Транзакции в базах данных

Бессарабов Н.В.
bes@fpm.kubsu.ru
2018 г.

Цели лекции

Базы данных, на которых строятся корпоративные информационные системы должны обеспечить одновременный и независимый доступ к данным для сотен и тысяч пользователей. Оказывается, при параллельной работе пользователей результаты вычислений могут зависеть от временн`ых соотношений между действиями пользователей.

Как избежать этой зависимости? Можно ли организовать чтение данных одним пользователем и одновременно их изменение другим? Как добиться, чтобы деньги, снятые с одного счета, и из-за сбоя системы не зачисленные на другой счет, не могли пропасть бесследно?

Можно ли восстановить данные при отказах и сбоях?

Все эти проблемы решаются за счёт использования **механизма транзакций**, работающего в базах с любыми моделями данных.

Транзакции обеспечивают:

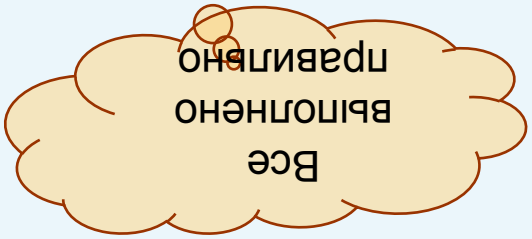
1. Сохранение целостности данных.
2. Параллельную работу пользователей с базой данных.
3. Восстановление данных при отказах и сбоях.

В этой лекции будут рассмотрены транзакции и часть вопросов, связанных с перечисленными тремя аспектами их применения. Тема довольно сложна для начинающих, в частности, из-за необходимости рассмотрения процессов во времени.

Сбой при выполнении связанных действий

Снятие 300 р. со счета А
Зачисление 300 р.на счет В

Снятие 300 р. со счета А
Сбой сервера, зачисление не может быть выполнено



Время

Следовательно

Необходим механизм, позволяющий исключить такую возможность

Ошибка при параллельной работе пользователей. Сальдо по группе счетов

П1: Подсчёт сальдо С

П2: Перевод со счета №1 на счет №1000

0

t

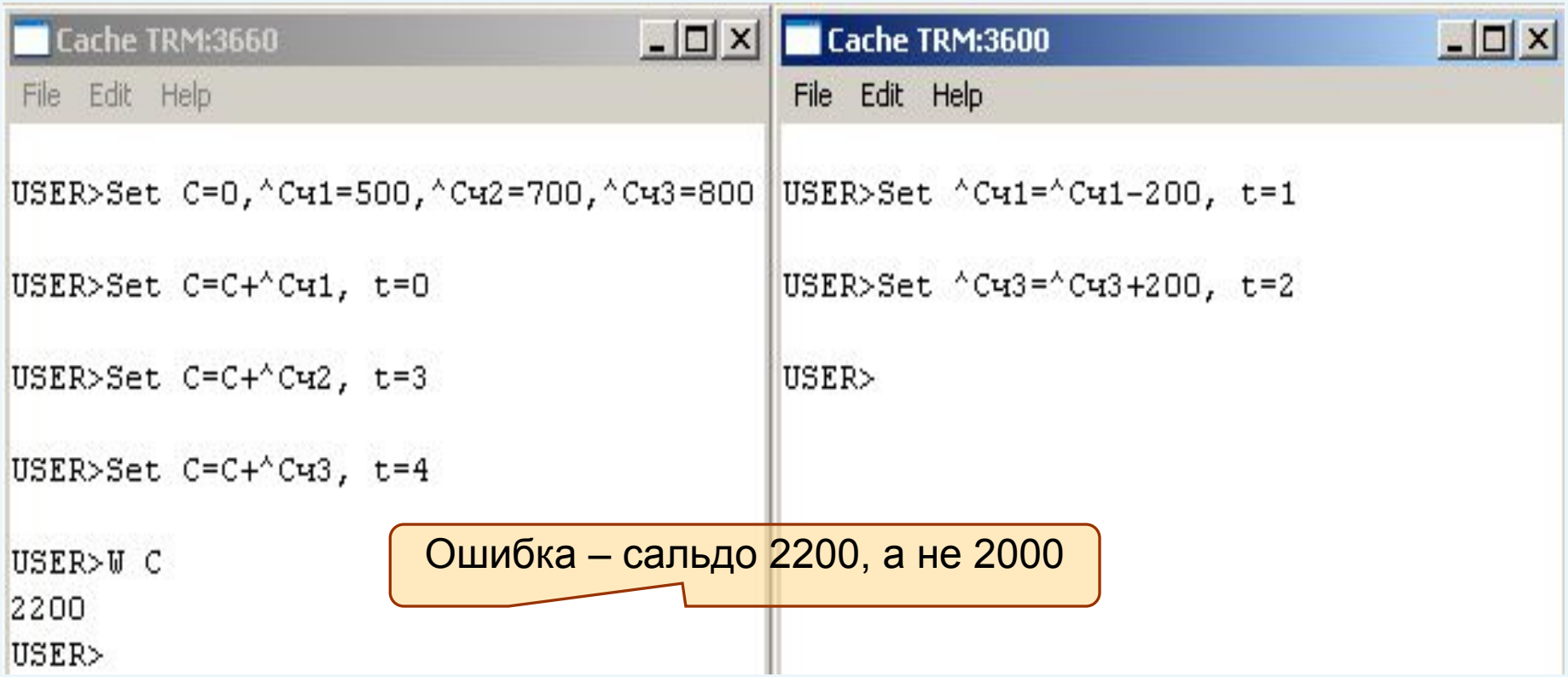
```
C=0
C = C + Сч1
C = C + Сч2
C = C + Сч3
……..
……..

C = C +
Сч1000
```

```
Сч1 = Сч1 – 200
Сч1000 = Сч1000 + 200
```

Ошибка подсчета сальдо С составила +200

Необходим механизм, позволяющий исключить такие ошибки

Моделируем параллельную работу двух пользователей



Сальдо по группе счетов ^Сч1, ^Сч2, ^Сч3. Два запущенных терминала моделируют работу двух пользователей. Присваивание переменной t значений 0,1,2,3,4 просто отмечает порядок выполнения действий. Так, первая строка в терминале справа (с присваиванием t=1) выполняется после второй строки в левом терминале (с присваиванием t=0)

Простейшая транзакция

Вернемся к рассмотренной ранее операции -- переводу денег с одного счета на другой. Эта операция включает две атомарные (элементарные, неделимые) операции:

- снятие суммы с одного счета,
- зачисление суммы на другой счет.

Если СУБД будет выполнять (или не выполнять) эти две операции только вместе, то база будет находиться в **согласованном состоянии** всегда за исключением промежутка времени выполнения этих операций.

Не будут возникать ситуации при которых суммы, снятые с одного счета, ни на какой другой счет не поступают. Объединение нескольких операций в неделимое целое называют **транзакцией**.

Вопрос: Что делать, если часть данных уже изменена, а остальные измениться не могут?

Ответ: Откатить выполненные изменения, то есть вернуться к ранее существовавшим значениям. Но как?

Определение: База находится в согласованном состоянии, если выполняются все записанные в ней ограничения целостности.

Во время выполнения транзакций база может рассогласовываться.

Вопрос: Какое ограничение целостности нарушается в последнем рассмотренном примере?

Чего ждем от транзакций

Мы уже говорили о том, что транзакционный механизм должен обеспечить:

1. Сохранение целостности данных
2. Параллельную работу пользователей
3. Восстановление данных при отказах и сбоях

При отказах информационной системы желательно сохранить сведения о попытке выполнения транзакции. Если при этом транзакция была выполнена частично, то уже выполненную часть операций необходимо **откатить**. После восстановления системы транзакция должна быть выполнена повторно.

Определение транзакции

Определение: **Транзакция** – это последовательность операторов, выполняющаяся как единое целое и переводящая базу данных из одного **целостного состояния** в другое **целостное состояние.**

В транзакциях используют следующие операторы:

- чтения данных;
- манипулирования данными;
- **блокирования и разблокирования ресурсов**.

Уточнение: Употреблённый в определении термин “последовательность” не означает, что по структуре транзакция это обязательно линейная цепочка операторов, но каждое исполнение транзакции – это именно последовательность действий.

Вспоминаем

Определение: База данных находится в **согласованном (целостном) состоянии**, если выполнены все определённые в ней ограничения целостности.

Свойства транзакций

Свойства АСИД:

(А) Атомарность. Транзакция выполняется как единое целое (либо все выполняется, либо все не выполняется).

(С) Согласованность. Транзакция переводит базу данных из одного согласованного состояния в другое согласованное состояние. Внутри транзакции согласованность базы данных может нарушаться.

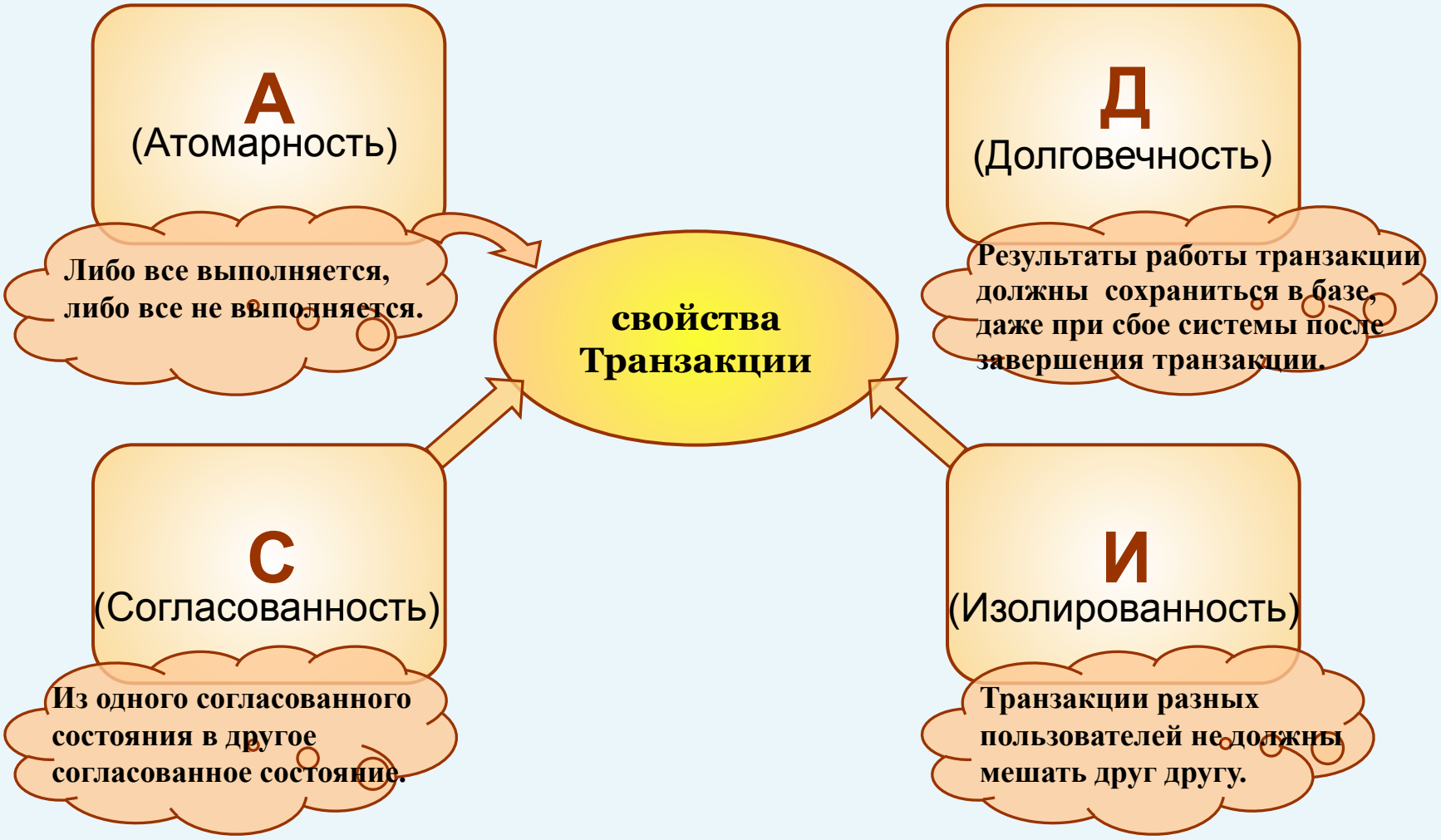
(И) Изолированность. Транзакции разных пользователей не должны мешать друг другу.

