база данных будет находиться в несогласованном состоянии

Определение транзакции

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

Теория транзакций

В основе теории транзакций лежит такая модель:

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

- **Атомарность**
- **Согласованность**
- **Изоляция**
- **Долговечность**

Атомарность транзакции

Атомарность (atomicity) – транзакция или выполняется целиком или не выполняется ничего.

Т.е. если в транзакции 1000 команд SQL, и успешно выполнились 999 команд, а сбой произошёл в 1000-й команде – вся транзакция будет отменена

Согласованность транзакции

Согласованность (consistency) – транзакция переводит базу данных из согласованного в согласованное состояние.

В процессе работы транзакции база данных может быть несогласованна

Например, после выполнения действия 1, но перед выполнением действия 2 БД «Расписание авиарейсов» будет несогласованна.

Изоляция транзакций

Изоляция (isolation) – транзакции отделены друг от друга. Любое обновление из любой транзакции будет скрыто до тех пор, пока эта транзакция еще выполняется.

Изоляция транзакций запрещает пользоваться данными, которые могут быть несогласованны из-за работы другой транзакции

Изоляция - самое сложное по реализации свойство транзакции

Долговечность транзакций

Долговечность (durability) – если транзакция выполнена, то изменения, произошедшие в процессе ее работы, сохраняются, даже если в следующий момент происходит сбой системы.

Свойство долговечности гарантирует, что если транзакция завершилась успешно, то что бы ни случилось потом, данные будут изменены так, как это было предписано командами транзакции

Диспетчер транзакций

- 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, которые будут выполняться в транзакциях.

Транзакции и SQL

Язык SQL имеет операторы для управления ходом транзакций

- Начало транзакции
  - оператор BEGIN TRANSACTION
- Результатом выполнения транзакции может быть:
  - оператор COMMIT
  - оператор ROLLBACK

Успешное завершение транзакции

Оператор **COMMIT** означает **успешное окончание транзакции**

COMMIT сообщает диспетчеру о том, что база данных снова находится в согласованном состоянии, а все обновления могут стать постоянными, т.е. зафиксированными.

COMMIT устанавливает так называемую «**точку фиксации**» (в РСУБД - syncpoint), т.е. список обновлений признается зафиксированным.

Откат транзакции

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.

Журнал транзакций

- Список всех обновлений базы данных хранится в **журнале транзакций** (также его называют файлом регистрации).
- Журнальные записи используются СУБД для возвращения базы данных в последнее из согласованных состояний.
- Запись об операции в журнал осуществляется перед ее выполнением
  - **правило предварительной записи в журнал**
- Предварительная запись в журнал гарантирует, что даже если после выполнения операции COMMIT произойдет сбой системы, то после перезагрузки системы по записи в журнале все обновления восстанавливаются.

Журнал транзакций

- Как правило, отдельный хранимый файл или несколько хранимых файлов
  - Например, в MS SQL Server, этот файл может иметь расширение .log / .ldf
- Может храниться вместе с файлом данных, а может и отдельно (для большей безопасности)

Распределённые транзакции

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 ресурсов руководит собственным набором ресурсов и имеет свой журнал регистрации.

Протокол двухфазной фиксации

- Протокол **двухфазной фиксации** позволяет гарантировать согласованные обновления данных во всех ресурсах.
- Руководит процессом такой распределенной транзакции специальный координатор, который гарантирует выполнение свойств АСИД.
- Этот координатор распределённой транзакции запускается на том сервере, который инициализирует выполнения распределенной транзакции.

Протокол двухфазной фиксации

Фаза 1. Подготовка

Каждый участник транзакции должен принудительно сохранить все записи журнала для своих ресурсов.

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

Протокол двухфазной фиксации

Фаза 2. Выполнение

Когда координатор получает ответы от **всех** участников, он принудительно 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.

Пример распределённой транзакции

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).

Структура транзакции

BEGIN TRANSACTION

Команда 1

Команда 2

...

Команда N

If условие

ROLLBACK TRANSACTION

else COMMIT TRANSACTION

Параллельная обработка 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.

Проблемы параллельного доступа

Проблема последнего изменения (the lost update problem)

Время	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.

Проблемы параллельного доступа

Проблема «грязного» чтения (the uncommitted dependency problem)

Время	Транзакция A	Транзакция B
t1	взять *p*	
t2	изменить *p*	взять *p*
t3	изменить *p*	-
t4	-	-

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

Проблемы параллельного доступа

Проблема неповторяемого чтения (the inconsistent analysis problem)

Время	Транзакция A	Транзакция B
t1	взять *p*	взять *p*
t2	-	изменить *p*
t3	взять *p*	-
t4	-	-

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.

Проблемы параллельного доступа

Проблема чтения фантомов (the phantom read problem)

здесь {p} – множество кортежей,

p', p'' – отдельные кортежи

Время	Транзакция A	Транзакция B
t1	взять {*p*}	взять {*p*}
t2	-	{*p*} ∪ p'
t3	{*p*} ∪ p''	-
t4	-	-

p', p'' - эти кортежи становятся фантомами.

Механизм блокировок

- Для решения указанных проблем параллельного доступа был предложен механизм блокировок

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

Механизм блокировок

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

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

Модель транзакции с учётом блокировок

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

Расписания 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 т.и т.т.к.

