ТРАНЗАКЦИИ

Что такое транзакция?

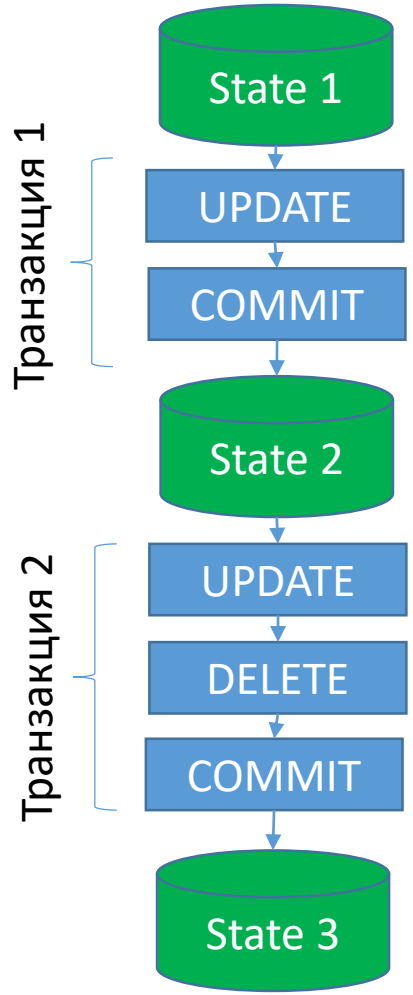
- **Транзакция –** это один или несколько операторов SQL, выполняемых как единая логическая единица работы (задача)
  - Промежуточные состояния данных, изменяемых операторами внутри транзакции, не видны другим транзакциям

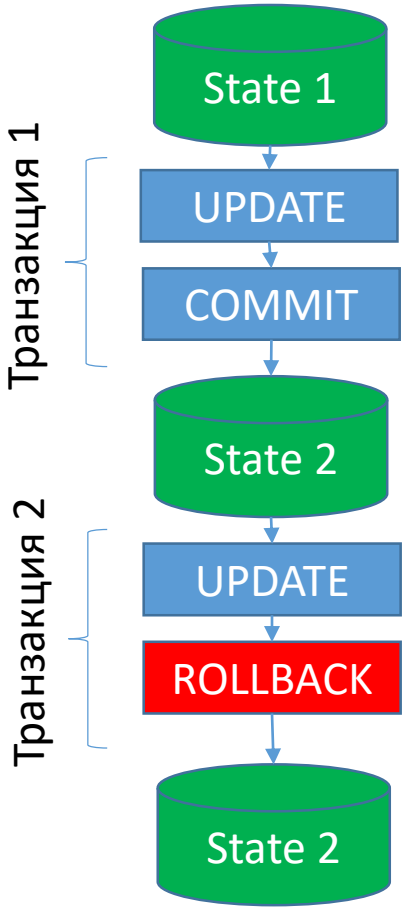
- Транзакция должна гарантировать соблюдение целостности данных
  - Если какой-либо оператор внутри транзакции не может быть успешно завершен, изменения других операторов не должны сохранится в базе данных