(Д) Долговечность. Результаты работы выполненной транзакции должны сохраниться в базе данных, даже если по завершении транзакции произойдет сбой системы.

Замечание: Соответствующая английская аббревиатура ACID (Atomicity, Consistency, Isolation, Durability)

Основной автор рассматриваемой модели транзакций **Джеймс Николас «Джим» Грей** (James Nicholas "Jim" Gray) награждённый премией Тьюринга за вклад в развитие баз данных в 1998 году за разработки конца 70-х годов.

Свойства АСИД (1/2)



Замечание о свойствах АСИД

Свойства АСИД транзакций за исключением атомарности не всегда выполняются в полном объеме.
Согласованность. Нарушается внутри транзакции. При неполной изоляции транзакций несогласованные данные могут быть доступны другим транзакциям.
Изоляция. Полной изоляции транзакций можно достигнуть только если принудительно выстраивать транзакции в очередь и исполнять их строго одну после другой. При этом достаточно одной из транзакций зависнуть (“сначала я пила кофе, потом меня вызвал начальник, а потом был перерыв”) или выполняться слишком долго, чтобы остановить работу. Поэтому вводится несколько уровней неполной изоляции.
Долговечность. Может нарушаться, например, если транзакция T2, вызванная из транзакции T1 будет завершена успешно, а T1 откатится.

Оформление транзакций в языках

Два способа запуска транзакции:

1. Транзакция начинается автоматически с момента присоединения пользователя к СУБД или по завершении предыдущей транзакции (например, в Oracle).
2. Транзакция начинается специальной командой BEGIN TRANSACTION или другой командой аналогичного назначения (например, в Caché).

Транзакция завершается (успешно или не успешно) специальными командами или по событию:

1. Команда COMMIT [WORK] (зафиксировать транзакцию).
2. Команда ROLLBACK [WORK] (откатить транзакцию).
3. Событие “Отключение пользователя от СУБД”.
4. Событие “Сбой системы”.
5. Событие “Отключение системы”

В языке COS начало транзакции TStart, успешное завершение TCommit, неуспешное TRollback.

Транзакции в Caché

Команда SQL	Команда COS	Описание
%BEGTRANS	TStart	Начало транзакции
COMMIT WORK	TCommit	Успешное завершение транзакции
ROLLBACK WORK	TRollback	Неуспешное завершение транзакции

Замечание 1: В Oracle любые действия вкладываются в транзакцию, которая всегда начинается неявно (без команды начала).
Замечание 2: В Caché транзакции используются по желанию пользователя, то есть можно работать с данными, находясь вне какой-либо транзакции. Такой пользователь как бы “не замечает“, чужих транзакций. Поэтому в Caché можно экспериментально исследовать сам механизм транзакций.

Простейшая транзакция (программа ^tr1)

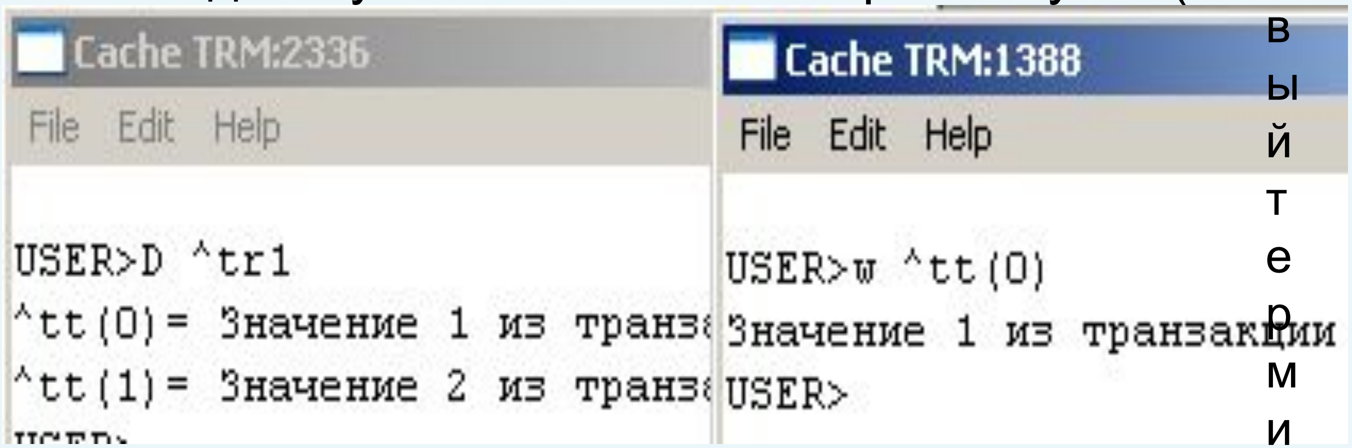
```
Set ^tt(0)=0, ^tt(1)=1  ;значения до транзакции
TStart              ;начало транзакции
Set ^tt(0)= "Значение 1 из транзакции“
Read x                      ;приостанавливает исполнение
Set ^tt(1)= "Значение 2 из транзакции"
TCommit                  /* конец транзакции -
                                успешное завершение */
Write "^tt(0)= ",^tt(0),!, "^tt(1)= ", ^tt(1)
Quit
```

Транзакция

Замечание: Команда Read позволяет приостановить работу программы. Это позволяет задавать (или имитировать) сдвиг во времени, а также прервать транзакцию, не завершая её командой Tcommit или Trollback.

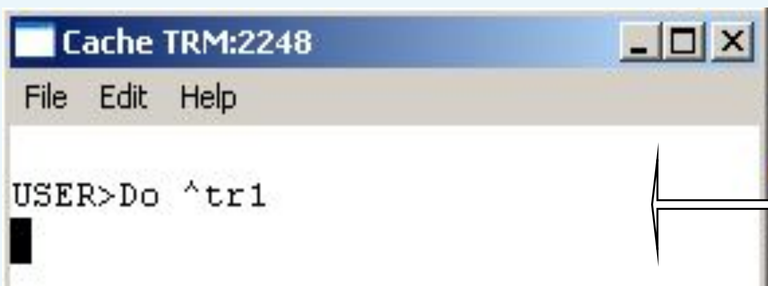
Варианты исполнения транзакции

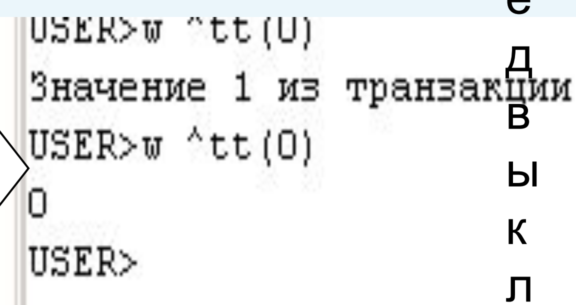
1. С вводом пустого значения в переменную х (левый терминал)



Сначала в левом терминале запускается программа ^tr1, затем в правом читается значение глобала

2. С выключением левого терминала (нажать на ✕)





н а л а , а в т о - р й р а з п о с л е е г о в

л е в ы й т е р м и н а л п е р е д в ы к л ю ч

I. Транзакции и ограничения целостности

В этом разделе рассмотрим реакции СУБД на нарушения целостности.

Приведем примеры ограничений целостности, в том числе ссылочных.

Изучим классификацию ограничений целостности по способам реализации, по времени проверки и по области действия.

В частности, выясним что такое триггеры базы данных.

Нарушения целостности

Уточняем определение: База данных находится в **согласованном (целостном) состоянии**, если выполнены все (процедурные и декларативные) ограничения целостности, определенные в реализации базы данных.

Для распределённых баз данных важна ещё согласованность копий фрагментов данных в узлах. Меру различия между копиями данных в узлах сети называют **связностью** данных. В распределённой базе **невозможно обеспечить одновременно высокую связность и высокую скорость работы**.

Система управления базами данных обязана **реагировать на любые попытки нарушения целостности**.

Два основных типа реакции СУБД:

- Отказ выполнить "недопустимую" операцию.
- Выполнение действий компенсирующих нарушения ограничений целостности . Вариант реализуется процедурно.

Давайте выделим основные виды ограничений целостности.

Классификация ограничений целостности

Основания классификации:

1. По **способам реализации**.
2. По **времени проверки**.
3. По **области действия**.

Замечание: Любые ограничения целостности расширяют семантику данных, поддерживаемую в базе данных

1. Классификация **по способам реализации**

- **Декларативная** поддержка ограничений целостности.
- **Процедурная** поддержка ограничений целостности.

Декларативная поддержка ограничений целостности реализуется определениями ограничений средствами языка ЯОД определения данных (DDL - Data Definition Language).

Пример (для языка SQL):

CREATE TABLE PERSON (PersId INTEGER **PRIMARY KEY**, PersName CHAR(30) **NOT NULL**);

.

Процедурная поддержка ограничений целостности

Процедурные ограничения целостности реализуются использованием триггеров и хранимых процедур.

Пример ограничения целостности обычно реализуемого процедурно: “Начальник отдела может изменять заработную плату только своим подчиненным и только в сторону уменьшения”.

Деление на процедурные и декларативные ограничения целостности условно, так как оба вида ограничений в действительности поддерживаются специальными процедурами.

