

# Введение в базы данных

7 января 2022 г.

## Содержание

<b>1</b>	<b>Теория</b>	<b>2</b>
1.1	Развитие баз данных . . . . .	3
1.1.1	Простые и структурированные файлы . . . . .	3
1.1.2	Файловая система . . . . .	3
1.1.3	Иерархическая модель данных . . . . .	4
1.1.4	Сетевая модель данных . . . . .	4
1.1.5	Реляционная модель данных . . . . .	5
1.1.6	Объектные базы данных . . . . .	5
1.1.7	NoSQL (Not only SQL) . . . . .	6
1.2	Реляционная алгебра. Предназначение и свойства . . . . .	7
1.3	Реляционная алгебра. Унарные и множественные операции . . . . .	9
1.3.1	Проекция . . . . .	9
1.3.2	Фильтрация . . . . .	10
1.3.3	Переименование . . . . .	12
1.3.4	Множественные операции . . . . .	12
1.4	Транзакции. Восстановление. Классический алгоритм . . . . .	15
1.4.1	Транзакции . . . . .	15
1.4.2	Восстановление . . . . .	16
1.4.3	Классический алгоритм восстановления . . . . .	18
<b>2</b>	<b>Практика</b>	<b>20</b>

# 1 Теория

## 1.1 Развитие баз данных

### 1.1.1 Простые и структурированные файлы

Простые файлы состоят из:

- Заголовок – название столбцов.
- Данные – значения, разделённые запятой.

В структурированных файлах в заголовках написано не только название столбца, но и его тип и длина.

**Замечание.** В структурированной версии можно быстро искать запись по номеру, то есть прочитать заголовок и узнать сколько занимает одна запись, умножить на нужный номер и прочесть сразу нужную запись.

Достоинства:

- Простота чтения – написать код который будет читать такие данные просто.
- Сложность поиска – не реализовать эффективный поиск которому не нужно было бы загружать всё в память.
- Сложность обработки.
- Сложно хранить нетривиальные типы данных например даты - теряется информация на какой позиции месяц, на какой день.
- Нет проверки целостности (ограничений).

### 1.1.2 Файловая система

Устройство:

- Файл – одна запись.
- Иерархия записей – иерархия каталогов.

Достоинства:

- Простота реализации.
- Структурированные данные.

Недостатки:

- Сложно извлекать требуемые данные.
- Нет проверки целостности.
- Большое количество файлов.

### 1.1.3 Иерархическая модель данных

**Замечание.** Иерархия это хорошо, но использовать для этого файловую систему не эффективно.

**Деревья** Отношение родитель – ребёнок соответствует каталогу и его подкаталогам в файловой системе, но не будет выделяться по файлу для каждой записи, вместо этого записи с одинаковым типом будут группироваться (благодаря этому не нужно будет лишней раз обходить файловую систему).

Достоинства:

- Проверка целостности появляется благодаря структурированности (а именно связи родитель - ребёнок), например можно проверять что у человека нет двух оценок по одному предмету (хотя в файловой системе тоже можно было это делать).
- Последовательное расположение записей - ускорение выполнения запросов.

Недостатки:

- Представление только древовидных структур.
- Нет отношения многие ко многим, например у множества студентов есть множество оценок по разным предметам и родителем будет студент, а детьми оценки или наоборот, запросы к обоим этим множествам выполняться эффективно не могут.

### 1.1.4 Сетевая модель данных

Обобщение иерархических баз данных, нет единой строгой иерархии, есть базовая иерархия и есть дополнительные иерархии вида владелец – запись

Достоинства:

- Представление всех типов связей (в том числе многие-ко-многим).
- Возможность описания структуры.
- Эффективность реализации – эффективные запросы к обоим мн-вам из связи многие ко многим, но эффективность разная из-за последовательной записи, только записи базовой иерархии записаны последовательно.

Недостатки:

- Более сложная реализация.
- Жесткое ограничение структуры – если мы не подумали о каком то виде запросов заранее, то возможно для его исполнения придётся поднять все данные.