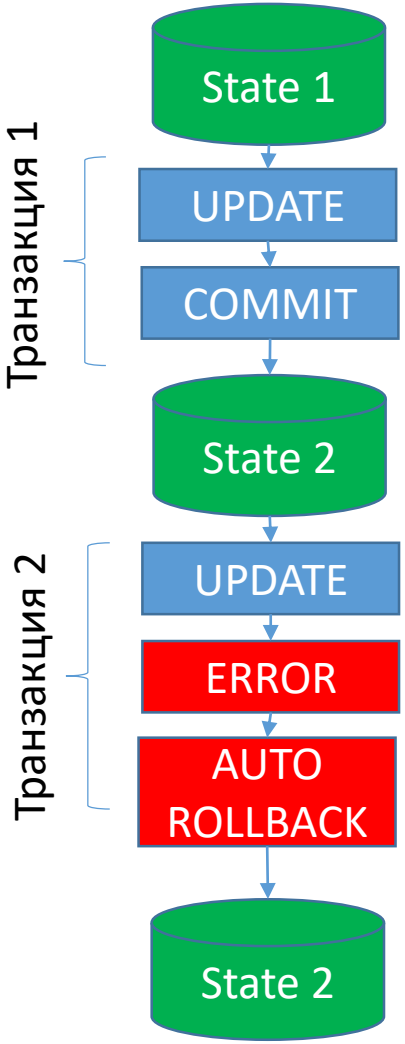
Соответствие требованиям ACID

- Для соблюдения целостности данных транзакции должны соответствовать требованиям **ACID**:
  - **Атомарность** (**A**tomicity)
    - это условие, при котором либо транзакция успешно выполняется целиком, либо, если какая-либо из ее частей не выполняется, вся транзакция отменяется
  - **Согласованность** (**C**onsistency)
    - это условие, при котором данные, записываемые в базу данных в рамках транзакции, должны соответствовать всем правилам и ограничениям, включая ограничения целостности
  - **Изоляция** (**I**solation)
    - необходима для контроля над согласованностью и гарантирует базовую независимость каждой транзакции
  - **Надежность** (**D**urability)
    - подразумевает, что все внесенные в базу данных изменения на момент успешного завершения транзакции считаются постоянными

