

Транзакции

1. Основы

Транзакция

- **Транзакции** объединяют последовательность действий в одну операцию — логическую единицу работы с данными.
- Промежуточные состояния внутри последовательности операций **не видны** другим транзакциям.
- Если что-то помешает успешно завершить транзакцию, ни один из результатов этих действий не сохранится в базе данных.

ACID

- Atomicity (Атомарность)
- Consistency (Согласованность)
- Isolation (Изолированность)
- Durability (Устойчивость, долговечность)

Atomicity (атомарность):

- гарантирует, что результаты работы транзакции не будут зафиксированы в системе частично;
- будут либо выполнены все операции в транзакции, либо не выполнено ни одной.

Consistency (согласованность):

- после выполнения транзакции база данных должна быть в согласованном (**целостном**) состоянии;
- во время выполнения транзакции (после выполнения отдельных операций в рамках транзакции) согласованность не требуется.

Isolation (изолированность):

- во время выполнения транзакции другие транзакции (выполняющиеся параллельно) не должны оказывать влияние на результат транзакции.

Durability (долговечность):

- при успешном завершении транзакции результаты ее работы должны остаться в системе независимо от возможных сбоев оборудования, системы и тд.

BEGIN;

UPDATE BANK_ACCOUNT

SET AccValue = AccValue+1000 WHERE Client_ID = 1;

UPDATE BANK_ACCOUNT

SET AccValue = AccValue-1000 WHERE Client_ID = 2;

COMMIT;

- ROLLBACK;
- SAVEPOINT;

Какими бывают транзакции?

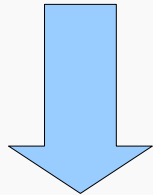
- **Неявные** (implicit) — СУБД начинает без явного указания;
 - любое действие осуществляется в контексте некоторой транзакции.
- **Явные** (explicit) — транзакции, которые задает пользователь самостоятельно (BEGIN, COMMIT).
- Если нет явного указания начала транзакции, любое предложение выполняется в рамках **неявной транзакции**;

Неявные транзакции

- `SELECT * FROM STUDENTS;`
- `INSERT INTO STUDENTS VALUES ...`

Неявные транзакции

```
SELECT * FROM STUDENTS;
```



```
BEGIN;
```

```
SELECT * FROM STUDENTS;
```

```
COMMIT;
```

BEGIN;

CREATE TABLE STUDENTS (StudId int, Name text);

CREATE TABLE EXAMS (ExamId int, ExName text);

ROLLBACK;

2. Время и транзакции

Как узнать время в PostgreSQL?

CURRENT_TIME, CURRENT_TIMESTAMP, now()

- Функция CURRENT_TIME:

```
SELECT CURRENT_TIME as my_time;
```

```
-- 13:30:45. ...
```

Как узнать время в PostgreSQL?

Функция CURRENT_TIME:

```
SELECT CURRENT_TIME as my_time_1;
```

```
-- 13:30:45. ...
```

```
-- [ ... ONE HOUR LATER ... ]
```

```
SELECT CURRENT_TIME as my_time_2;
```

```
-- ???
```

Как узнать время в PostgreSQL?

Функция CURRENT_TIME:

```
SELECT CURRENT_TIME as my_time_1;
```

```
-- 13:30:45. ...
```

```
-- [ ... ONE HOUR LATER ... ]
```

```
SELECT CURRENT_TIME as my_time_2;
```

```
-- 14:30:45. ...
```

Как узнать время внутри транзакции?

```
BEGIN;  
  
SELECT CURRENT_TIME as my_time_1;  
  
-- 13:30:45. ...  
  
-- [ ... ONE HOUR LATER ... ]  
  
SELECT CURRENT_TIME as my_time_2;  
  
-- ???  
  
COMMIT;
```


Как узнать время в PostgreSQL?

```
BEGIN;  
  
SELECT CURRENT_TIME as my_time_1;  
  
-- 13:30:45. ...  
  
-- [ ... ONE HOUR LATER ... ]  
  
SELECT CURRENT_TIME as my_time_2;  
  
-- 13:30:45. ...  
  
COMMIT;
```