### 1.1.5 Реляционная модель данных

**Хранение** Данные хранятся в таблицах, также в таблицах хранится информация о связях, связи задаются в запросах.

Достоинства:

- Представление всех типов связей-
- Гибкая структура данных – можно задавать произвольные запросы.
- Математическая модель – позволяет говорить что некоторые запросы эквивалентны, то есть запрос не обязан исполняться как написан, мб исполнен любой эквивалентный запрос, выбирается самый эффективный из эквивалентных и получается тот же самый результат что и при исходном запросе потому что запросы эквивалентны.

Недостатки:

- Сложность реализации.
- Сложность представления иерархических данных.
- Сложность составления эффективных запросов.

### 1.1.6 Объектные базы данных

Цель – хранить граф объектов, который уже находится в памяти, в базе данных. Обычная реализация – слой трансляции в реляционную базу данных

Достоинства:

- Работа в терминах объектов а не записей.
- Логичное направление ссылок, например можем легко взять все оценки студента потому что есть соответствующее отображение из студента в оценки.

Недостатки:

- Сложность реализации.
- Сложность миграции схемы, например добавление поля объекту, в базе уже есть объекты без этого поля.
- Малая распространенность.

### 1.1.7 NoSQL (Not only SQL)

**Основная мысль** Реляционные базы данных умеют слишком много – они заточены чтобы работать одинаково эффективно в куче различных сценариев, а если у нас какой то один сценарий, то можно оптимизировать ровно для него и написать эффективней.

Различные типы:

- Документ-ориентированне – есть куча документов, важно что внутри них, главное уметь их быстро искать.
- Ключ-значение – всё что предоставляет движок - быстро по ключу достать значение.
- Табличные и столбчатые – хранить таблицы по столбцам, так если у нас множество запросов к конкретным двум столбцам, то мы сможем прочитать только их, читать придётся дважды (каждый столбец отдельно читается), но зато не нужно читать все столбцы как в табличном подходе.
- Графовые – хотим хранить графы.

Достоинства:

- Большой выбор – отказываемся почти от всего кроме одного, у чего получаем большую производительность.
- Гибкость – в момент разработки базы, не тогда когда уже есть база.
- Скорость работы.

Недостатки:

- Множество вещей делается в коде.
- Нет стандартных оптимизаторов.
- Легко ошибиться.

## 1.2 Реляционная алгебра. Предназначение и свойства

Базы данных нужно уметь не только проектировать, но и использовать. Существует несколько способов формулировать запросы. Первый из рассматриваемых – *реляционная алгебра*.

**Мотивация** Действительно, в базах данных можно не только хранить данные, но и делать выборки, изменять их каким-либо образом. Для этого вводится понятие запроса. При первом рассмотрении, запросы нужны как минимум для выполнения следующих действий:

- Выборка данных: получить данные из базы, чтобы тем или иным способом обрабатывать их уже извне.
- Область действия обновлений: запросы позволят указывать область действия тех или иных операций, что крайне полезно. Например, к таким операциям относятся операции удаления или изменения данных: хочется указывать, на какие именно записи эти операции подействуют.
- Ограничения целостности: до сих пор было только два вида ограничений (ключи и внешние ключи). Некоторые базы данных позволяют создавать произвольные ограничения целостности, заданные на поддерживаемом языке. В рамках этих ограничений очень удобно пользоваться запросами.
- Ограничения доступа.

**Определение.** *Реляционная алгебра* – алгебра над множеством всех отношений.

Далее будут определены некоторые из операций (которые по определению должны быть замкнуты над носителем), и ограничения, которые им соответствуют. В целом, реляционная алгебра – императивный язык для работы с отношениями, который позволяет в явном виде, по действиям, описать, каким именно образом должен быть получен результат.

**Примеры** Рассмотрим несколько простых примеров операций в рамках реляционной алгебры.

- Проекция отношения на множество атрибутов:  $\pi_A(R)$ ;
- Естественное соединение  $R_1 \bowtie R_2$ .