Просто в каждой СУБД определен список ограничений, которые можно записать декларативно.

Другие ограничения реализуются явным заданием в процедурной части приложения.

Не следует думать, что в любой СУБД реализуются любые мыслимые ограничения целостности!

Триггеры

Триггеры это процедуры специального вида, прикрепленные к объекту базы, обычно таблице, и срабатывающие при наступлении одного события из некоторого набора событий. Наиболее известны триггеры реализуемые в языке SQL.

В набор событий обычно входят вставка записи (оператор INSERT в SQL), удаление записи (оператор DELETE) и обновление записи (оператор UPDATE).

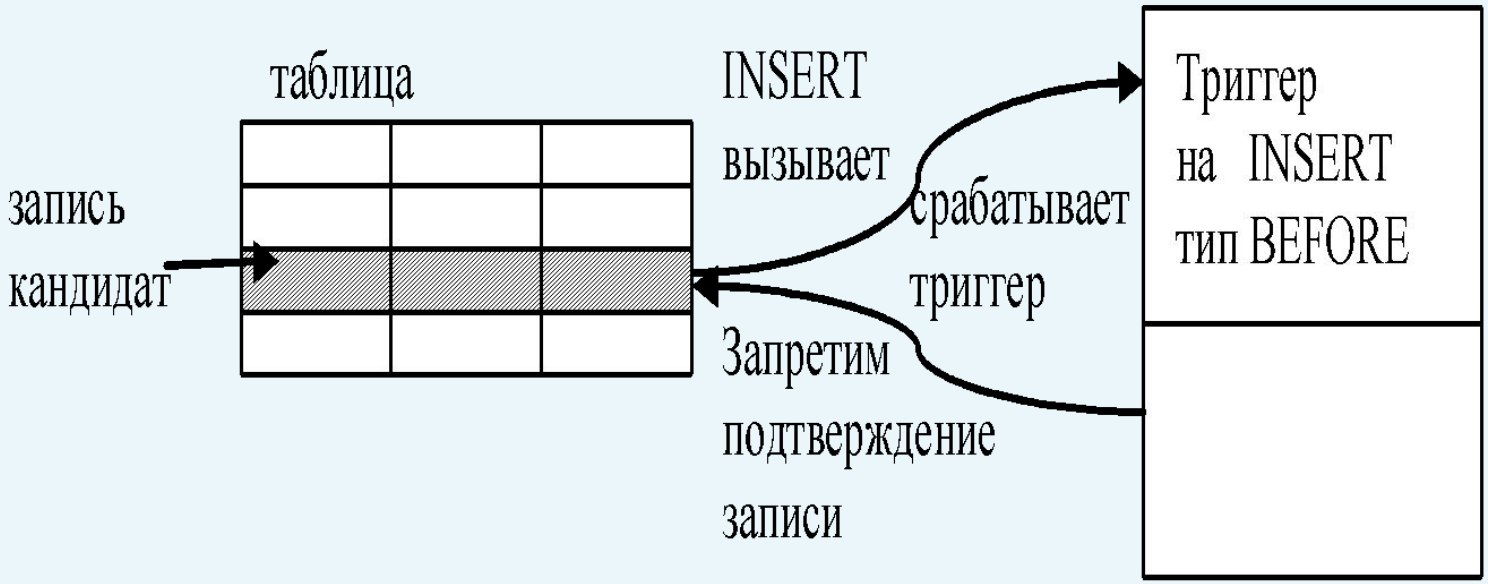
Триггер может выполняться до отработки события (триггер BEFORE) и после его отработки (триггер AFTER).

Выделяют также триггеры уровня строки (записи), срабатывающие один раз на каждую выбранную строку, и триггеры уровня выражения, срабатывающие один раз при обработке набора записей.

Итого 12 типов триггеров на один набор записей (таблицу).

но не штук! Триггеров одного типа м.б.несколько.

Пример срабатывания строчного триггера



Для проверки ограничения на возраст принимаемого на работу создают триггер на команду вставки записи INSERT, срабатывающий до ее выполнения (нельзя допускать запись о неправильном приеме – значит тип - before). В теле триггера вычисляют возраст, проверяют условие 21 ≤ возраст ≤ 35. Если условие не выполнено, попытка записи о приеме аннулируется.

Пример ссылочного ограничения целостности

Перечень отделов в таблице
DEP(DepId, DepName, DeptQuan),
где
DeptId – ид. подразделения,
DeptName – название подразделения
DeptQuan – кол-во сотрудников
в подразделении.

Таблица DEP

DeptId	DeptName	DeptQuan
1	Цех розлива	3
2	Транспортный цех	2

Список сотрудников в таблице
PERSON(PersId, PersName, DeptId),
где PersId – ид. сотрудника,
PersName - имя сотрудника,
DeptId – ид. подразделения,
в котором работает сотрудник.

Таблица PERSON

PersId	PersName	DeptId
1	Иванов	1
2	Петров	2
3	Сидоров	1
4	Злобис	2
5	Вредис	1

Пожелание: В конце курса, когда Вы начнёте разбираться с семантикой данных, обратите внимание на странность этого примера вызванную отсутствием временн`ых данных.

Описание к примеру ограничения ссылочной целостности

Ограничение ссылочной целостности: поле DeptQuan должно содержать количество сотрудников, записанных в таблице PERSON для данного отдела.

Например, прием нового сотрудника **не может быть выполнен одной операцией**. Необходимо вставить запись в таблицу PERSON и одновременно увеличить на 1 значение поля DeptQuan в одной из строк.

Выполняем два шага, которые конечно же необходимо включить в транзакцию:

Шаг 1. Вставить запись о сотруднике в таблицу PERSON

INSERT INTO PERSON VALUES (6, “Петросян”, 1).

Шаг 2. Увеличить значение поля DeptQuan командой

UPDATE DEPART SET DeptQuan=DeptQuan+1 WHERE Dept_Id=1

Заметим, что для увольнения и перемещения сотрудников необходимы свои транзакции.

Замечание: Вам ещё не известны операторы select, insert и update языка SQL. Воспринимайте их пока просто как фразы написанные на фрагменте естественного (как бы английского) языка, который понимает СУБД.

Примеры ограничений целостности

Пример 1. Возраст сотрудника не может быть меньше 21 и больше 45 лет (Декларативное ограничение типа check).
Пример 2. Атрибут “Табельный номер” уникален. (Декларативное ограничение -- уникальный или первичный ключ).
Пример 3. Сотрудник обязан числиться в одном из отделов. (Декларативное ссылочное ограничение целостности).
Пример 4. Сумма накладной равняется сумме произведений цен товаров на количество товаров для всех товаров, входящих в накладную. (Определение вычислимого столбца. Обычно процедурное ограничение).

С вычислимыми столбцами связано необычное ограничение целостности: “Сущность, выделяемая по теореме Хиса должна быть осмысленной”. Например в сущности “строка_заказа” имеется функциональная зависимость из столбцов “price” и “amount” в вычислимый столбец “subtotals”, но таблица с этой тройкой столбцов не образует осмысленной сущности.

Замечание: Ограничение целостности может определять не только значения атрибутов, но и особенности удаления или обновления кортежей. Например, в схеме “Отдел” – “Сотрудник” удаление отдела может вызвать удаление его сотрудников или их перевод в другие отделы.

Классификация ограничений целостности по времени проверки

Продолжаем изучать классификацию ограничений целостности.

По **времени проверки** выделяют два вида ограничений:

1. **Немедленно проверяемые** ограничения. Проверяются непосредственно в момент выполнения операции, могущей нарушить ограничение.
   Пример: проверка уникальности значения PK.

2. Ограничения **с отложенной проверкой**. Проверяются в момент фиксации транзакции оператором COMMIT.
   Пример: проверка ссылочного ограничения целостности в примере на слайдах 22-23.

Классификация ограничений целостности по области действия

1. Ограничения домена (из-за отсутствия в СУБД поддержки доменов обычно переносятся на атрибуты).
2. Ограничения атрибута (немедленно проверяемые ограничения на допустимые значения атрибута). Реализуются почти всегда декларативно. Проверяют обычно попадание в диапазон или в список. Проверки на соответствие шаблону, например, в записи телефонного номера в настоящее время реализуются с помощью регулярных выражений.
3. Ограничения кортежа. Это ограничения на соотношение атрибутов одного экземпляра сущности.
4. Ограничения отношения (или сущности или таблицы). Для проверки ограничения необходимо обработать все кортежи отношения.

   Пример : “Только у двух человек заработная плата может быть больше 1000 “
5. Ограничения на допустимые связи.
6. Ограничения базы или схемы данных (межтабличные). Накладываются на значения двух или более связанных между собой отношений. Пример: ссылочная целостность.

Ограничения на допустимые связи

- Поскольку связи в реляционной модели не выделены как “существа” рядоположные сущностям, то ограничения на допустимые связи могут определяться только как свойства самих связываемых сущностей.
- В типичном случае имеется несколько взаимно исключающих связей. Например, сущность A может быть связана бинарной связью с B или C, но если существует связь A с B, то не может существовать связь A с C, и наоборот.
- Возможно, наличие или отсутствие связи определяется состоянием экземпляров сущностей, которые могут входить в связь. Под состоянием здесь понимаются значения атрибутов сущностей. Заметим, что реализация рассматриваемого класса ограничений достаточно сложна.
- Одна из возможных связей – наследование. Реализация ограничений на наследование будет рассмотрена позже при изучении объектных моделей.
- В EER имеется псевдонаследование – категория.

Классификация ограничений целостности (1/2)

II. Транзакции и параллельная работа транзакций

Рассмотрим проблемы возникающие при параллельной работе транзакций. В соответствии с традицией будем называть эти проблемы феноменами.

На основе феноменов зададим уровни изолированности транзакций. В определениях изолированности будет использован стандарт версии SQL-92 языка SQL.

Для других моделей данных соответствующие стандарты отсутствуют.

В заключительной части раздела рассмотрим блокировки ресурсов и эффекты возникающие при блокировании.