Время и транзакции

- **CURRENT_TIME** — возвращает время начала транзакции (CURRENT_TIMESTAMP, transaction_timestamp);
- **clock_timestamp()** — возвращает текущее время (timestamp);

```
BEGIN;  
  
SELECT CURRENT_TIME as my_time_1;  
  
-- 13:30:45. ...  
  
-- [ ... ONE HOUR LATER ... ]  
  
SELECT clock_timestamp() as my_time_2;  
  
-- 2022-04-25 14:30:45. ...  
  
COMMIT;
```

3. Идентификация транзакций

Идентификатор транзакции (xid)

- Идентификатор транзакции (xid) — назначается СУБД для каждой новой транзакции;
 - xid — уникален;
- При создании/модификации записи хранится xid транзакции.

pg_current_xact_id()

- Возвращает xid текущей транзакции (bigint):
 - `SELECT pg_current_xact_id();` -- 7435
 - `txid_current()`
- Что можно сказать о выводе `txid_current()`?
 - `SELECT Column1, pg_current_xact_id()`
`FROM TABLE1;` -- **(1) ???**
 - `SELECT Column2, pg_current_xact_id()`
`FROM TABLE2;` -- **(2) ???**

Идентификатор транзакции (xid)

- Идентификатор транзакции (xid) — назначается СУБД для каждой новой транзакции;
 - xid — уникален.
- 32 бита, беззнаковое число.
- При создании/модификации записи хранится xid транзакции.

Как он хранится?

Как можно посмотреть xid?

```
SELECT * FROM STUDENTS;
```

STUDENTS

Stud_ID	Name	Surname	Group
1	Ivan	Ivanov	33313
2	Petr	Petrov	33314

Как можно посмотреть xid?

```
SELECT xmin, * FROM STUDENTS;
```

STUDENTS

xmin	Stud_ID	Name	Surname	Group
345	1	Ivan	Ivanov	33313
345	2	Petr	Petrov	33314

xid транзакции

Скрытые (системные) поля

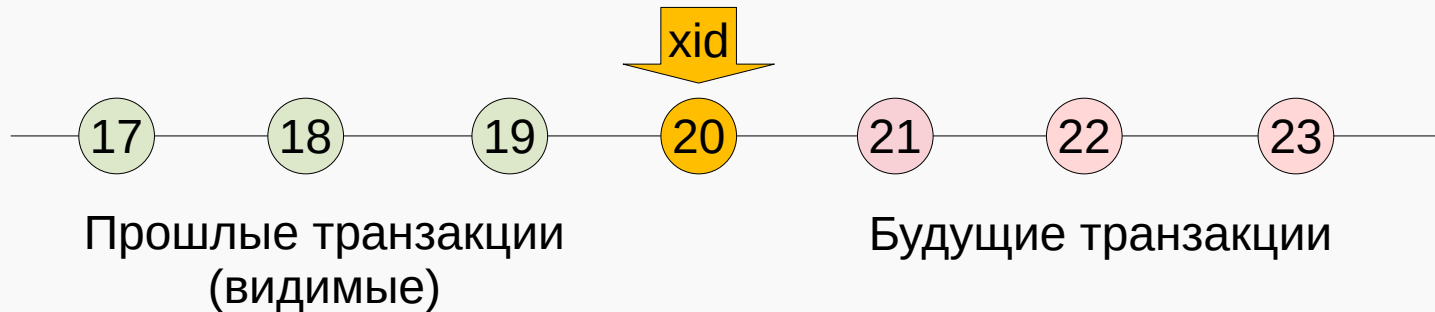
- **xmin** — хранит xid транзакции, в рамках которой запись (копия записи) была создана;
- **xmax** — хранит xid транзакции, в рамках которой запись была удалена;
- **cmin** — идентификатор команды, создавшей запись;
- **cmah** — идентификатор команды, удалившей запись;

xid — возможные значения

- xid — 32 бита, беззнаковое.
- В **простейшем** случае: чем больше xid транзакции — тем позже она началась:
 - xid до: прошлое
 - xid после: будущее
- xid — начинается с 3 (значения 0, 1, 2 — зарезервированы).