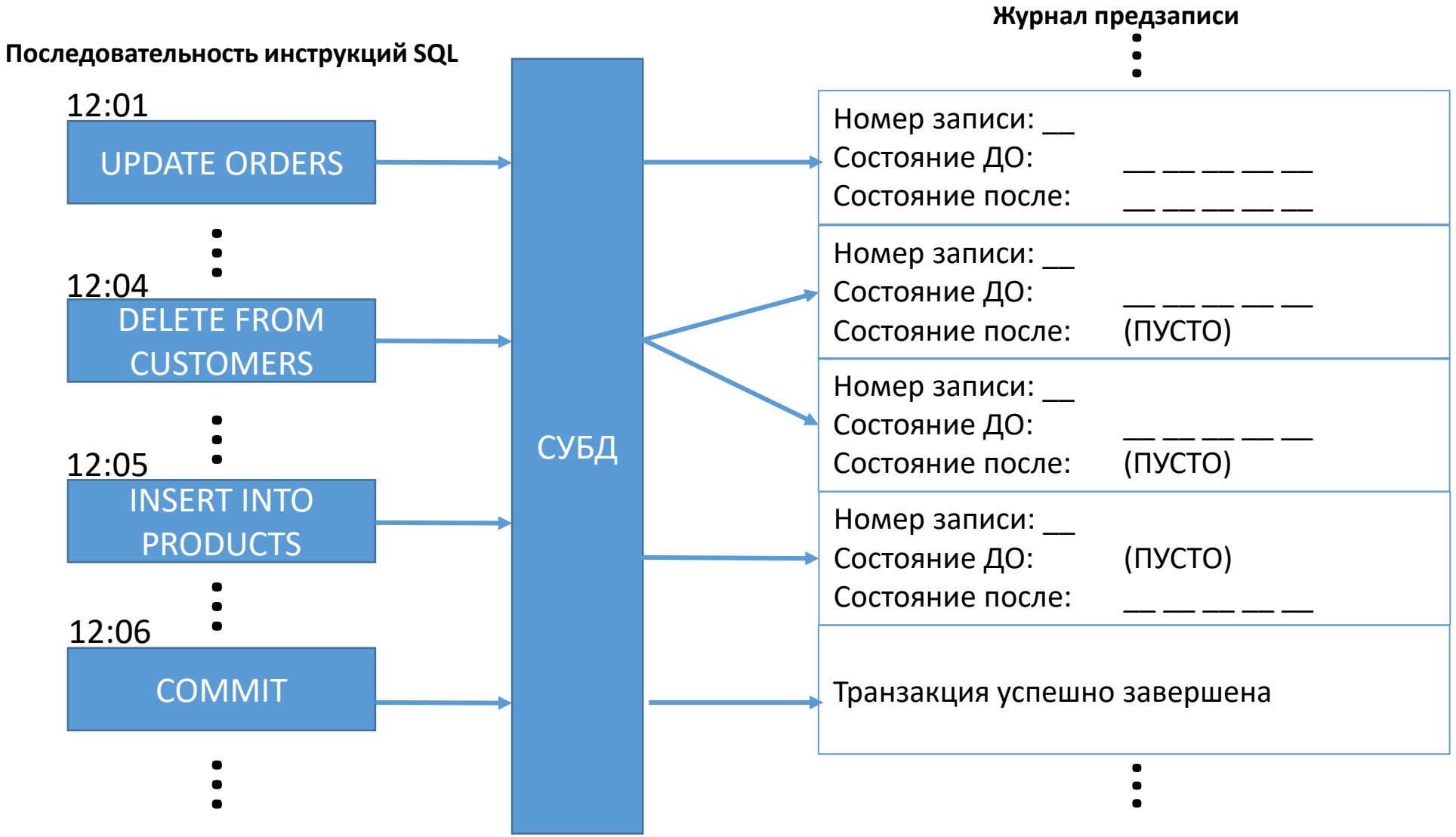
Модель транзакции







Журнал предзаписи



Типы транзакций

- AUTOCOMMIT
- Явные транзакции

Режим транзакций: AUTOCOMMIT

```
CREATE TABLE
"TestBatch" (cola INT PRIMARY KEY, colb CHAR(3)); --1 транзакция

INSERT INTO "TestBatch" VALUES (1, 'aaa'); --2 транзакция
INSERT INTO "TestBatch" VALUES (2, 'bbb'); --3 транзакция
INSERT INTO "TestBatch" VALUSE (3, 'ccc'); --4 транзакция
                                    --Синтаксическая ошибка

SELECT *
FROM "TestBatch" ; --5 транзакция
```

	cola	colb
1	1	aaa
2	2	bbb

Явные транзакции

- BEGIN
- COMMIT
- ROLLBACK
- SAVEPOINT

Откат транзакции

	ClientID	Lname	Fname	City	RegDate
1	6	Sergeev	Sven	London	2022-05-12

	ClientID	Lname	Fname	City	RegDate
1	6	Sergeev	Anton	London	2022-05-12

	ClientID	Lname	Fname	City	RegDate
1	6	Sergeev	Sven	London	2022-05-12

```
begin;
    SELECT "ClientID","Lname","Fname","City","RegDate"
    FROM public.client
    WHERE "ClientID"=6;

    UPDATE public.client
    SET "Fname" = 'Anton'
    WHERE "ClientID"=6;

    SELECT "ClientID","Lname","Fname","City","RegDate"
    FROM public.client
    WHERE "ClientID"=6;
rollback;

SELECT "ClientID","Lname","Fname","City","RegDate"
FROM public.client
WHERE "ClientID"=6;
```

Фиксация транзакции

	orderid	productid	unitprice	qty	discount
1	10 875	19	$9.20	25	0
2	10 875	47	$9.50	21	0,1
3	10 875	49	$20.00	15	0
4	10 924	10	$31.00	20	0,1
5	10 924	28	$45.60	30	0,1
6	10 924	75	$7.75	6	0
7	10 924	19	$9.20	25	0

	orderid	productid	unitprice	qty	discount
1	10 924	10	$31.00	20	0,1
2	10 924	28	$45.60	30	0,1
3	10 924	75	$7.75	6	0
4	10 924	19	$9.20	25	0

	orderid	productid	unitprice	qty	discount
1	10 875	47	$9.50	21	0,1
2	10 875	49	$20.00	15	0
3	10 924	10	$31.00	20	0,1
4	10 924	28	$45.60	30	0,1
5	10 924	75	$7.75	6	0
6	10 924	19	$9.20	25	0

```
BEGIN;
    insert into "Sales"."OrderDetails"
    select 10924, productid, unitprice, qty, discount
    from "Sales"."OrderDetails"
    where orderid = 10875 and productid = 19;
SAVEPOINT my_savepoint;
    delete  from "Sales"."OrderDetails"
    where orderid = 10875;
-- Ошибочное действие... Его нужно забыть!
-- Необходимо удалить только одну запись
ROLLBACK TO my_savepoint;
    delete  from "Sales"."OrderDetails"
    where orderid = 10875 and productid = 19;
COMMIT;
SELECT *
FROM "Sales"."OrderDetails"
WHERE orderid IN (10875, 10924);
```

Особенности

- Поведение некоторых функций и типов данных PostgreSQL в транзакциях подчиняется особым правилам

- **Например**, изменения последовательностей (и следовательно, счётчика в столбце, объявленном как **serial**)
  - немедленно видны во всех остальных транзакциях
  - не откатываются назад, даже если выполнившая их транзакция прерывается