Феномен “Потерянные изменения”

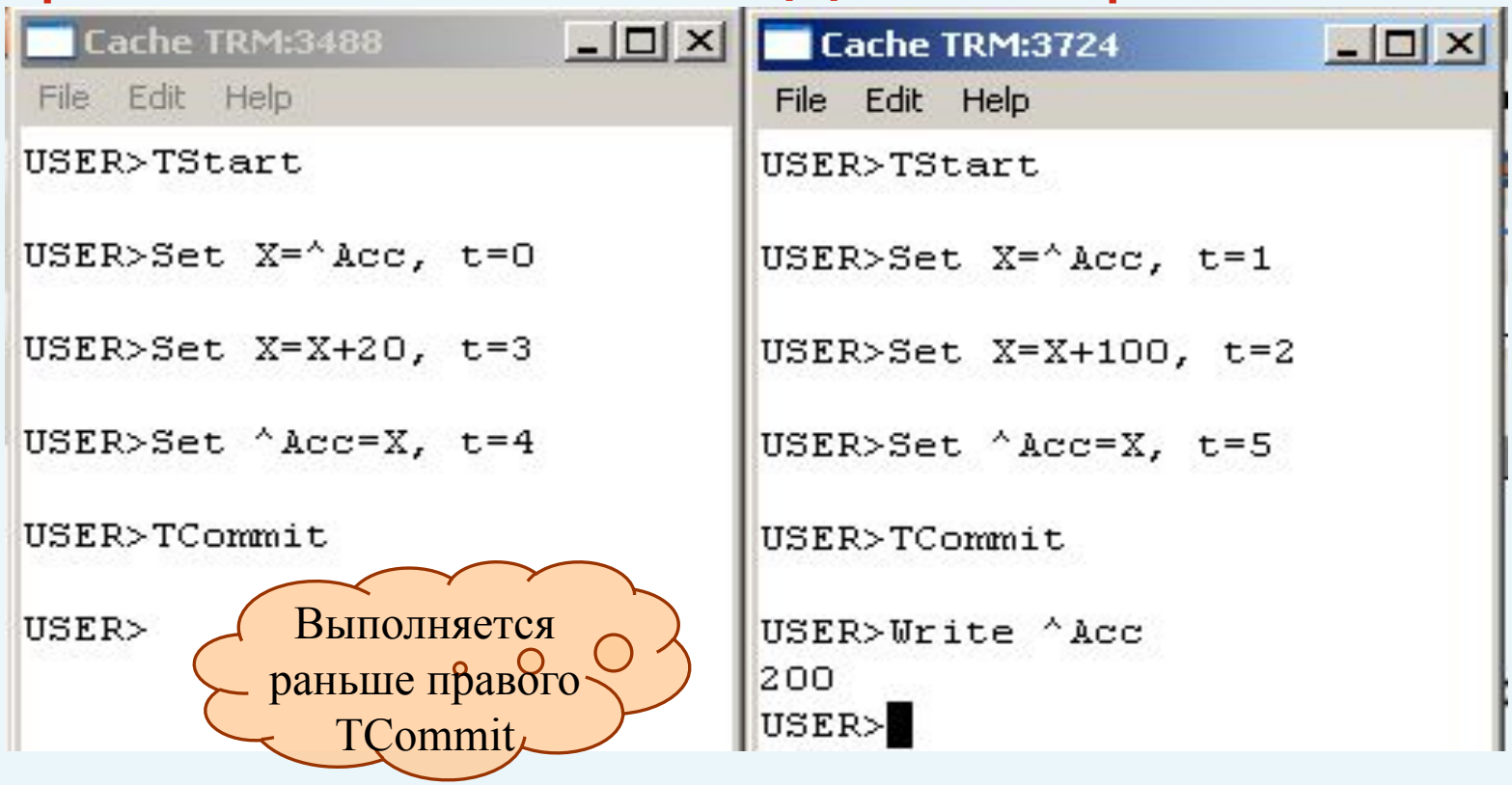
При отсутствии блокировок или других средств, ограничивающих доступ к ресурсам используемым транзакцией, некоторые изменения могут быть утеряны.

Пример: На счёт Acc с начальным сальдо 100, зачисляют первой транзакцией 20 руб., а второй 100 руб. Изменения, внесенные первой транзакцией, теряются.

Замечание: В каждой транзакции переменная X своя!

Существенно, что в первой транзакции commit производится до записи в базу второй транзакцией своих внутренних данных

“Потерянные изменения”.Демонстрация в Caché



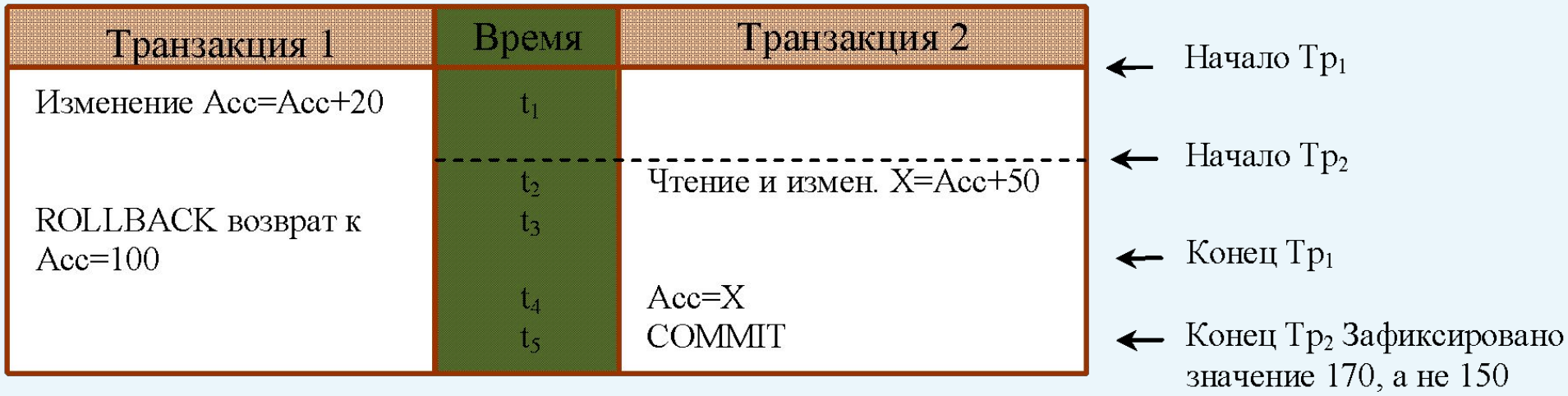
Начальное значение ^Acc=100. Если несколько транзакций до завершения некоторых из них читают одни и те же данные, то в итоге полученные значения определит последняя комитованная транзакция.

TStart – начало транзакции, TCommit – успешное завершение.

Переменная t использована вместо комментария для задания отметки очерёдности, соответствующей моментам времени t_0, t_1, …

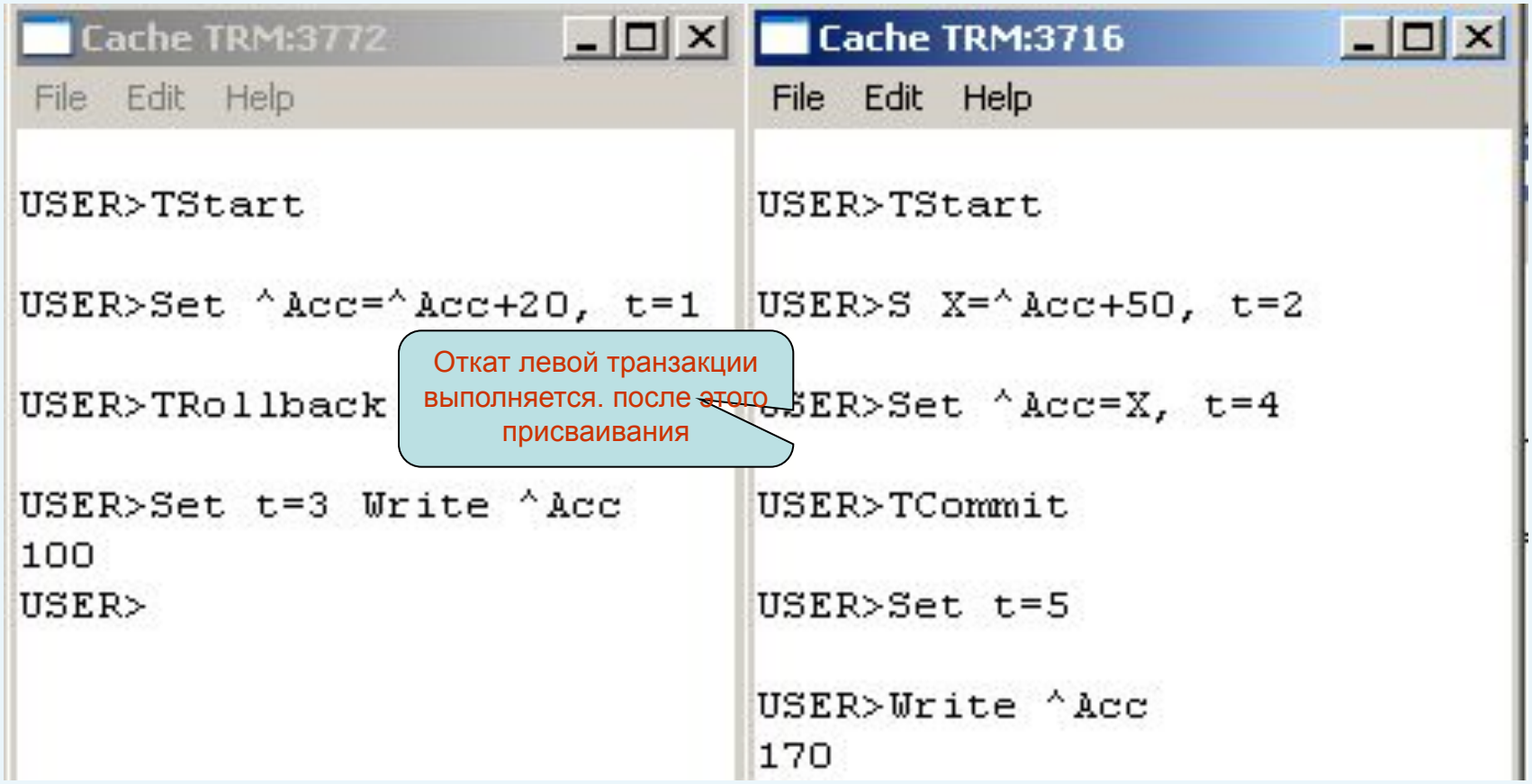
Зависимость от незафиксированных результатов (чтение “грязных” данных)

Возникает, когда читаются данные незавершённой транзакции, которые впоследствии откатываются. Начальное значение Acc в примере равно 100.