Могут ли xid закончиться?

- Начинается выдача значений заново (с 3);



- В простейшем случае: чем больше xid транзакции — тем позже она началась.

Проблема: что делать, когда перешли через MAX: **xid wraparound problem**

xid wraparound problem

- Установка статуса неактуальных записей:
 - Статус **FROZEN** — для неактуальных записей (~ транзакции для которых уже давно завершены).
- Экономия xid:
 - если в транзакции нет операций, изменяющих состояние БД, ей выдается виртуальный xid.
- WARNING: database test must be vacuumed within [...] transactions ...
 - **ПАНИКА!**

Выдача xid для транзакции

```
BEGIN;  
SELECT * FROM STUDENTS;  
UPDATE STUDENTS SET Group = 3100 ... ;  
COMMIT;
```

Выдача xid:
`SELECT txid_current_if_assigned();`
-- xid текущей транзакции
-- (если xid не назначен — NULL)

4. Реализация транзакций в PostgreSQL

Реализация транзакций

- Транзакции должны удовлетворять требованиям ACID.
- СУБД должна обеспечивать многопользовательский доступ:
 - разные пользователи могут запрашивать и изменять одни и те же данные в одно время.
- Существуют разные способы реализации транзакций:
 - могут основываться на блокировании ресурсов, синхронизации доступа к данным, создании копий данных, ...

Реализация транзакций в PostgreSQL

- Реализация транзакций (DML) в PostgreSQL — **MVCC**;
- MVCC — Multi-Version Concurrency Control
 - в PostgreSQL: Serializable Snapshot Isolation (SSI).
- Особенности MVCC (SSI):
 - при изменении данных в транзакции — создается новая копия данных;
 - существующие копии данных не изменяются;
 - при выборке данных берется подходящая для данной транзакции версия данных;

MVCC в PostgreSQL. Пример

STUDENTS

Stud_ID	Name	Surname	Group
1	Ivan	Ivanov	33313

```
UPDATE STUDENTS  
SET Group = 33314  
WHERE Stud_ID = 1;
```

STUDENTS

Stud_ID	Name	Surname	Group
1	Ivan	Ivanov	33313
1	Ivan	Ivanov	33314

Добавление данных в таблицу

```
SELECT xmin, * FROM STUDENTS;
```

STUDENTS

xmin	Stud_ID	Name	Surname	Group
345	1	Ivan	Ivanov	33313
345	2	Petr	Petrov	33314

```
BEGIN;
```

```
INSERT INTO STUDENTS VALUES (3, 'Vlad', 'Vladov',  
33313);
```

380

```
SELECT pg_current_xact_id_if_assigned(); COMMIT;
```

```
SELECT xmin, cmin * FROM STUDENTS;
```

xmin	xmin	Stud_ID	Name	Surname	Group
345	0	1	Ivan	Ivanov	33313
345	0	2	Petr	Petrov	33314
380	0	3	Vlad	Vladov	33313

Удаление данных

```
SELECT xmin, * FROM STUDENTS;
```

STUDENTS

xmin	Stud_ID	Name	Surname	Group
345	1	Ivan	Ivanov	33313
345	2	Petr	Petrov	33314
380	3	Vlad	Vladov	33313

```
BEGIN;
```

```
DELETE FROM STUDENTS;
```

390

```
SELECT pg_current_xact_id_if_assigned(); COMMIT;
```

xmin	xmax	cmx	Stud_ID	Name	Surname	Group
345	390	0	1	Ivan	Ivanov	33313
345	390	0	2	Petr	Petrov	33314
380	390	0	3	Vlad	Vladov	33313

Обновление данных

```
SELECT xmin, * FROM STUDENTS;
```

STUDENTS

xmin	Stud_ID	Name	Surname	Group
400	1	Egor	Egorov	33313

```
BEGIN;
```

```
UPDATE STUDENTS SET GROUP=33316;
```

```
SELECT pg_current_xact_id_if_assigned(); COMMIT;
```