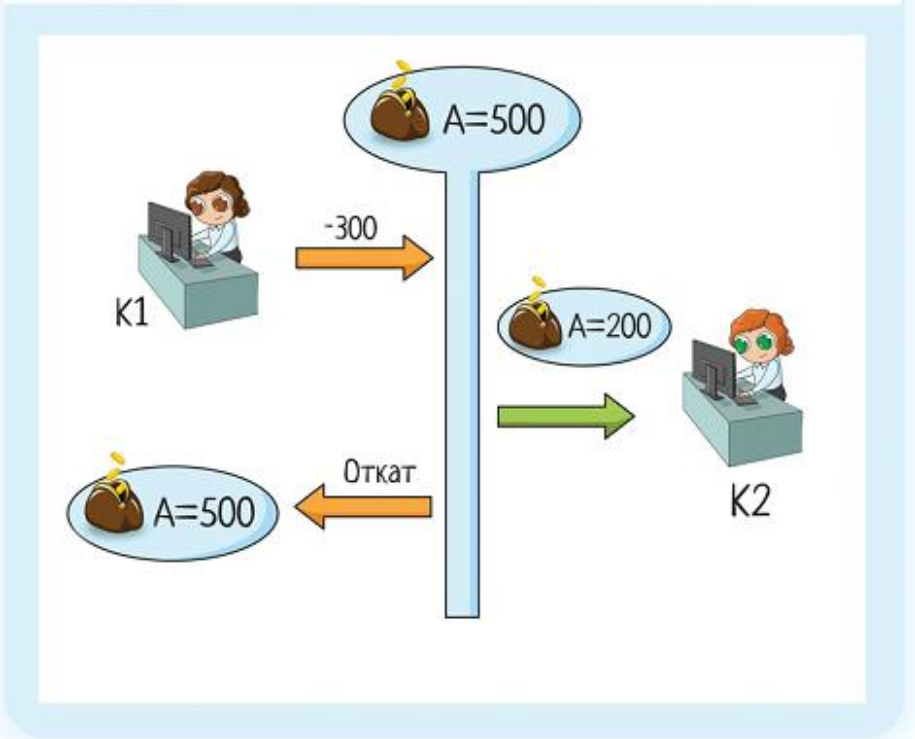
Изоляция транзакций

Проблемы конкурентного доступа

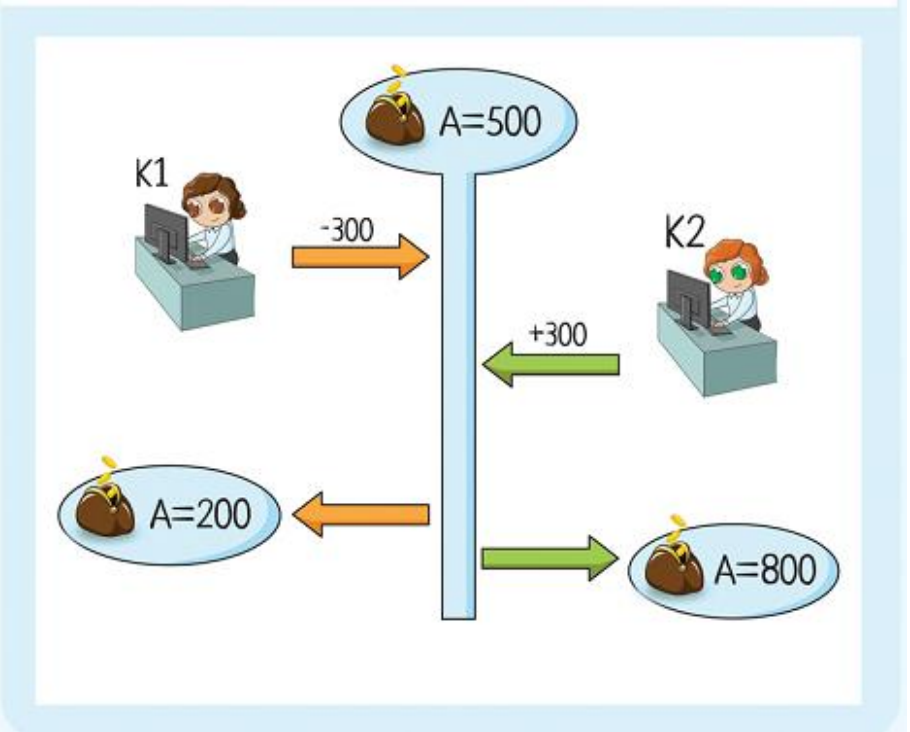
- **Грязное чтение (Dirty read)**
  - Транзакция читает данные, записанные параллельной **незавершённой** транзакцией
- **Потерянное обновление (Lost update)**
  - Выполняются два одновременных обновления; результат первого обновления потерян
- **Неповторяемое чтение (Non-repeatable read)**
  - Транзакция повторно читает те же данные, что и раньше, и обнаруживает, что они были изменены другой транзакцией (которая завершилась после первого чтения)
- **Фантомное чтение (Phantom read)**
  - Данные, удовлетворяющие некоторому условию, читаются одной транзакцией. В это время вторая транзакция добавляет/удаляет записи, удовлетворяющие этому условию
- **Двойное чтение (Double read)**
  - Данные в диапазоне читаются дважды, потому что значение ключа диапазона изменяется