**Замечание.** Как уже говорилось, все операции в рамках алгебры замкнуты по определению. Это означает, что их можно комбинировать произвольным образом (при сохранении условий на возможность исполнения операции). Например:  $\pi_A(R_1 \bowtie \pi_B(R_2)) \bowtie R_3$ .

**Операции** В текущем контексте полезно уточнить, что именно понимается под операцией над отношениями в рамках реляционной алгебры. А именно, для того, чтобы определить операцию, необходимо определить следующее:

- Правило построения заголовка по заданным отношениям;
- Правило построения тела по заданным отношениям;
- Условия, при которых операция выполнима, то есть ограничения на отношения, к которым она применяется.



### 1.3 Реляционная алгебра. Унарные и множественные операции

В этом разделе будут описаны унарные операции в рамках реляционной алгебры. В соответствии с определением, для определения каждой операции нужно указать способ построения заголовка, тела отношения, а также условия применимости, если такие есть.

#### 1.3.1 Проекция

**Определение.** *Проекцией* отношения  $R$  на множество атрибутов  $A = \{a_1, a_2, \dots, a_n\}$  называется отношение, полученное из исходного путем удаления атрибутов не из  $A$ . Обозначается  $\pi_A(R)$ .

Данная операция может быть полезна для следующего:

- Привести отношение к виду, в котором над ним можно будет осуществить другую операцию (например, объединение);
- Выбрать из отношения только нужные данные (для выборки).

На рисунке 1 приведена иллюстрация к определению  $\pi_{A_2, A_4, A_5}(A)$ .

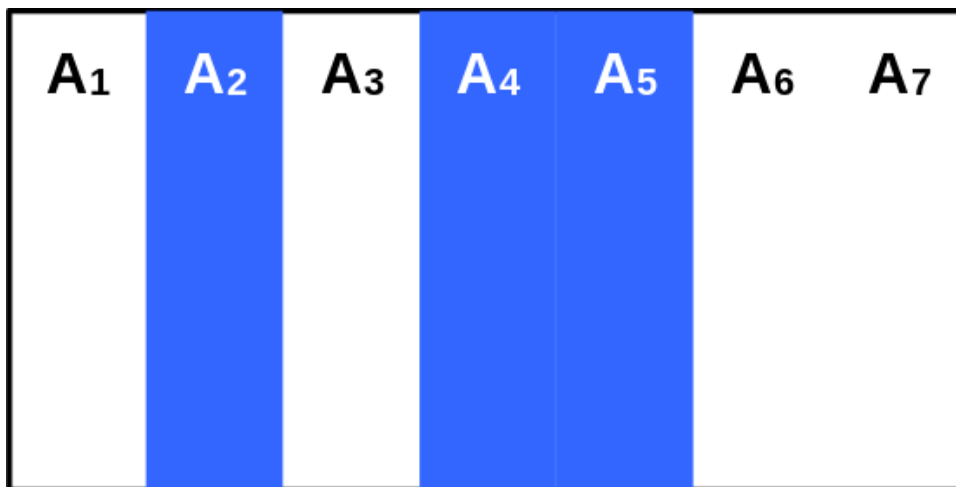


Рис. 1: Иллюстрация к определению проекции

Синим здесь обозначены столбцы, которые есть в результате операции. Остальные столбцы не используются, и результат никак не зависит от их содержимого.