410

xmin	xmax	Stud_ID	Name	Surname	Group
400	410	1	Egor	Egorov	33313
410	0	1	Egor	Egorov	33316

«Мертвая» запись

Блокировки данных

- Даже несмотря на использование MVCC в некоторых случаях исполнение транзакции блокируется:

Transaction 1:	Transaction 2:
BEGIN;	
	BEGIN;
UPDATE STUDENTS SET Group = 33314 WHERE Stud_ID = 1 ;	
	UPDATE STUDENTS SET Group = 33315 WHERE Stud_ID = 1 ;
COMMIT;	
	COMMIT;

Transaction 2
Блокируется!

Transaction 2
Разблокируется!

5. Изоляция транзакций

Изолированность

- Во время выполнения транзакции параллельные транзакции не должны оказывать влияния на её результат.
- Полноценная реализация требует очень много ресурсов!
- В реальных системах используется **концепция уровней изолированности транзакций**.


Виды конфликтов при параллельном доступе к данным

При организации параллельного доступа к данным могут возникнуть различные проблемы:

- Грязное чтение («dirty read»);
- Неповторяющееся чтение («nonrepeatable read»);
- Фантомное чтение («phantom read»);

«Грязное чтение»

Проблема возникает, если одна транзакция видит измененные данные другой (незавершенной транзакции):

Transaction 1:	Transaction 2:				
BEGIN;					
	BEGIN;				
UPDATE STUDENTS SET Group = 33314 WHERE Stud_ID = 1;					
	SELECT * FROM STUDENTS;	STUDENTS			
ROLLBACK;		Stud_ID	Name	Surname	Group
		1	Ivan	Ivanov	33314
	COMMIT;				

Неповторяющееся чтение

Проблема возникает, если один и тот же запрос в рамках транзакции возвращает разные результаты:

Transaction 1:	Transaction 2:				
BEGIN;					
	BEGIN;				
	SELECT * FROM STUDENTS;	STUDENTS			
		Stud_ID	Name	Surname	Group
		1	Ivan	Ivanov	33313
UPDATE STUDENTS SET Group = 33314 WHERE Stud_ID = 1 ;					
COMMIT;		STUDENTS			
	SELECT * FROM STUDENTS;	Stud_ID	Name	Surname	Group
		1	Ivan	Ivanov	33314
	COMMIT;				

Фантомное чтение

Проблема возникает, если один и тот же запрос в рамках транзакции возвращает разное число записей:

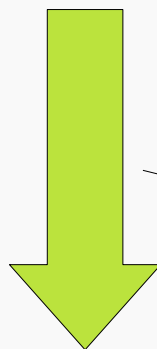
Transaction 1:	Transaction 2:	STUDENTS			
BEGIN;					
	BEGIN;	STUDENTS			
	SELECT * FROM STUDENTS;				
INSERT INTO STUDENTS VALUES (2, 'Viktor', 'Ivanov', 33315);		Stud_ID	Name	Surname	Group
		1	Ivan	Ivanov	33313
COMMIT;		STUDENTS			
	SELECT * FROM STUDENTS;				
	COMMIT;	Stud_ID	Name	Surname	Group
		1	Ivan	Ivanov	33313
		2	Viktor	Viktorov	33315

Сами данные не изменяются

Режимы для организации доступа к данным в PostgreSQL

По SQL-стандарту — 4 уровня изоляции:

- 1) Read Uncommitted
- 2) Read Committed
- 3) Repeatable Read
- 4) Serializable



Ограничения строже

Read Uncommitted — не поддерживается PostgreSQL.