Грязное чтение и Потерянное обновление

Dirty read



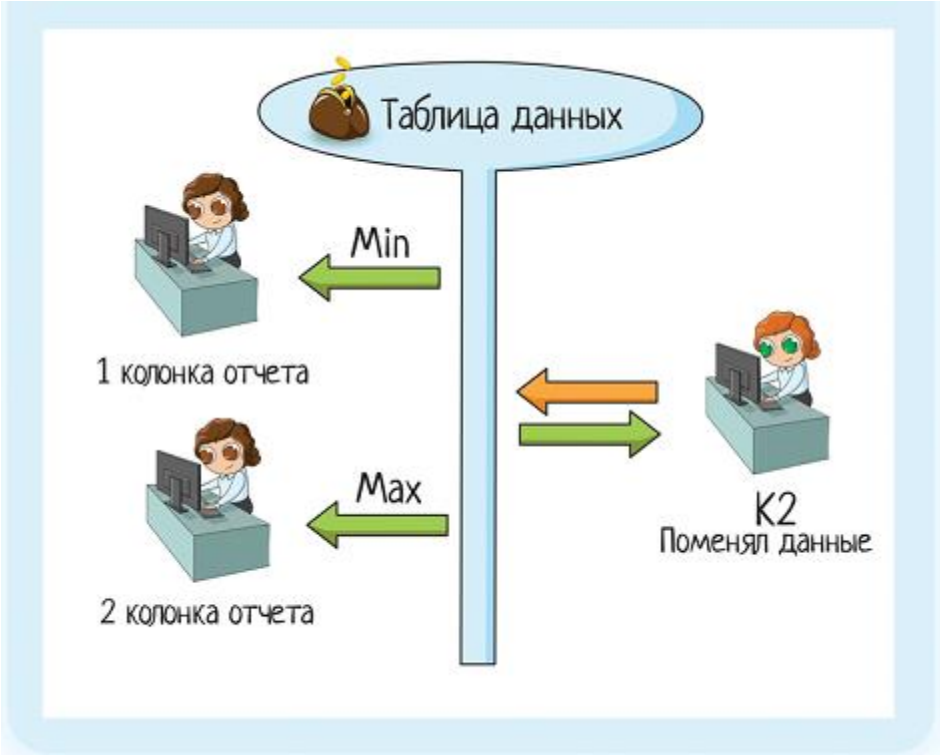
Lost update



https://habr.com/ru/post/555920

Неповторяемое чтение и Фантомное чтение

Non-repeatable read



Phantom read

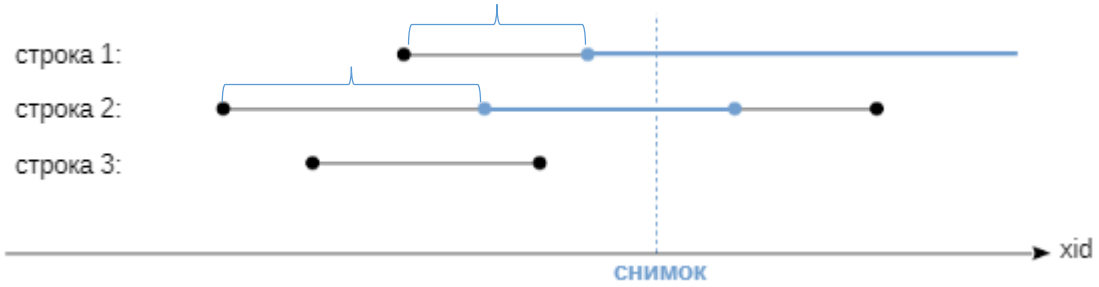
https://habr.com/ru/post/555920

Управление конкурентным доступом

- PostgreSQL поддерживает целостность данных, реализуя модель **MVCC**
- **MVCC –** Multiversion Concurrency Control, Многоверсионное управление конкурентным доступом:
  - каждый SQL-оператор видит снимок данных (версию базы данных) на определённый момент времени
  - это обеспечивает изоляцию транзакций для каждого сеанса баз данных
- Преимущество использования модели **MVCC**:
  - блокировки MVCC, полученные для чтения данных, не конфликтуют с блокировками, полученными для записи, и поэтому чтение никогда не мешает записи, а запись чтению
  - снижается уровень конфликтов блокировок и, таким образом, обеспечивается более высокая производительность в многопользовательской среде