их совместный результат будет тем же самым, что и при исполнении этих транзакций **последовательно**, в некотором порядке.

Расписания транзакций

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

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

Простая модель транзакции с учётом блокировок

- 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.

Простая модель транзакции с учётом блокировок

- 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

Проверка сериализуемости расписания

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

Вход: расписание транзакций S вида

a1; a2;…an

где:

ai: Tj: LOCK Am

ai: Tj: UNLOCK Am

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.

Проверка сериализуемости расписания

***Метод**:*

Построим ориентированный граф G, вершинами которого являются транзакции T1,…, Tj,…Ts,…Tk.

Дуга определяется так: для шага *ai* вида:

Tj:UNLOCK Am

ищем следующий за ним шаг *ap*:

Ts: LOCK Am.

Если такой шаг существует, то из Tj в Ts строится дуга.

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

Проверка сериализуемости расписания

- 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 Tj, 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.

Протокол, гарантирующий сериализуемость

- Можно сформулировать протокол, гарантирующий сериализуемость расписания транзакций, удовлетворяющих простой модели транзакции.
- Этот протокол называется (также) **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.

Другие протоколы

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

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).

Модель транзакции с блокировками на чтение и на запись

Причем справедливо:

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

Модель транзакции с блокировками на чтение и на запись

Матрица совместимости блокировок показывает возможные сочетания:

Транзакция B / Транзакция A	X	S	-
X	нет	нет	да
S	нет	да	да
-	да	да	да

где «нет» – отмена запроса на блокировку,

«да» – выполнение запроса на блокировку

Модель транзакции с блокировками на чтение и на запись

Эти блокировки используют таким образом:

- Транзакция на чтание кортежа использует S-блокировку.
- Транзакция на обновления кортежа использует X-блокировку.
- Если запрос на блокировку со стороны транзакции B отвергается из-за конфликта с блокировкой от транзакции A, то транзакция B переходит в **состояние ожидания**.
- X-, S-блокировки сохраняются до конца транзакции или до тех пор, пока транзакция не снимет блокировку.

Модель транзакции с блокировками на чтение и на запись

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.

Уровни блокировок (стандарт ANSI)

- Американским институтом стандартов ANSI был разработан стандарт на уровни блокирования, состоящий из четырех уровней:
  - Уровень 0 – запрет «загрязнения» данных (no trashing of data)
  - Уровень 1 – запрет на «грязное» чтение (no dirty reads)
  - Уровень 2 – запрет неповторяемого чтения (no nonrepeatable reads)
  - Уровень 3 – запрет фантомов (no phantom)

Уровень 0

Уровень 0 – запрет «загрязнения» данных

Изменять данные может только одна транзакция.

Если другой транзакции необходимо изменить эти же данные, она должна дождаться завершения первой транзакции.

Установка этого уровня решает проблему

Время	Транзакция A	Транзакция B	Транзакция C
t1	взять *p*	взять *p*	взять *p*
t2	изменить *p*	-	-
t3	-	изменить *p*	-
t4	-	-	изменить *p*

Уровень 1

Уровень 1 – запрет на «грязное» чтение

Если транзакция начала изменение данных,

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

Установка этого уровня решает проблему

Время	Транзакция A	Транзакция B
t1	взять *p*	
t2	изменить *p*	взять *p*
t3	изменить *p*	-
t4	-	-

Уровень 2

Уровень 2 – запрет неповторяемого чтения

Если транзакция считывает данные,

то никакая другая транзакция не сможет их изменить.

Таким образом, при повторном чтении данных они будут находиться в исходном состоянии.

Установка этого уровня решает проблему

Время	Транзакция A	Транзакция B
t1	взять *p*	взять *p*
t2	-	изменить *p*
t3	взять *p*	-
t4	-	-

Уровень 3

Уровень 3 – запрет фантомов

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**

Установка этого уровня решает проблему

Время	Транзакция A	Транзакция B
t1	взять {*p*}	взять {*p*}
t2	-	{*p*} ∪ p'
t3	{*p*} ∪ p"	-
t4	-	-

Уровни блокировок (стандарт ANSI)

- 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.

Управление изоляцией в SQL

- 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

Тупики и транзакции-жертвы

- 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.

Пример тупика

- Взаимное ожидание действий двух транзакций

Время	Транзакция A	Транзакция B
t1	(S) взять *p*	-
t2	-	(S) взять *p*
t3	(X) изменить *p*	-
t4	ожидание	(X) изменить *p*
t5	ожидание	ожидание

Предотвращение тупиков

Способы предотвращения тупиковых ситуаций сводятся к таким:

- 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.
- Ввести произвольное линейное упорядочивание элементов и потребовать, чтобы все транзакции запрашивали блокировки в этом порядке.

Итоги

- Познакомились с механизмом проведения согласованных обновлений системы баз данных - механизмом транзакций и блокировок
- Познакомились со свойствами транзакций
- Узнали о проблемах параллельного доступа к данным и о способах их решения

Вопросы:

- Вариант1

  1) определение транзакции

  2) пример нарушения свойства изоляции транзакции
- Вариант2

  1) модели транзакции

  2) виды блокировок
- Вариант3

  1) пример нарушения свойства атомарности транзакции

  2) что такое сериализуемое расписание транзакций
- Вариант4

  1) пример нарушения свойства согласованности транзакции

  2) что значит, что расписание несериализуемое, и что с таким расписанием делать