Уровни изоляции в PostgreSQL

	Грязное чтение	Неповторяющееся чтение	Фантомное чтение	Аномалии сериализации
Read uncommitted/ нет в PostgreSQL	+	+	+	+
Read committed/ по умолчанию в PostgreSQL	-	+	+	+
Repeatable read	-	-	+(стандарт)/- (PostgreSQL)	+ (write skew/ read-only transaction skew)
Serializable	-	-	-	-

Выбор режима изоляции транзакции

```
BEGIN TRANSACTION ISOLATION LEVEL SERIALIZABLE;
```

```
BEGIN;
```

```
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
```

Нельзя менять по ходу транзакции!

Snapshot Isolation (SI)

- снимок (snapshot) — характеризует временное окно видимости данных для транзакции:
 - каждая транзакция видит свой набор данных, определяемый связанным с ней снимком;
 - снимок: какие xid (версии данных) видит данная транзакция;
 - можно узнать через `pg_current_snapshot()`.
- `pg_current_snapshot()` (`txid_current_snapshot()`) - возвращает текущий снимок:
 - Формат: **xmin:xmax:xid1,xid2**
 - Верхняя граница (xmax — не включается);

Предоставляется менеджером транзакций:

- READ COMMITTED: свой снимок для каждого выполняемого SQL-предложения.
- REPEATABLE READ, SERIALIZABLE: снимок — при выполнении только первого SQL-предложения.

Выбор подходящей версии записи определяется по **правилам проверки видимости**, которые используют:

- СНИМОК;
- значения x_{\min} , x_{\max} , c_{\min} , c_{\max} , ...

READ COMMITTED

Tr 1 (xid = 321):	Tr 2 (xid = 322):				
BEGIN REP. READ;					
	BEGIN READ COM.;				
Snapshot: 321:321	Snapshot: 321:321				
SELECT * FROM STUDENTS;	SELECT * FROM STUDENTS;	Stud_ID	Name	Surname	Group
Snapshot: 321:321		1	Ivan	Ivanov	33313
UPDATE STUDENTS SET Name = 'Egor' WHERE Stud_ID = 1;					
	Snapshot: 321:321				
	SELECT * FROM STUDENTS;	Stud_ID	Name	Surname	Group
		1	Ivan	Ivanov	33313
COMMIT;					
	Snapshot: 322:322				
	SELECT * FROM STUDENTS;	Stud_ID	Name	Surname	Group
		1	Egor	Ivanov	33313
	COMMIT;				

REPEATABLE READ

Tr 1 (xid = 321):	Tr 2 (xid = 322):				
BEGIN REP. READ;					
	BEGIN READ COM.;				
Snapshot: 321:321	Snapshot: 321:321				
SELECT * FROM STUDENTS;	SELECT * FROM STUDENTS;	Stud_ID	Name	Surname	Group
Snapshot: 321:321		1	Ivan	Ivanov	33313
UPDATE STUDENTS SET Name = 'Egor' WHERE Stud_ID = 1;					
	Snapshot: 321:321				
	SELECT * FROM STUDENTS;	Stud_ID	Name	Surname	Group
		1	Ivan	Ivanov	33313
COMMIT;					
	Snapshot: 321:321				
	SELECT * FROM STUDENTS;	Stud_ID	Name	Surname	Group
		1	Ivan	Ivanov	33313
	COMMIT;				

SERIALIZABLE, REPEATABLE READ

Потерянное обновление

Запрещено, чтобы сторонние транзакции изменяли данные, используемые в текущей транзакции:

Tx 1:	Tx 2:
BEGIN ISOLATION LEVEL SERIALIZABLE;	
	BEGIN ISOLATION LEVEL SERIALIZABLE;
UPDATE STUDENTS SET Name = 'Roman' WHERE Stud_ID = 1;	<div>Блокировка Tx 2 До завершения Tx 1</div>
	UPDATE STUDENTS SET Name = Upper(Name) WHERE Stud_ID = 1;
COMMIT;	<div>ERROR: could not serialize access...</div>
<div>Tx 1 завершится успешно</div>	COMMIT;



6. VACUUM

- Реализация транзакций в PostgreSQL — **MVCC**;
- MVCC — Multi-Version Concurrency Control
 - в PostgreSQL: Serializable Snapshot Isolation (SSI).
- Особенности MVCC (SI):
 - при изменении данных в транзакции — **создается новая копия данных**;
 - существующие копии данных **не изменяются**;

С накопленными неактуальными данными нужно что-то делать!