Существенно, что чтение второй транзакции производится после изменения данных первой транзакцией, но до отката первой транзакции

Чтение “грязных” данных – демонстрация в Caché



Феномен “Неповторяющееся чтение”

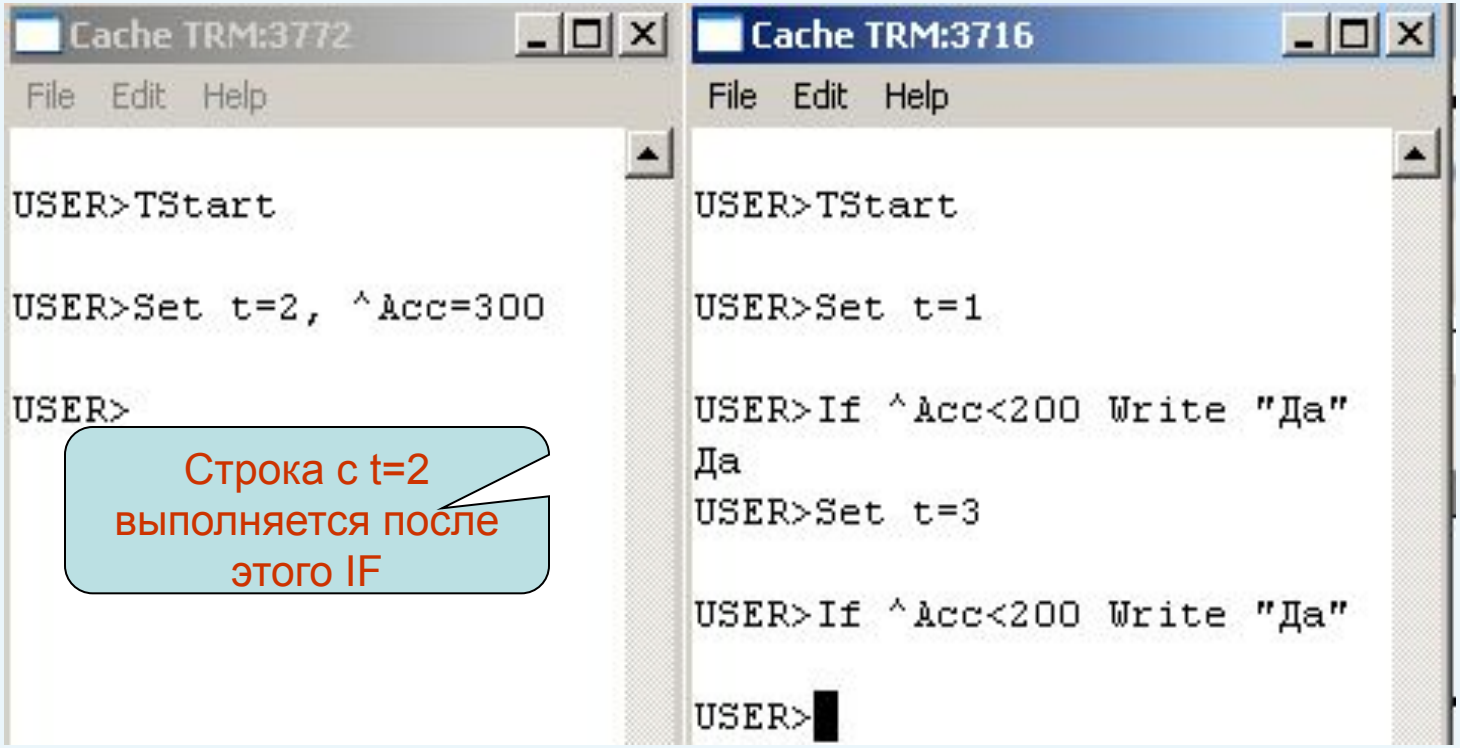
Неверные результаты могут быть получены в транзакциях, которые только считывают результаты, но не изменяют их, как в двух предыдущих феноменах. Для проявления эффекта необходимо, чтобы первая транзакция изменяла данные, используемые второй.

Транзакция 1	Время	Транзакция 2
	t_1	Чтение Acc (Acc=100) Проверка Acc<200 -- ДА
Изменение Acc=300	t_2	
	t_3	Чтение Acc (Acc=300) Проверка Acc<200 – НЕТ
----------		----------

← Начало Tp_2

← Начало Tp_1

“Неповторяющееся чтение” – демонстрация в Caché



Появление записей “фантомов”

Блокировка ресурса может выполняться инструкцией типа LOCK имя_ресурса.

Пусть за счёт блокировок на уровне строк перечисленные выше три феномена исключены. Транзакция Tp1 выбирает группу строк из таблицы T, удовлетворяющих условию Y. Затем транзакция Tp2 вставляет в T ещё одну строку, удовлетворяющую условию Y. При повторной выборке транзакцией Tp1 к набору добавляется ещё одна строка, называемая фантомом.

Как обеспечить правильную работу транзакций, обращающихся к одним и тем же ресурсам

Два основных способа:

- Блокирование “общих” ресурсов
- Предоставление транзакциям, конкурирующим за ресурсы, разных экземпляров данных

Блокирование

По степени разделяемости блокируемых ресурсов различают:

- Монопольные блокировки (e**X**clusive locks или X-locks), называемые ещё блокировками записи. Не разрешают другим транзакциям доступ к блокированному ресурсу.
- Разделяемые блокировки (**S**hared locks или S-locks) иначе блокировки чтения. Разрешают совместный доступ к блокированному ресурсу, но только по чтению.

Уровни изолированности пользователей (1/2)

Стандарт ANSI SQL-92, основываясь на перечисленных феноменах, определяет четыре уровня изолированности:

- **Read uncommitted**. Чтение незафиксированных изменений. Самый низкий уровень изоляции. Воспринимаются как окончательные, так и промежуточные результаты других транзакций.(В Oracle отсутствует.) Предотвращает только потерянные изменения. Но за счёт чего?
- **Read committed**. Чтение только зафиксированных изменений. (В Oracle устанавливается по умолчанию). Уровень изоляции выше чем у read uncommited. Промежуточные результаты других транзакций не доступны. Параллельно транзакции могут работать с разными данны-ми. Нет потерянных из… …я. Возможны фантомы и неповторяющееся…

Понятно это станет после изучения блокировок.

Реализация основывается либо на блокированиях данных, либо на **версионности**.

В Oracle имеется.

Уровни изолированности пользователей (2/2)

- **Repeatable read**. Повторяемое чтение. Транзакция не имеет доступа к промежуточным или окончательным результатам других транзакций. Уровень изоляции у неё выше, а степень параллелизма ниже, чем у транзакций уровня изоляции Read commited. Монопольные блокировки применяются ко всем данным, считываемым любой инструкцией транзакции, и сохраняются до её завершения. Другие транзакции не могут изменять строки, считанные незавершённой транзакцией. Однако, они могут вставлять новые строки, удовлетворяющие условиям фраз WHERE инструкций, содержащихся в текущей транзакции. При повторном запуске таких инструкций будут извлечены эти новые строки. Так появляются фантомные записи. (В Oracle этот уровень не нужен)
- **Serializable**. Сериализуемость. Самый высокий уровень изоляции. Результат параллельного выполнения транзакций будет точно таким же как при последовательном их выполнении. Все перечисленные выше феномены отсутствуют, но параллельное выполнение транзакций, работающих с одними ресурсами невозможно. (В Oracle имеется)

  В Oracle три уровня изоляции: Read Committed, Serializable, Read-Only. Но Read-Only не предусмотрен ANSI и представляет собой разновидность Serializable. Считаем, что два.

  Read uncommitted, Repeatable read в Оракле нет. По умолчанию Read Committed.

Уровни изолированности и отсутствие феноменов

Уровень изоляции	Потерянные изменения	“Грязное” чтение	Неповтор. чтение	Фантомы
Serializable	-	-	-	-
Repeatable read	-	-	-	+
Read commited	-	-	+	+
Read uncommited	-	+	+	+

Знак “+” означает, что феномен имеется, а “-”, что отсутствует.
Замечание: При задании типа транзакции может указываться ещё способ доступа (read only или read write), определяющий может ли транзакция изменять данные, и уровень изоляции. Уровень read uncommited совместим только со способом доступа read only.

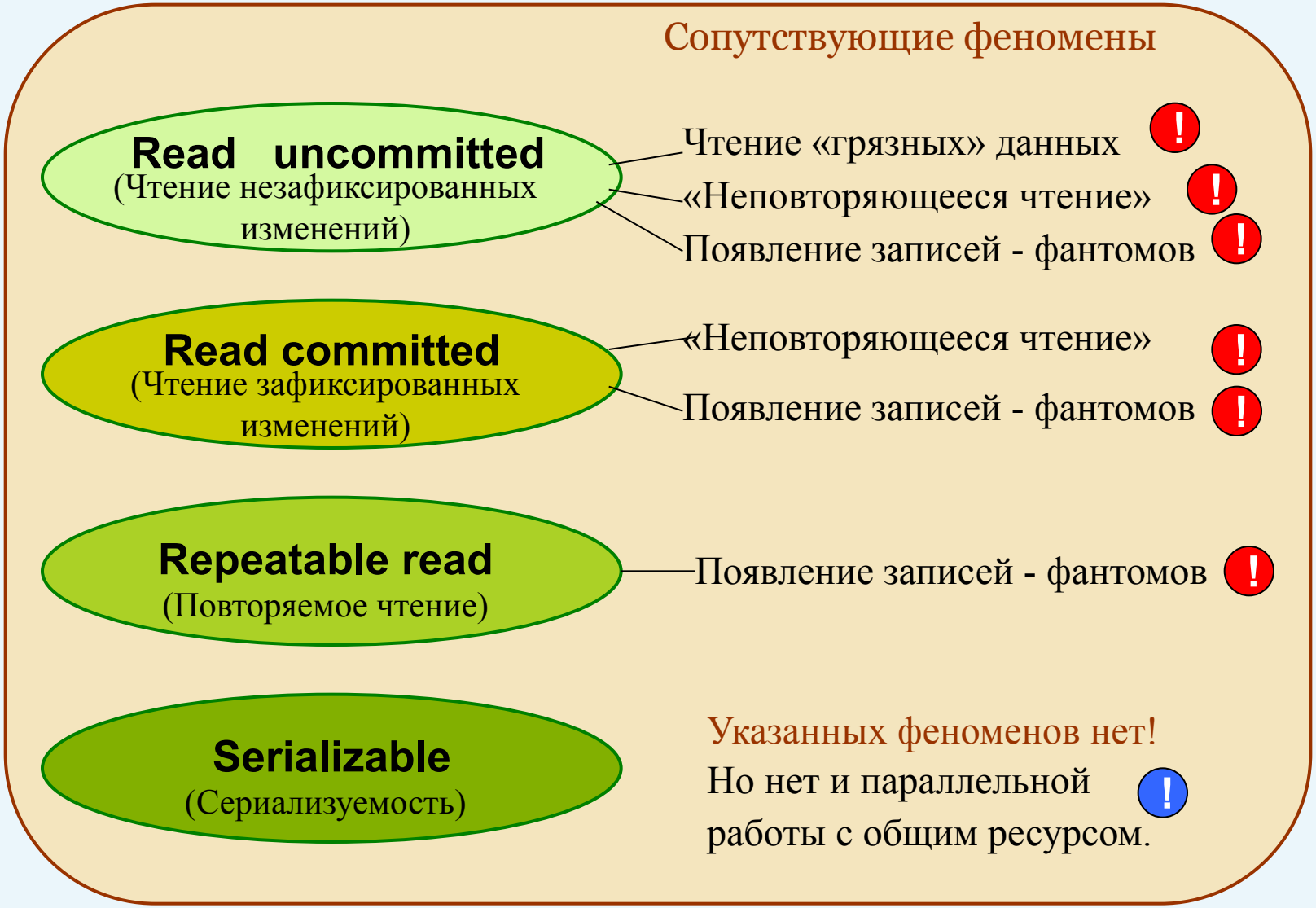
Так зачем нужны “плохие” виды транзакций, и особенно, Read uncommited?