**Примеры** Приведем несколько тривиальных примеров применения проекции.

- $\pi_{FirstName, LastName}$

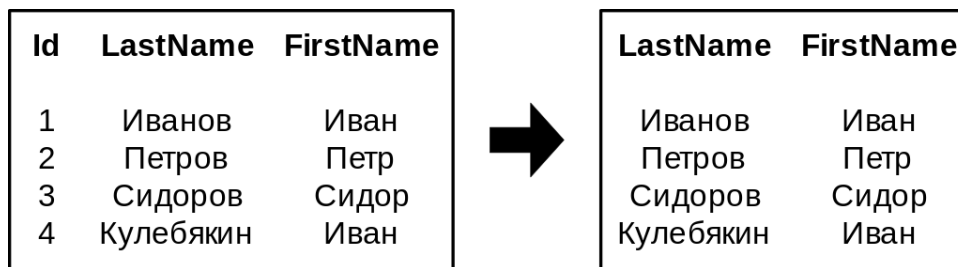


Рис. 2: Проекция. Пример 1

- $\pi_{FirstName}$

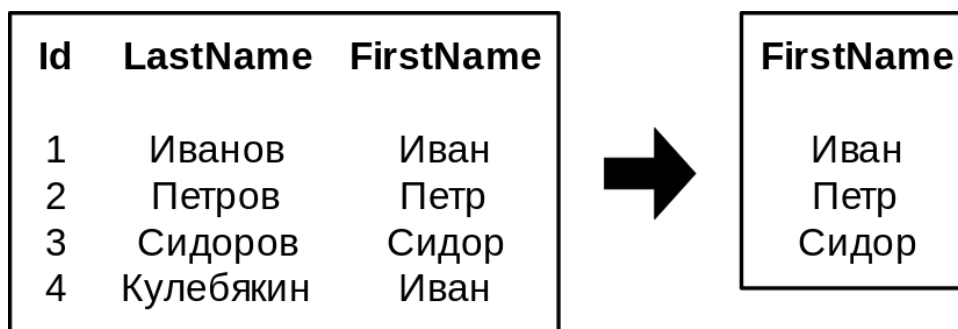


Рис. 3: Проекция. Пример 2

### 1.3.2 Фильтрация

**Определение.** *Фильтрацией (селекцией, выборкой из)* отношения  $R$  называется отношение, чей заголовок полностью совпадает с заголовком  $R$ , но тело содержит только кортежи, удовлетворяющее условию  $s$ . Обозначение:  $\sigma_c(R)$ .

Операция часто используется для

- Ограничения области действия изменяющих запросов;
- Получения выборки данных, соответствующих определенному условию.

На рисунке 4 приведена иллюстрация к определению  $\sigma_c(R)$ .



Рис. 4: Иллюстрация к определению фильтрации

Синим здесь обозначены столбцы, которые есть в результате операции. Остальные столбцы не используются, и результат никак не зависит от их содержимого.

**Примеры** Приведем несколько тривиальных примеров применения фильтрации.

- $\sigma_{Id>2}$

Id	LastName	FirstName
1	Иванов	Иван
2	Петров	Петр
3	Сидоров	Сидор
4	Кулебякин	Иван

➔

Id	LastName	FirstName
3	Сидоров	Сидор
4	Кулебякин	Иван

Рис. 5: Фильтрация. Пример 1

- Можно писать более сложные условия.  $\sigma_{Id>2 \wedge FirstName=Иван}$

Id	LastName	FirstName
1	Иванов	Иван
2	Петров	Петр
3	Сидоров	Сидор
4	Кулебякин	Иван

➔


Id	LastName	FirstName
4	Кулебякин	Иван

Рис. 6: Фильтрация. Пример 2

- Можно использовать функции, доступные в используемой БД.

$\sigma_{\text{length}(FirstName)+2 \geq \text{length}(LastName)}$

Id	LastName	FirstName
1	Иванов	Иван
2	Петров	Петр
3	Сидоров	Сидор
4	Кулебякин	Иван



Id	LastName	FirstName
1	Иванов	Иван
2	Петров	Петр
3	Сидоров	Сидор

Рис. 7: Фильтрация. Пример 3

### 1.3.3 Переименование


**Определение.** *Переименованием* называется операция, при которой меняются названия атрибутов отношения. Тело при этом остается неизменным.

Операция часто применяется для того, чтобы отношение можно было использовать в рамках другой операции (например, при объединении с другим отношением).