Неактуальные данные

STUDENTS

Stud_ID	Name	Surname	Group
1	Ivan	Ivanov	33313

```
UPDATE STUDENTS  
SET Group = 33314  
WHERE Stud_ID = 1;
```

После выполнения
транзакции эта строка
больше не нужна,
она занимает место

STUDENTS

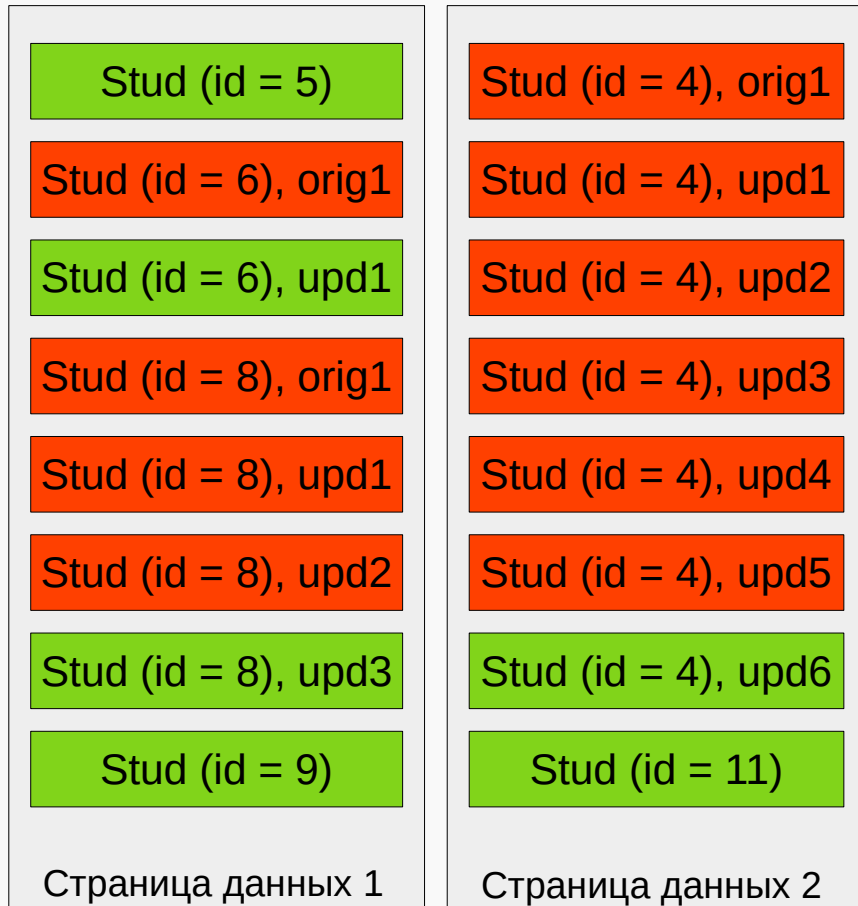
Stud_ID	Name	Surname	Group
1	Ivan	Ivanov	33313
1	Ivan	Ivanov	33314