Ответ простой. С одной стороны, чем выше уровень изоляции, тем меньше шансов на параллельную работу транзакций. С другой стороны, полная изоляция не всегда полезна.

Пример: Библиотекари регулярно работают с каталогом, открывая транзакции для сверки и изменения каталога. Эти работы могут длиться часами. Что Вы предпочитаете, ждать появления исправленного каталога, или просматривать его в любой момент времени, может быть получив неправильные данные?

Замечание: Правильнее было бы дать библиотекарям работать с копией каталога, а когда они закончат работу быстро обновить каталог, запретив пользователям доступ на несколько минут.

Уровни изолированности пользователей



Виды блокировок

Блокировки различают ещё **по размерам блокируемого ресурса** (поле записи, запись, отношение, страница (блок базы), группа отношений, вся база). В современных СУБД со строчной организацией блокировка полей не применяется. В СУБД со столбцовой организацией может использоваться.

Иерархичность. Если объект верхнего уровня обладает блокировкой, то такой же блокировкой обладает его объект нижнего уровня.

Время жизни: X-блокировки сохраняются до конца транзакции. S-блокировка снимается по завершении чтения.

Взаимодействие блокировок:

1) Имеется X-блокировка. Запросы на любые блокировки от других транзакций будут отменены.
2) Имеется S-блокировка. Запрос других транзакций на X-блокировку отвергается. Запрос на S-блокировку принимается.

Доступ по чтению и записи

Возможный протокол доступа к данным по чтению и записи с блокировками:

1. Перед чтением объекта базы транзакция должна наложить на него S-блокировку.
2. Перед записью объекта базы транзакция должна наложить на него X-блокировку. Если перед этим транзакция выполняла S-блокировку этого объекта, то она заменяется X-блокировкой.
3. Если объект уже заблокирован X-блокировкой другой транзакции, то любая блокировка отвергается, и транзакция переводится в состояние ожидания до тех пор, пока мешающая блокировка не снимется.
4. Если объект уже заблокирован S-блокировкой другой транзакции, то другая S-блокировка принимается, а X-блокировка отвергается.

Пример использования блокировок

Транзакция 1	Транзакция 2
Начало транзакции Lock (A) ;монопольная блокировка Read (A) A:=A+1 Write A Unlock (A) Конец транзакции	Начало транзакции Lock (A) ;попытка блокирования ; Работа с A невозможна ; Работа с A невозможна ; Теперь можно блокировать A Read (A) A:=A+10 Write A Unlock (A) Конец транзакции

Замечание: В примере считается, что команда Unlock снимает монопольную блокировку до конца транзакции

Совместимость блокировок

Если транзакция B начинается позже транзакции A, то успешность блокирования объекта базы транзакцией B определяется следующей **матрицей совместимости блокировок :**

Транзакция-читатель мешает транзакции-писателю

	Транзакция B пытается наложить блокировку	
Транзакция A наложила блокировку	**S-блокировку**	**X-блокировку**
S-блокировку	**Да**	**Нет (Конфликт R-W)**
X-блокировку	**Нет (Конфликт W-R)**	**Нет (Конфликт W-W)**

Отсюда следует важный вывод: Транзакции-читатели могут мешать транзакциям-писателям. Транзакции-писатели всегда мешают другим транзакциям-писателям и транзакциям-читателям.

А если убрать S-блокировки чтобы не мешали? Подумайте!

Проблема эскалации блокировок

Если, например, в одной таблице блокируются много строк, то некоторые СУБД переходят к блокированию всей таблицы. Это явление называемое **эскалацией блокировок**, может резко уменьшить возможности конкурентного доступа.

Блокировки в Caché ObjectScript (1/3)

Выполняет блокировку команда имеющая формат:
LOCK имя_блокируемого_ресурса
Пример: LOCK ^a(1) или, сокращённо, L ^a(1) .

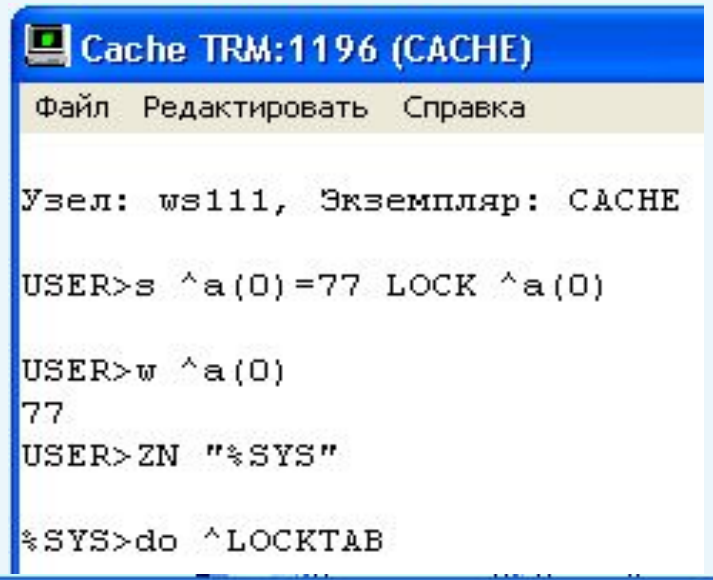
Формат	Действие
LOCK	Отмена всех блокировок
LOCK список_имен	Снимает все блокировки, затем блокирует по одному имени из списка
LOCK+имя	Инкрементальная блокировка – добавление блокировки к уже существующим блокировкам
LOCK-имя	Удаляет указанную блокировку

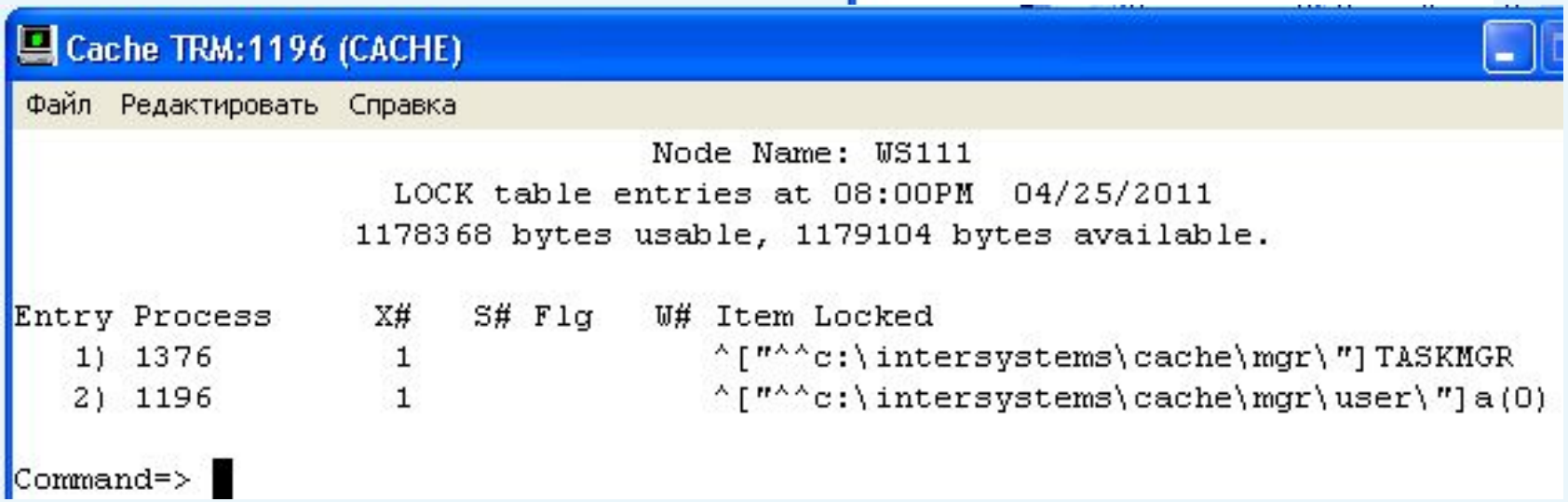
LOCK ^A(1), ^B(3) эквивалентно LOCK ^A(1) LOCK +^B(3)
Использование времени ожидания (для инкрементальных блокировок рекомендуется всегда):
LOCK ^A(1):5 Попытка блокирования прерывается, если прошло 5 секунд, устанавливается $TEST=0

Блокировки в Caché ObjectScript (2/3)

Таблица блокировок. Просмотреть текущие блокировки можно двумя способами:

1. С помощью панели управления (на слайде не показано)
2. Программой ^LOCKTAB, запускаемой из терминала





Блокировки в Caché ObjectScript (3/3)

Блокировки распространяются на всех потомков и предков.

Пример:

S ^A(1)=1,^A(1,1)=11,^A(2)=2,^A(2,1)=21

Команда L ^A(1) блокирует не только ^A(1), но и ^A,^A(1,1).

Замечание 1: Блокируются и локалы и глобалы.

Замечание 2: LOCK не проверяет существование узлов. Поэтому блокировка создаётся и для несуществующих узлов.

Рекомендация: Для удобства тестирования желательно, чтобы количество инкрементальных блокировок узла совпадало с количеством инкрементальных деблокировок.

Пример: Если выполнена команда L +(^A,^B,^C), а затем L ^B, то останется заблокированным только ^B.

Тупики (Clinch,Deadlock)

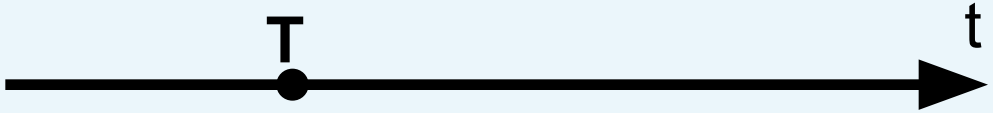
Обозначения операций транзакций:

$БР_i$ – блокирование ресурса i;

$РР_i$ – разблокирование ресурса i;

$Д_i$ – действие над ресурсом i.

Ситуация тупика:



Тр1: $БР_1$, $БР_2$ …….

Тр2: $БР_2$, $БР_1$ …….

С момента времени T продолжение работы транзакций невозможно, если СУБД не использует какой-нибудь алгоритм разрешения конфликта.