**Примеры** Ниже приведен тривиальный пример-пояснение для операции переименования.

- $\rho_{Name=FirstName, Surname=LastName}$

Id	LastName	FirstName
1	Иванов	Иван
2	Петров	Петр
3	Сидоров	Сидор
4	Кулебякин	Иван



Id	Surname	Name
1	Иванов	Иван
2	Петров	Петр
3	Сидоров	Сидор
4	Кулебякин	Иван

Рис. 8: Переименование. Пример

### 1.3.4 Множественные операции

Из теории множеств в реляционную алгебру естественным образом переходят операции:

- $R_1 \cup R_2$  – объединение.
- $R_1 \cap R_2$  – пересечение.
- $R_1 \setminus R_2$  – разность.

Эти операции по определению применимы только к отношениям с одинаковыми заголовками. В результате получается отношение с таким же заголовком и телом, полученным в соответствии с множественной операцией. Иначе говоря, заголовок остается тем же, а над телами отношений производится соответствующая множественная операция (объединение, пересечение, вычитание и прочие).

## Примеры

- Объединение отношений:  $R_1 \cup R_2$

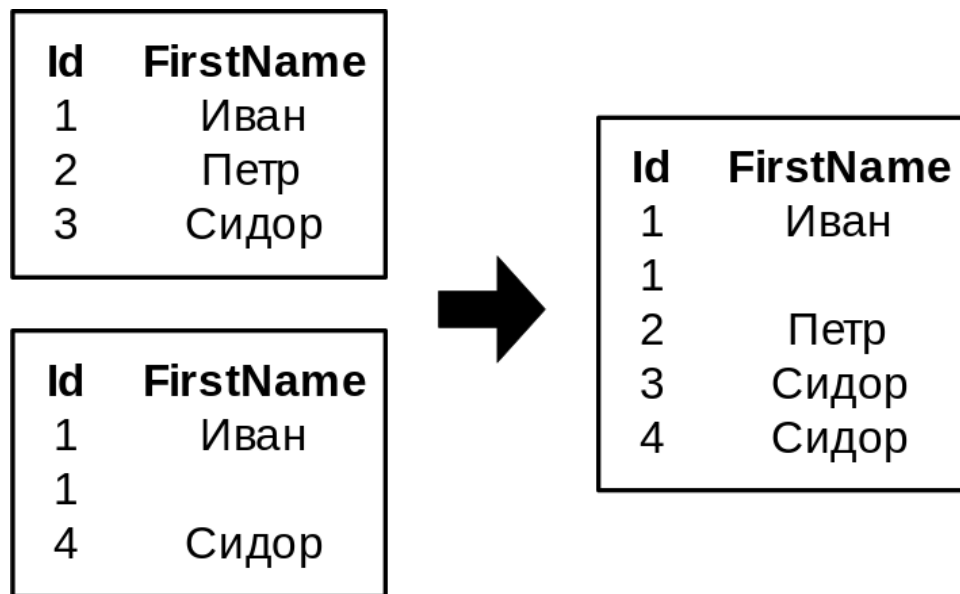


Рис. 9: Объединение отношений

- Пересечение отношений:  $R_1 \cap R_2$

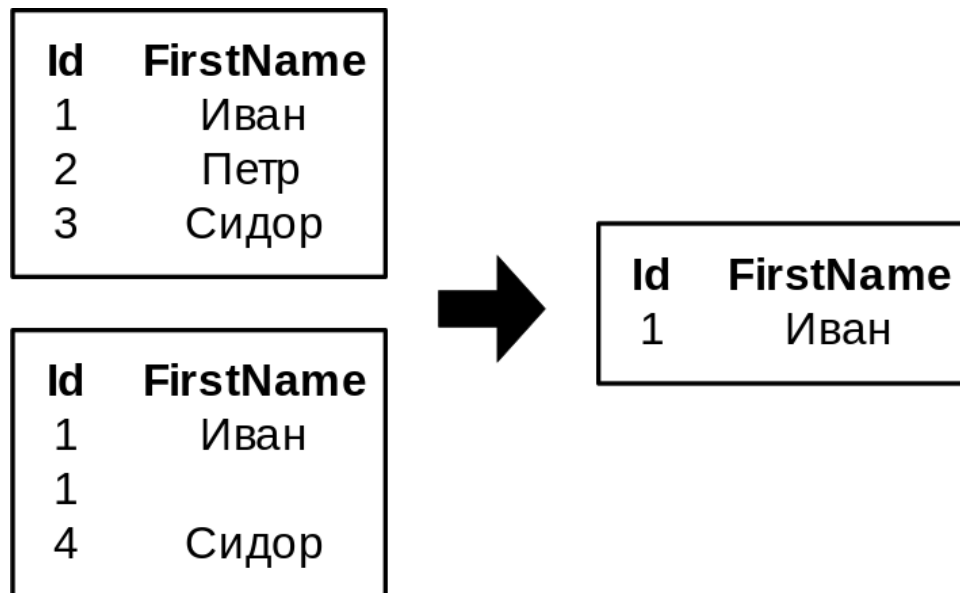


Рис. 10: Пересечение отношений

- Разность отношений:  $R_1 \setminus R_2$

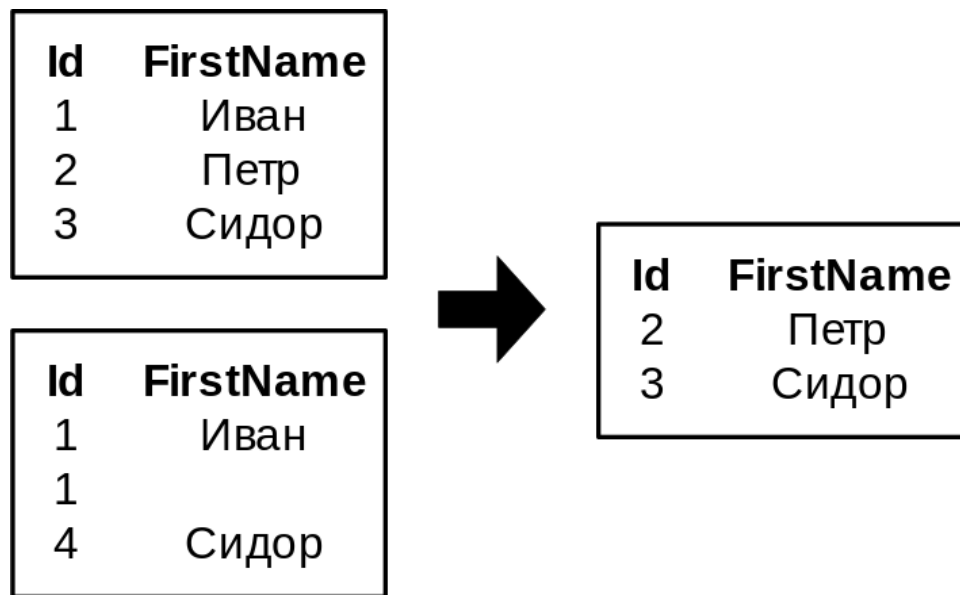


Рис. 11: Разность отношений

Стоит отметить, что для объединения отношений с различающимися именами атрибутов, но при равном их количестве, можно воспользоваться переименованием для того, чтобы привести заголовки к одному виду.

## 1.4 Транзакции. Восстановление. Классический алгоритм

### 1.4.1 Транзакции

**Определение.** *Транзакция* – минимальный объем работы, который можно зафиксировать в базе данных.

Каждый оператор заключен в неявную транзакцию, которая начинается непосредственно перед оператором и заканчивается после него. Не все действия в СУБД являются транзакционными. Например, во многих реализациях не поддерживается транзакционное изменение схемы данных.

#### Свойства транзакций (ACID)

**Определение.** *Атомарность (Atomicity)* – с точки зрения БД, транзакция либо выполняется целиком, либо полностью откатывается. Иначе говоря, никто со стороны не может увидеть промежуточное состояние выполнения транзакции.

**Определение.** *Согласованность (Consistency)* – после завершения транзакции БД остается в согласованном состоянии.

**Определение.** *Изоляция (Isolation)* – транзакции не могут взаимодействовать между собой. Это означает, что транзакции не могут пользоваться промежуточными результатами друг друга.