VACUUM

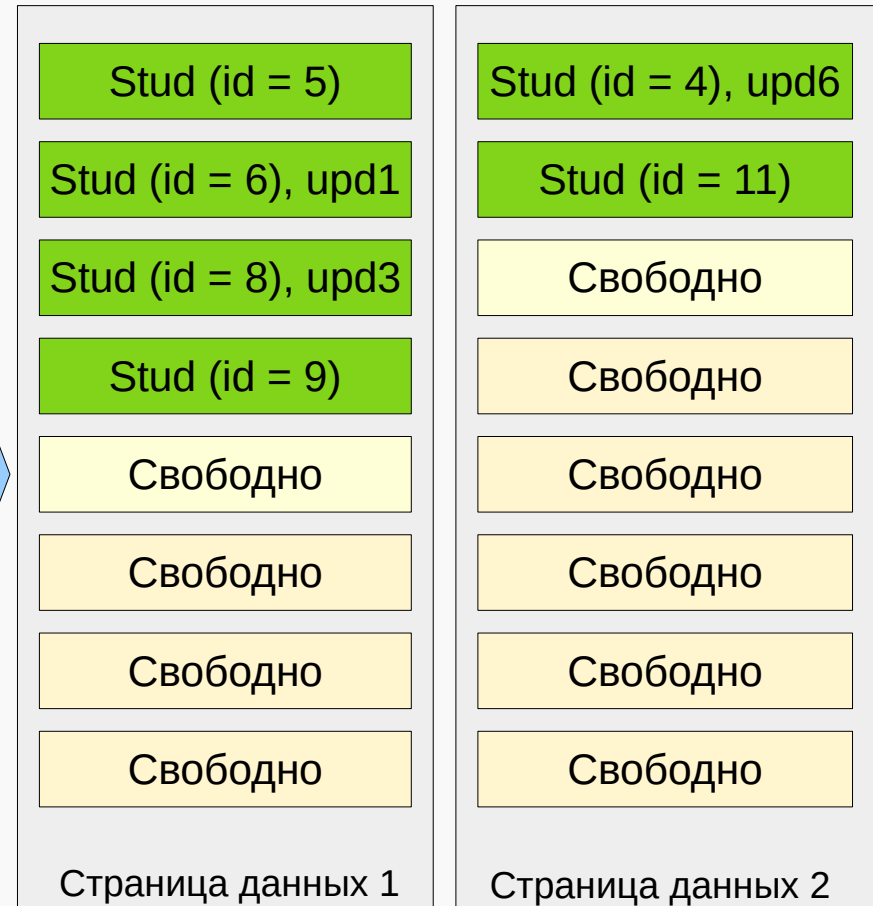
- Старые версии данных (мертвые записи) нужно удалять — они занимают место на диске.
- Для удаления таких записей используется VACUUM:
`VACUUM table[, table2, ...];`
- VACUUM:
 - По умолчанию;
 - `VACUUM FULL;`
 - `VACUUM FREEZE;`

VACUUM

До:

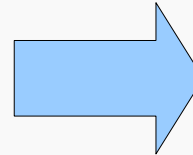
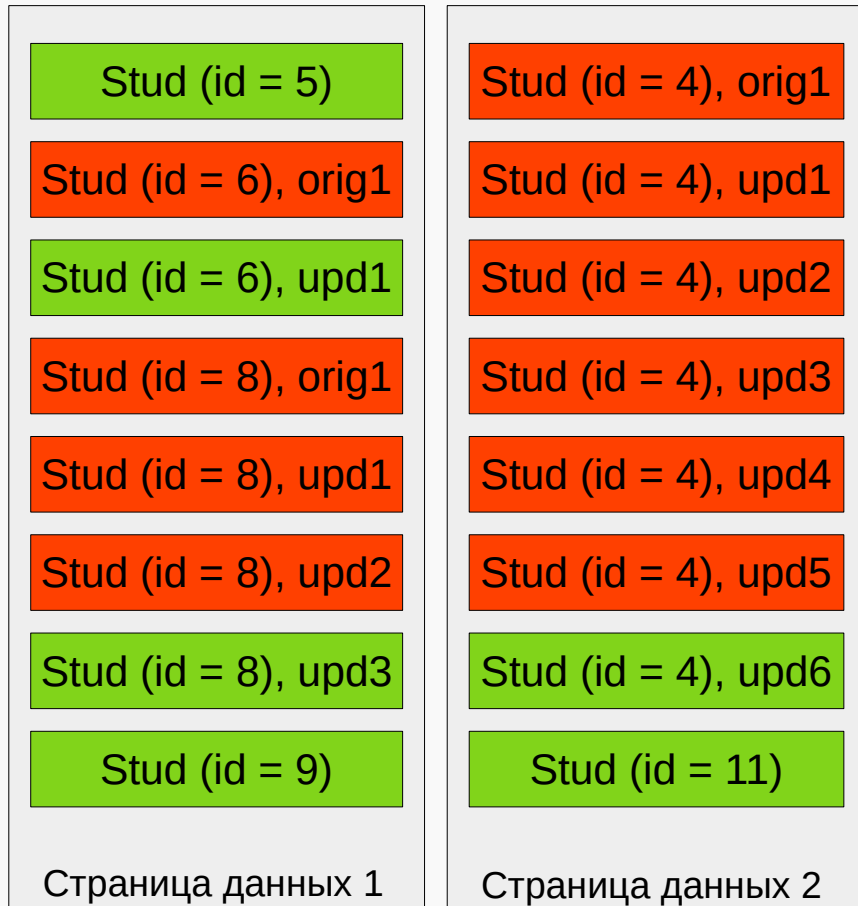