Решение проблем параллельного доступа при использовании блокировок

Пример ситуации чтения “грязных” данных:

Транзакция 1	Время	Транзакция 2
S-блокировка Acc успешна	t_1	
Чтение Acc	t_2	
	t_3	S-блокировка Acc успешна
Попытка записи	t_4	Чтение Acc
X-блокировка Acc отвергается	t_5	Попытка записи
Ожидание	t_6	X-блокировка Acc отвергается
Ожидание	t_7	Ожидание

X - блокировки отвергаются, так как уже установлены две S – блокировки. Обе транзакции ожидают завершения другой транзакции. Тупик, но феномен чтение “грязных” данных отсутствует. Если же S – блокировка транзакции 2 будет выполнена после успешной X – блокировки транзакции 1, то транзакция 2 будет ждать завершения транзакции 1.

Многоверсионные данные (1/2)

В СУБД с единственной версией данных транзакции-читатели могут мешать транзакциям – писателям (конфликты R-W).

В многоверсионных СУБД транзакциям-читателям предоставляются свои версии данных, получаемые откатом части схемы базы до последнего согласованного состояния. Такие транзакции не блокируют данных и потому не мешают транзакциям -писателям.

Многоверсионная СУБД Oracle создаёт и поддерживает системный номер изменения SCN (system change number). Каждая завершёная транзакция увеличивает его. Поэтому можно считать SCN уникальным идентификатором последней завершённой транзакции.

В заголовок блока данных записывается SCN завершившейся транзакции, которая изменяла блок последней.

При чтении результирующее множество запроса формируется следующим образом:

Многоверсионные данные(2/2)

1. Анализируется текущий SCN_i. Будем его называть SCN запроса.
2. При считывании блока данных Oracle сравнивает SCN запроса с SCN из заголовка этого блока чтобы определить -- читать ли сам блок или воспользоваться сегментом отката. Возможны два варианта:

- Если SCN блока меньше или равен SCN запроса, то последняя изменявшая блок транзакция завершилась до начала чтения. В этом случае читается сам блок;
- Если же SCN блока больше SCN запроса, то изменения блока завершатся после начала чтения. В этом случае из сегмента отката читается нужный блок, сохранённый в состоянии до начала чтения.

Замечание: Стратегия многовариантных данных применяется редко. Если же не использовать многовариантности и не применять блокировки, то можно получать несогласованные данные.

Изоляция транзакций в некоторых СУБД

Транзакционные СУБД не всегда поддерживают все четыре уровня изоляции и могут вводить дополнительные уровни. Возможны также различные нюансы в обеспечении изоляции.

- Oracle в принципе не поддерживает уровень Repeatable read , так как многоверсионность исключает грязные и неповторяющиеся чтения. По умолчанию для Oracle уровни изоляции -- Read committed (по умолчанию) и Serializable. Механика SCN гарантирует повторяемость чтения (если SELECT в одной транзакции выбирает из базы набор строк, и в это время вторая транзакция изменяет какие-то из этих строк, то результирующий набор, полученный первой транзакцией, “не заметит” этих изменений. Уже упоминалось, что в Oracle существуют READ-ONLY транзакции, которые не могут изменять данные, а по уровню изоляции соответствуют Serializable.
- Microsoft SQL Server поддерживает все четыре стандартных уровня изоляции транзакций, а дополнительно — уровень SNAPSHOT, находящийся между Repeatable read и Serialized. Транзакция этого уровня ведёт себя так, как будто получила при запуске моментальный снимок данных базы и работает только с ним.

О разделении времени

В однопроцессорной системе практически всегда применяют квазипараллельную обработку. В течение короткого промежутка времени процессор обрабатывает задание (программу) одного пользователя. Затем её выполнение прерывается, и небольшое время выделяется другому пользователю. После обработки задания последнего пользователя процессор вновь займётся первым пользователем и т. д. Такой способ обработки называется режимом разделения времени. В современных операционных системах выделяемые промежутки времени могут быть неодинаковые. Будем называть квазипараллельно выполняющиеся программы процессами.

Обратим внимание на то, что внутри процесса последовательность обработки информации не изменяется, но за счёт разделения времени все процессы замедляются.

Аналоги транзакций в языках общего назначения (1/2)

Необязательный раздел

Аналогами транзакций в языках программирования общего назначения можно считать процессы (processes) или потоки (threads).

Многозадачность, основанная на потоках предпочтительнее из-за того, что потоки разделяют одно адресное пространство, переключение контекста и взаимодействие между потоками дешевле, чем между процессами.

В Java потоки создаются уникальными объектами класса java.lang.Thread. Выделяют пользовательские потоки и потоки-демоны, управляемые средой исполнения.

Разделение доступа к ресурсам называется синхронизацией. В Java все объекты, включая массивы, имеют блокировку. Поток должен сначала получить блокировку объекта, связанного с разделяемым ресурсом, и только затем войти в разделяемый ресурс.

Используя термины из области баз данных блокировки в Java можно назвать монопольными, так как только один поток одновременно получает доступ к блокируемлму ресурсу.

Аналоги транзакций в языках общего назначения (2/2)

Необязательный раздел

По виду ресурса различают два варианта синхронизации :

- синхронизация методов;
- синхронизация блоков.

В определении синхронизируемых методов должно быть указано ключевое слово synchronized. Блокировка такого метода реализуется просто его вызовом.

Если же метод уже заблокирован, то вызывающий метод ждёт.

Например, в реализации стека операторы push и pop синхронизированы, что исключает их одновременное исполнение.

Синхронизация блоков кода может обеспечить более тонкое управление. Синтаксис записи синхронизированного блока:

synchronized (ссылка_на_объект) {блок-кода}.

Замечание: Обратите внимание на то, что в отличие от баз данных блокируется процедурная часть, а не данные.

III. Транзакции, откаты и восстановление после сбоев

Требование атомарности транзакций означает, что откатившиеся транзакции или транзакции, не успевшие завершиться, не должны оставлять никаких следов своей работы в базе данных.

Требование долговечности означает, что данные полученные зафиксированными транзакциями должны сохраниться в базе даже если в следующий за подтверждением момент произойдет сбой.

При современном уровне развития вычислительной техники единственный способ достичь приемлемой производительности это буферизация данных в оперативной памяти. Кажущийся естественным способ немедленной записи данных завершенных транзакций на диск неприемлем, так как требует работы с медленными файлами с адресным доступом.

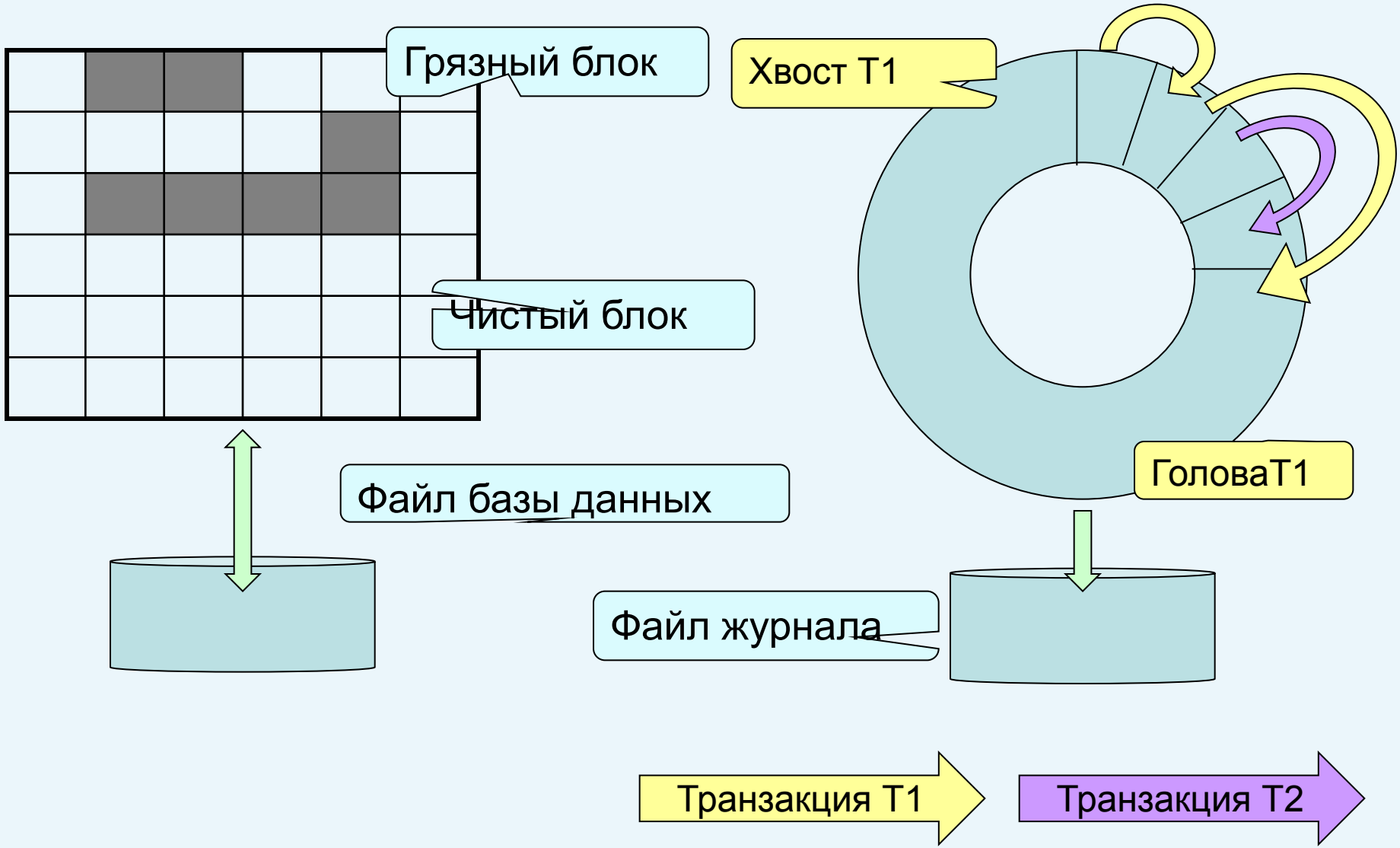
Выход из положения – использование дополнительных кольцевых последовательных буферов и файлов журналов имеющих последовательный доступ. Такие файлы быстрее адресных.

Образующаяся избыточность позволяет хранить в базе и измененные данные и их версии, существовавшие до внесения изменений.

Oracle -- пример архитектуры СУБД

Структуру запоминать не следует. Важны цепочки “Кэш буферов базы –Файлы данных” и “Журнальный буфер – Журнальные файлы”

Кэш буферов базы и буфер журнала



Восстановление данных

Основные ситуации, в которых требуется восстановление данных:

1. Откат транзакции по команде ROLLBACK.
2. Мягкий сбой системы (утрата части или всей первичной памяти).
3. Жесткий сбой системы (отказ аппаратуры, чаще повреждение вторичной памяти).

Далее будет рассмотрена упрощённая система организации откатов и восстановлений после сбоев, полученная на основе СУБД Oracle.

Блоки кэша буферов базы, содержимое которых отличается от содержимого соответствующих блоков на диске называются “грязными”. Эти блоки должны быть записаны (как говорят, вытолкнуты) во вторичную память.

