Изоляция транзакций

Изоляция транзакций

- Для реализация изоляции транзакций в современных СУБД используются:
  - протокол двухфазного блокирования
  - протокол изоляции на основе снимков
- ***Протокол двухфазного блокирования:***
  - в процессе выполнения транзакция блокирует строки, с которыми работает, а при завершении — освобождает блокировки
  - чем больше блокировок захватывает транзакция, тем лучше она изолирована от других транзакций
- ***Протокол изоляции на основе снимков:***
  - каждая транзакция работает с согласованным снимком данных на определенный момент времени, в который попадают только те изменения, которые были зафиксированы до момента создания снимка
  - такая изоляция автоматически не допускает грязное чтение
- ***Использование блокировок снижает производительность системы***

Multiversion Concurrency Control

- В PostgreSQL реализован многоверсионный вариант протокола изоляции на основе снимков (MVCC):
  - PostgreSQL хранит несколько версий одной и той же строки
  - При этом пишущая транзакция работает со своей версией, а читающая видит свою
- **Преимущество использования MVCC:**
  - Блокировки MVCC для чтения данных, не конфликтуют с блокировками для записи, и поэтому «Читатели» и «Писатели» не мешают друг другу
  - **Снижается** уровень конфликтов блокировок и, таким образом, обеспечивается более высокая производительность в многопользовательской среде

Multiversion Concurrency Control

- Единицей многоверсионности служат строки таблиц
  - Табличный блок содержит набор версий строк (tuples)
- Для каждой версии хранятся номера двух транзакций: начальной (**xmin**) и конечной (**xmax**)
  - когда строка создается, она помечается номером транзакции, выполнившей команду **INSERT** (**xmin)**
  - когда удаляется — версия помечается номером транзакции, выполнившей **DELETE** (**xmax**). ***Но физически не удаляется***!
  - **UPDATE** состоит из двух операций **DELETE** и **INSERT**

T3

```
BEGIN;
Update cars
set price = 550000
Where id = 2012;
```

T4

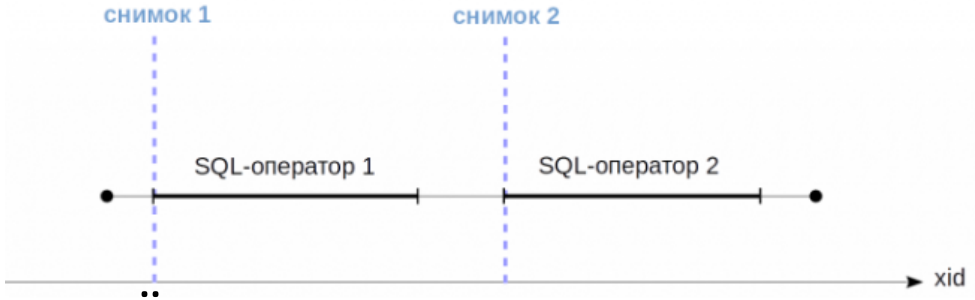
```
select *
from cars
where id = 2012;
```

				xmin,xmax
2012	Nissan Tiida I Рестайлинг	550000	v2	T3, 0
2012	Nissan Tiida I Рестайлинг	500000	v1	T2, T3
1988	LADA (ВАЗ) 2109	45000	v1	T2, 0

\pg_xact*

Номер транзакции	Статус транзакции	
T1	01	Rollback
T2	10	Commit
T3	00	Active

Снимок данных



- Транзакция всегда работает с определённым снимком данных
- **Снимок данных** – это согласованный срез базы данных на определённый момент времени
  - транзакция должна видеть согласованные данные, которые другие транзакции уже успели зафиксировать
  - транзакция не должна видеть изменения, которые другие транзакции успели произвести, но не успели зафиксировать
- Снимок основывается на номере транзакции и на списке активных транзакций в этот момент времени
  - Список транзакций нужен, чтобы рассматривать только зафиксированные транзакции в этот момент времени и отбрасывать не зафиксированные или начатые после текущей транзакции
- **Снимок – это не физическая копия данных БД!**

```
--Текущий снимок данных
SELECT txid_current_snapshot();
```

Снимок данных

1

```
--получение информации о записях
SELECT xmin, xmax, *
FROM public.t;
```

	xmin	xmax	id	col2
1	18963	0	1	val1
2	18963	0	2	val2

2

```
begin;
--номер текущей транзакции
SELECT txid_current();
```

	txid_current
1	18 965

3

```
update public.t
set col2 = 'Hello'
where id = 2;
SELECT xmin, xmax, * FROM public.t;
```

	xmin	xmax	id	col2
1	18963	0	1	val1
2	18965	0	2	Hello

4

```
SELECT xmin, xmax, *
FROM public.t;
```

	xmin	xmax	id	col2
1	18963	0	1	val1
2	18963	18965	2	val2

5

```
commit;
```

6

```
SELECT xmin, xmax, *
FROM public.t;
```

	xmin	xmax	id	col2
1	18963	18967	1	val1
2	18965	0	2	Hello

Транзакция, удалявшая запись была отменена

Управление уровнем изоляции транзакций

Блокировки ядра СУБД

- Многоверсионность позволяет обойтись только самым необходимым минимумом блокировок, тем самым увеличивая производительность системы
- **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
- **Время жизни блокировок**
  - Блокировки устанавливаются автоматически по мере необходимости или вручную
  - Снимаются автоматически при завершении транзакции

```
select *  from pg_locks;
```

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

```
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 архитектурой многоверсионного управления конкурентным доступом

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;
```

К прочтению

- https://habr.com/ru/company/postgrespro/blog/442804/
- https://devcenter.heroku.com/articles/postgresql-concurrency