После VACUUM:

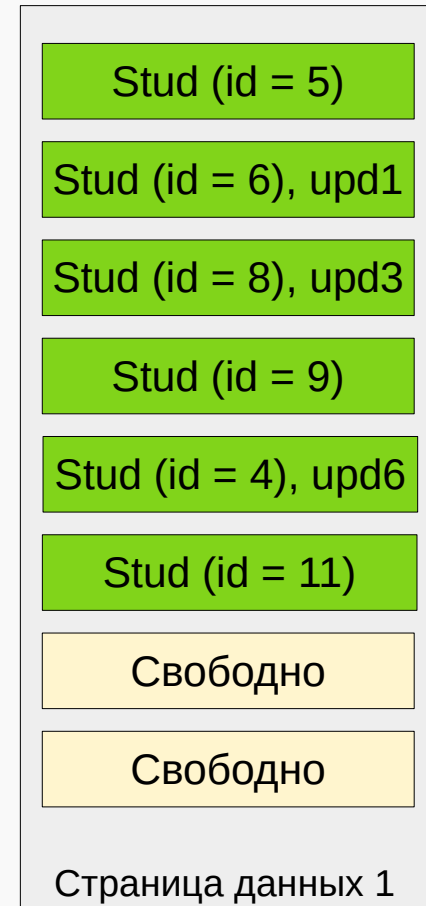


VACUUM FULL

До:



После VACUUM FULL:



autovacuum

- autovacuum = true (по умолчанию);
- меняется в postgresql.conf;
- autovacuum_vacuum_threshold
- autovacuum_vacuum_scale_factor
- autovacuum_cost_limit

Документация PostgreSQL:

<https://www.postgresql.org/docs/14/index.html>

Лицензия PostgreSQL:

PostgreSQL is released under the PostgreSQL License, a liberal Open Source license, similar to the BSD or MIT licenses.

PostgreSQL Database Management System
(formerly known as Postgres, then as Postgres95)

Portions Copyright © 1996-2022, The PostgreSQL Global Development Group

Portions Copyright © 1994, The Regents of the University of California

Permission to use, copy, modify, and distribute this software and its documentation for any purpose, without fee, and without a written agreement is hereby granted, provided that the above copyright notice and this paragraph and the following two paragraphs appear in all copies.

IN NO EVENT SHALL THE UNIVERSITY OF CALIFORNIA BE LIABLE TO ANY PARTY FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, INCLUDING LOST PROFITS, ARISING OUT OF THE USE OF THIS SOFTWARE AND ITS DOCUMENTATION, EVEN IF THE UNIVERSITY OF CALIFORNIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

THE UNIVERSITY OF CALIFORNIA SPECIFICALLY DISCLAIMS ANY WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE SOFTWARE PROVIDED HEREUNDER IS ON AN "AS IS" BASIS, AND THE UNIVERSITY OF CALIFORNIA HAS NO OBLIGATIONS TO PROVIDE MAINTENANCE, SUPPORT, UPDATES, ENHANCEMENTS, OR MODIFICATIONS.

<https://www.interdb.jp/pg/index.html>