Два противоречивых требования. С одной стороны, из-за недостатка оперативной памяти и из-за возможности сбоев необходимо выталкивать “грязные” блоки как можно раньше. Но это замедлит работу. Для ускорения работы необходимо записывать данные на диск как можно реже. Это увеличивает риск рассогласования базы.

Принцип “Write Ahead Log”

Измененные объекты базы данных должны попадать в файл журнала раньше, чем грязные блоки кэша буферов базы попадут в файлы данных. **Write Ahead Log!**

При этом может оказаться, что на диске имеется запись журнала, но ещё нет записи блока базы. Если уже записан блок данных, то журнальный блок тем более записан.

Периодически или при достижении кэшем буферов базы определенного состояния (например, количество страниц в dirty-списке превысило порог) возбуждается событие **контрольной точки**. Выполняются следующие два действия:

- во внешнюю память выталкивается часть “грязных” блоков;
- во внешнюю память записывается контрольная точка, то есть список всех выполняемых в этот момент транзакций.

Заметим, что в Oracle мы говорили бы скорее об инкрементной контрольной точке.

При выполнении одной транзакции восстановление последнего согласованного состояния базы гарантируется, если в файл журнала вытолкнуты все записи об изменении базы данных этой транзакцией, включая запись о конце транзакции.

Откат транзакции

Завершенные по COMMIT транзакции не могут быть откачены.

Выполнение отката незавершённой транзакции (завершение по ROLLBACK):

1. Идя по инвертированному списку блоков занятых транзакцией последовательно восстанавливают значения измененных блоков.
2. При успешном завершении отката в журнал заносится запись о завершении транзакции.

Восстановление после мягкого сбоя

Ситуация: принята последняя контрольная точка, через некоторое время произошел мягкий сбой.

Несколько возможных вариантов:

1. Транзакция успешно завершена до контрольной точки, все ее данные сохранены на диске. В восстановлении не нуждается.
2. Транзакция успешно завершена. Блоки журнала вытолкнуты полностью, а блоки буферов базы частично. Необходимо завершить операции, не отображенные в блоках базы.
3. Транзакция начата до последней контрольной точки и не завершилась до сбоя. Часть блоков данных и журнала переписана на диск по событию контрольной точки. Результатов остальных изменений нет. Откатить транзакцию.
4. Транзакция начата после последней контрольной точки и завершилась до сбоя. Записи журнала вытолкнуты на диск. Изменения в блоках базы на диске не производились. Необходимо повторить все действия (накатить транзакцию).
5. Транзакция начата после последней контрольной точки и не успела завершиться до сбоя. В журнале нет сведений о ней, изменения блоков базы были только в оперативной памяти. **Транзакция должна быть повторена!!**

Восстановление после жёсткого сбоя

Для создания и пополнения архивных файлов устанавливают режим архивирования (archivelog в Oracle). Если погибла часть жесткого диска или весь диск, то восстановление возможно только по данным архивных файлов и журнала транзакций. Желательно регулярно копировать файлы журналов на отдельные носители и хранить копии вне помещения сервера.

Порядок действий по восстановлению базы:

1. По архивным файлам восстановить базу данных. Желательно иметь возможность восстановить базу по состоянию на предыдущий день.
2. Если журнал сохранился, то по журналу повторяются все успешно завершившиеся транзакции последнего дня. (При этом не нужно откатывать транзакции прерванные в результате сбоя).
3. Если журнал погиб, восстановление последних транзакций невозможно и база создается по состоянию на момент последней архивации.

Замечание: Комплекс работ по сохранению и восстановлению данных называется backup&recovery.

Неклассические модели транзакций

Нестандартные модели транзакций используют для уменьшения затрат ресурсов сервера при откате (транзакции с точками останова), для более тонкого управления доступом к данными (вложенные транзакции), для решения проблем, возникающих при выполнении длительных транзакций (саги).

Пример: транзакция с точками останова (savepoints)

```
begin transaction;
…….
savepoint p;
update
      AccountInfo
set
      Accout = Account-sum
where
      Name = NameFrom;

if error then
begin
      rollback to savepoint p;
```

Вложенные транзакции

Транзакция может иметь одну и более вложенных транзакций. Вложенная транзакция представляется деревом транзакций.

У вмещающей транзакции имеется полный набор свойств ACID. У вложенных транзакций гарантирована только атомарность.

Свойства аварийного завершения составляющих транзакций:

- при откате родительской транзакции все её потомки откатываются;
- при аварийном завершении вложенной транзакции родительская транзакция решает вопрос о её откате, рестарте или других действиях; эти действия не влияют на решение о завершении родительской транзакции.

Во втором случае родительская транзакция может:

- проигнорировать откат подтранзакции, считая её **несущественной**;
- вызвать рестарт этой подтранзакции;
- инициировать альтернативную **зависимую** подтранзакцию;
- откатить родительскую транзакцию и все её подтранзакции.

Понятия несущественной и зависимой подтранзакции были введены именно для вложенных транзакций.

Протокол параллельного выполнения вложенных транзакций основан на стандартном двухфазном протоколе. При этом, если данные заблокированы транзакцией, то её потомки не имеют к ним

Саги

Эта модель базируется на понятии **компенсационных транзакци** и предназначена для работы с долгоживущими транзакциями.

Для заданной транзакции T **компенсационной** будет транзакция C, которая возвращает базу данных в состояние, существовавшее до начала транзакции T.

Сагой называется длительная транзакция, состоящая из цепочки подтранзакций T_1, T_2, … T_n , каждой из которых ставится в соответствие компенсационная транзакция C_i . Подтранзакции могут выполняться параллельно.

Будет выполнена либо вся последовательность T_1, T_2, … T_n, либо какая-нибудь последовательность T_1, … T_j , C_j, … C_1 . Иначе говоря, при аварийном завершении все выполненные подтранзакции будут скомпенсированы.

При завершении подтранзакции заблокированные ею данные освобождаются и становятся доступными другим параллельно выполняемым транзакциям, то есть нарушается изолированность.

При аварийном завершении саги делается попытка повтора, если же это невозможно, выполняется откат всей саги.

Заключение

В лекции обоснована необходимость введения механизма транзакций, который должен работать в базах **с любыми моделями данных**.

Возможны транзакции с ограничениями на свойства ACID.

Транзакции обеспечивают:

1. Сохранение целостности данных.
2. Параллельную работу с базой данных.
3. Восстановление данных при отказах и сбоях.

Изучены декларативные и процедурные ограничения целостности. Получено представление о триггерах.

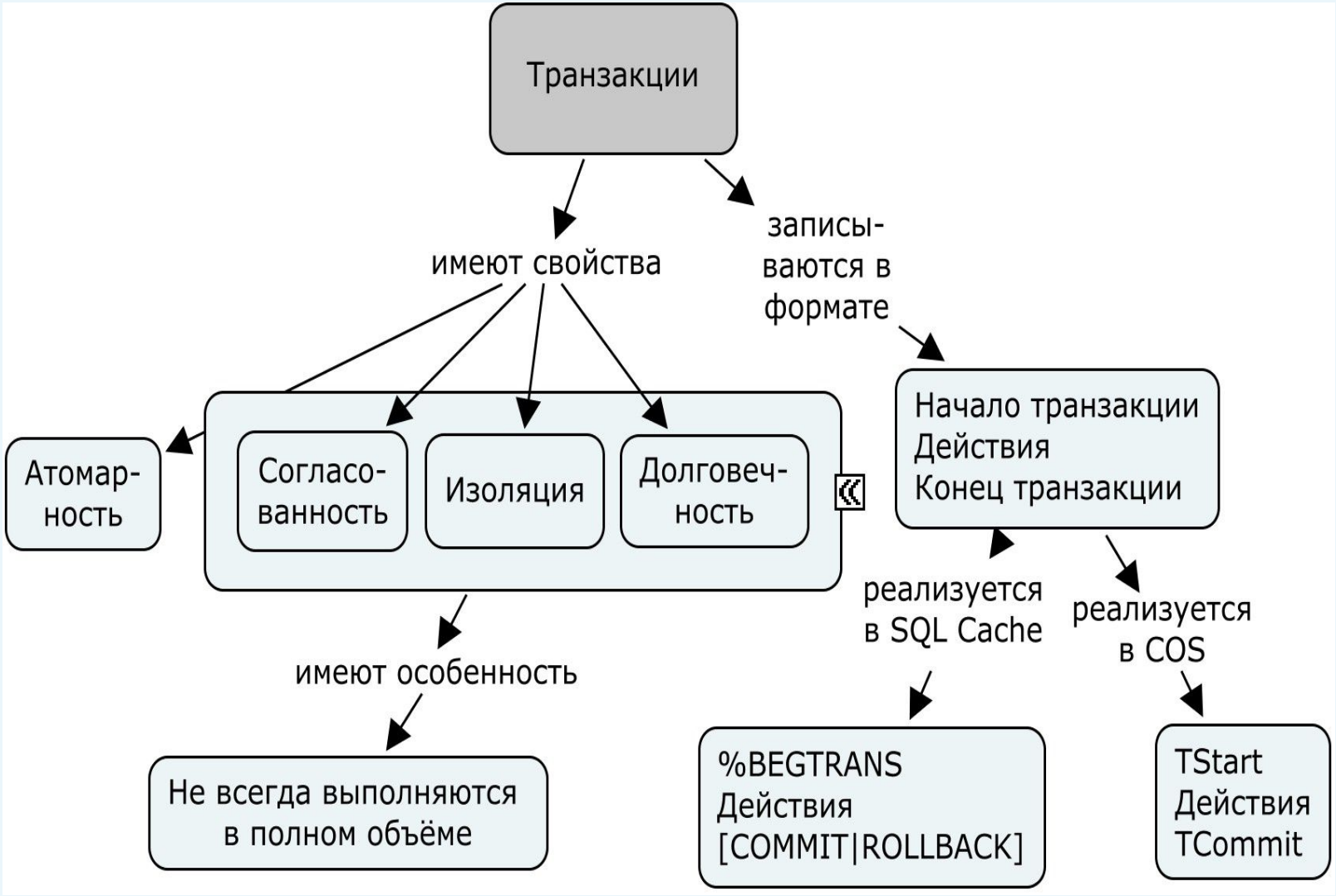
Рассмотрены феномены, определены уровни изоляции пользователей, блокировки и тупиковые ситуации.

Описаны элементы архитектуры СУБД, обеспечивающие буферирование данных и журналирование. Изучено восстановление данных при откатах транзакций, при мягких и жестких сбоях.

Приведены сведения о реализации транзакций в Caché.

Замечание: Не рассматривались транзакции в многозвенных системах, когда части транзакции кроме сервера баз данных выполняются на клиентах и на серверах промежуточного звена.

Основные понятия (1/2)



Основные понятия (2/2)