**Определение.** *Устойчивость (Durability)* – при успешном завершении транзакции результаты ее исполнения сохраняются в БД, при откате транзакции все внесенные ею изменения отменяются.

#### Корректность и согласованность

**Определение.** Состояние БД является *согласованным*, если оно удовлетворяет всем объявленным ограничениям. Это свойство автоматически проверяется СУБД.

**Определение.** Состояние БД является *корректным*, если оно соответствует реальному миру. Автоматически проверено быть не может.

Существуют условия корректности, которые нельзя проверить ограничениями. Например, после перевода денег в банке их общая сумма в системе не должна измениться. Однако, эта сумма заранее неизвестна, поэтому заранее задать ограничение невозможно.

**Минимизация транзакций** Транзакции требуют большие ресурсные затраты, поэтому должны быть минимальными. По возможности следует использовать неявные транзакции.

Однако, есть типичные ситуации, в которых использования неявных транзакций недостаточно:

- *Условное обновление* – проверку условия и обновление необходимо сделать в рамках одной транзакции, в противном случае в момент изменения условие может перестать выполняться;
- *Множественное обновление* – при обновлении данных, особенно в различных таблицах, использование транзакции необходимо для исключения несогласованности;
- *Промежуточная несовместимость* – некоторые действия требуют временного нарушения согласованности с последующим его восстановлением, использование транзакций позволяет откладывать проверку согласованности до завершения всех действий и исключают видимость несогласованного состояния другими пользователями.

Также следует отметить, что результат завершения транзакции **не должен зависеть от человека**. Человеческий фактор может привести к зависшей транзакции. Если требуется принятие решения от человека, транзакцию следует разбить на две: первая читает данные, а вторая их записывает, предварительно проверяя данные на соответствие результату первой. В таком случае от решения человека зависит применение второй транзакции.

#### 1.4.2 Восстановление

Напомним, что свойство *устойчивости (durability)* транзакции подразумевает сохранение результатов транзакции в БД даже при сбоях.

Хранение данных в оперативной памяти может приводить к потерям, например, при перезагрузке. Это считается нормальным, что приводит к необходимости хранения информации на дисках. Напомним, что с диска быстрее читать данные последовательно.

#### Типы сбоев

- **Локальный.** Сбой одной транзакции. Для восстановления достаточно откатить затронутую транзакцию.
- **Глобальный.** Сбой процесса СУБД, затрагивает все транзакции. Для восстановления достаточно откатить все незавершенные транзакции, а также заново применить все успешно завершённые транзакции.
- **Аппаратный.** Например, перезагрузка компьютера. С точки зрения СУБД, не существенно отличается от глобального сбоя.
- **Отказ оборудования.** СУБД не может восстановиться после этого типа сбоя. Однако, многие СУБД предоставляют *средства* для восстановления (например, запись данных на несколько дисков и синхронизация копий).

Свойство устойчивости не является абсолютным. Существуют сбои, при которых его нельзя поддержать.



**Восстановление после сбоя** Для восстановления БД достаточно сделать следующее:

- Успешные транзакции – зафиксировать;
- Откаченные транзакции – откатить;
- Незавершенные транзакции – откатить.

Существует несколько популярных подходов для отката:

**Shadow copy** Каждая транзакция пишет данные в новое место. При успешном завершении транзакции копия помечается успешной, пользователь уведомляется об успешной транзакции, и начинается запись из копии в БД. При сбое во время записи производится повторная запись. Проблема подхода заключается в частых чтениях и записях shadow copy, которые расположены в случайных местах на диске, что медленно.

**Transaction log** Данные пишутся сразу в БД, параллельно записывая изменения, примененные в рамках каждой транзакции, в журнал. Данный подход более популярен.

Журнал записывается в надежное хранилище изменений. Это означает, что его утрата есть невозстановимый сбой. Однако, записи ведутся последовательно, что делает данный подход быстрее предыдущего.

В журнал записываются: старые данные, новые данные, маркеры начала и завершения транзакции.

При завершении транзакции все изменения записываются в журнал, записывается маркер завершения транзакции, пользователь уведомляется о завершении транзакции.