Снимки данных

- PostgreSQL хранит несколько версий одной и той же строки
  - пишущая транзакция работает со своей версией, а читающая видит свою
  - каждая транзакция видит **только одну** из имеющихся версий каждой строки (или не видит ни одной)
- Версии записей помечаются двумя отметками, определяющими «время» действия данной версии – **xmin**, **xmax**
  - когда строка создается, она помечается номером транзакции, выполнившей команду **INSERT** (**xmin)**
  - когда удаляется — версия помечается номером транзакции, выполнившей **DELETE** (**но физически не удаляется**) (**xmax)**
  - UPDATE состоит из двух операций **DELETE** и **INSERT**
- **VACUUM –** серверный процесс, удаляющий неиспользуемые версии строк



Строка 1 – была изменена и зафиксирована до начала создания снимка

Строка 2 – начаты изменения, но еще не зафиксированы

Строка 3 – удалена до начала создания снимка

Уровни изоляции

- ***Read Uncommitted***
  - не поддерживается PostgreSQL: работает как Read Committed
  - не представляет практической ценности и не дает выигрыша в производительности
- ***Read Committed*** — используется по умолчанию
  - снимок строится на момент начала каждого оператора SQL
  - два одинаковых запроса, следующих один за другим, могут показать разные данные
- ***Repeatable Read***
  - снимок строится в начале транзакции (при выполнении первого оператора)
  - все запросы в одной транзакции видят одни и те же данные
  - транзакция может завершиться ошибкой сериализации
- ***Serializable***
  - гарантирует полную изоляцию
  - транзакция может завершиться ошибкой сериализации
  - приложение должно уметь повторять такие транзакции

Управление уровнем изоляции транзакций

```
SET TRANSACTION ISOLATION LEVEL
{ READ UNCOMMITTED | READ COMMITTED | REPEATABLE READ | SERIALIZABLE}
```

Уровень изоляции	«Грязное» чтение	Неповторяемое чтение	Фантомное чтение	Аномалия сериализации
READ UNCOMMITED	**Допускается, но не в PostgreSQL**	Возможно	Возможно	Возможно
READ COMMITTED	Невозможно	Возможно	Возможно	Возможно
REPEATABLE READ	Невозможно	Невозможно	**Допускается, но не в PostgreSQL**	Возможно
SERIALIZABLE	Невозможно	Невозможно	Невозможно	Невозможно

В связи с PostgreSQL архитектурой многоверсионного управления конкурентным доступом

Блокировка ядра СУБД

- **Блокировка** - это механизм, используемый ядром СУБД для синхронизации доступа нескольких пользователей к одному и тому же фрагменту данных в одно и то же время
- **Блокировки строк**
  - чтение никогда не блокирует строки
  - изменение строк блокирует их для изменений, но не для чтений
- **Блокировки таблиц**
  - запрещают изменение или удаление таблицы, пока с ней идет работа
  - запрещают чтение таблицы при перестроении или перемещении
- Блокировки устанавливаются автоматически и автоматически снимаются при завершении транзакции
  - **TRUNCATE** не может безопасно выполняться одновременно с другими операциями с этой таблицей, поэтому для избежания конфликта эта команда получает блокировку **ACCESS EXCLUSIVE** для данной таблицы

```
select *  from pg_locks;
```

Пример

```
--1 СЕССИЯ
begin;
select *  from public.client;



update public.client
set "Fname" = 'Alex'
where "ClientID" =12;



Commit;
```

1) Обе сессии видят одни и те же данные
2) 2 сессия удалила запись
3) 1 сессия находится в ожидании разблокировки записи
4) 2 сессия откатила изменения, 1 сессия – продолжила выполнение
5) 1 сессия зафиксировала изменения

```
--2 СЕССИЯ
begin;
select *  from public.client;

delete from public.client
where "ClientID" =12;

select *  from public.client;

rollback;
```