### Реализация журнала

- **Постоянная запись на диск.** При записи каждого изменения в журнал существенно возрастает число операций, конкуренция за доступ к диску, а также накапливаются откаченные транзакции, которые в будущем не принесут пользы.
- **Запись при завершении.** В журнал при завершении транзакции записываются порожденные изменения. При больших изменениях это приводит к росту потребления памяти журналом.
- **Точки восстановления.** В журнал периодически записывается "слепок" состояния системы: текущие изменения, завершенные транзакции (не записанные ранее), откаченные транзакции (не записанные ранее), открытые транзакции. Создание точки восстановления требует приостановки изменений.

**Структура журнала** С использованием механизма точек восстановления, получаем следующую структуру журнала.

- **Точка восстановления;**
- **События:**
  - Идентификатор транзакции,
  - Указатель на *предыдущее* событие транзакции:
    - \* Начало транзакции,
    - \* Изменение,
    - \* Завершение транзакции,
    - \* Откат транзакции.

**Примеры** Рассмотрим пример сбоя и определим, что должно произойти с каждой из транзакций.

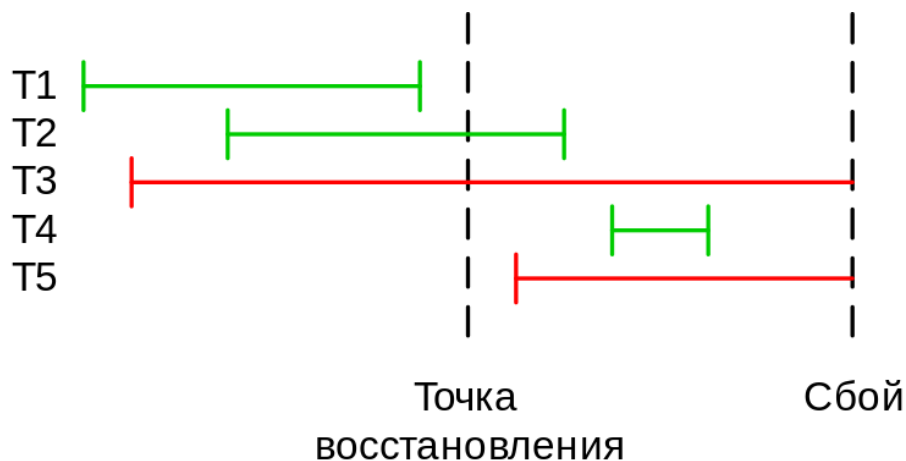


Рис. 12: Иллюстрация к определению проекции

Зеленым и красным цветом отмечены транзакции, которые должны быть завершены или отменены соответственно при восстановлении. Отметим, что эти решения однозначно определены гарантиями ACID транзакций.

### 1.4.3 Классический алгоритм восстановления

#### Фазы алгоритма

- **Разметка транзакций.** Каждая транзакция отмечается как *Redo* или *Undo*.
- **Откат транзакций.** Для помеченных как *Undo*.
- **Повтор транзакций.** Для помеченных как *Redo*.

### **Фаза разметки транзакций**

- Чтение журнала идет от последней точки восстановления до конца. Все открытые транзакции помещаются в *Undo*.
- При чтении маркера начала транзакция добавляется в *Undo*.
- При чтении маркера конца транзакция переносится из *Undo* в *Redo*.

### **Фаза повторения транзакций**

- Чтение журнала идет от последней точки восстановления до конца.
- При чтении маркера конца транзакция удаляется из *Redo*.
- При чтении изменения оно применяется, если транзакция в *Redo*.

**Утверждение 1.1.** После успешного выполнения всех фаз БД находится в корректном состоянии и гарантирует выполнения свойства устойчивости.

**Утверждение 1.2.** Рассмотрим все открытые транзакции. Для каждой из них найдем ближайшую точку восстановления из будущего. Все данные до самой ранней точки восстановления из рассматриваемых можно удалить, поскольку они не понадобятся при восстановлении.

## 2 Практика