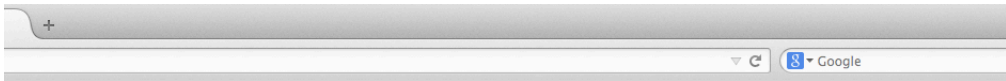


PostgreSQL: архитектура, настройка и оптимизация



Илья Космодемьянский
ik@postgresql-consulting.com

PostgreSQL-Consulting.com

[Donate](#) | [Contact](#)The world's most advanced
open source database.[Home](#) | [About](#) | [Download](#) | [Documentation](#) | [Community](#) | [Developers](#) | [Support](#) | [Your account](#)

PostgreSQL 2014-05-15 Beta Release

15th May 2014

The PostgreSQL Global Development Group is pleased to announce the availability of PostgreSQL 9.4 Beta 1, the first beta release of the upcoming 9.4 version.

- » [Release Announcement](#)
- » [Download](#)



> LATEST RELEASES

9.3.4 · March 20, 2014 · [Notes](#)
9.2.8 · March 20, 2014 · [Notes](#)
9.1.13 · March 20, 2014 · [Notes](#)
9.0.17 · March 20, 2014 · [Notes](#)
8.4.21 · March 20, 2014 · [Notes](#)

[Download](#) | [RSS](#)
[Why should I upgrade?](#)

> SHORTCUTS

» [Security](#)
» [International Sites](#)
» [Mailing Lists](#)

- Тому есть много причин

- Тому есть много причин
- Но на самом деле можно считать, что речь идет о поддержке транзакций

Счет А

Баланс 1000 руб

Счет А

Баланс 1000 руб
- 100 руб
- 200 руб

Сколько денег останется на балансе счета А?

Счет А

Баланс 1000 руб

- 100 руб

- 200 руб

900 руб?

800 руб?

700 руб?

Какой ответ правильный и почему?

700 руб

- И почему?

Какой ответ правильный и почему?

700 руб

- И почему?
- Здравый смысл и арифметика

Какой ответ правильный и почему?

700 руб

- И почему?
- Здравый смысл и арифметика
- Две операции списания не должны мешать друг другу

Очередность выполнения:

- -100 рублей потом -200
- Или наоборот

Очередность выполнения:

- -100 рублей потом -200
- Или наоборот
- Есть подозрение, что последовательно выполнять медленно и неэффективно

Действия выполняются независимо, одному действию неизвестно что происходит внутри другого

Какую еще проблему мы упустили?

Выполнение списания может **внезапно** прерваться

Если произошла ошибка
можем восстановить прежнее состояние данных
или
продолжить выполнение с того же места

Изолированность + Восстановимость = Атомарность

ACID

- Atomicity
- Consistency
- Isolation
- Durability

- Atomicity - восстановимость и изолированность
- Consistency - те самые арифметические правила и здравый смысл
- Isolation - независимость друг от друга
- Durability - мы считаем диск надежным хранилищем

Действие над данными,
обладающее свойствами ACID

- Переводит блок данных из одного консистентного состояния в другое
- Делает это атомарно
- Среди прочего, это означает что транзакции восстановимы и изолированы

Есть две проблемы чтобы заработало

- Конкурентный доступ к данным
- Внезапно все может упасть

Есть две проблемы чтобы заработало

- Конкурентный доступ к данным - для этого нужна **сериализация**
- Внезапно все может упасть - нужен какой-то умный алгоритм



- На первый взгляд просто
- Уровень concurrency бывает очень высоким



- На первый взгляд просто (например 2-3-10 транзакций вручную)
- Уровень concurrency бывает очень высоким

- Транзакции состоят из более мелких действий (списать со счета А, внести на счет В и т.п.)
- Наверное среди этих элементарных действий проще найти не влияющие друг на друга

- Транзакции состоят из более мелких действий (списать со счета А, внести на счет В и т.п.)
- Наверное среди этих элементарных действий проще найти не влияющие друг на друга - значит можно попробовать распараллелить

x и y

- ресурсы (блоки данных)

T1	T2
read(x)	write(x)
write(y)	write(y)

Дано две транзакции

x и y

- ресурсы (блоки данных)

T1

read(x)

write(y)

T2

write(x)

write(y)

История

$r_1(x)w_2(y)w_1(y)w_2(y)$

Семантика Эрбранта

$r_1(x)$ $w_1(x)$ $r_1(x)$ $w_1(x)$ $r_2(x)$ $r_2(x)$ $w_2(x)$ $w_2(x)$

$r_1(x)$ $w_1(x)$ $r_1(x)$ $w_1(x)$ $r_2(x)$ $r_2(x)$ $w_2(x)$ $w_2(x)$

ОК, нет конфликта

Конфликт, порядок имеет значение

Конфликт, порядок имеет значение

Конфликт, порядок имеет значение

Две истории для двух транзакций

T1

$r_1(x)$

$w_1(y)$

T2

$w_2(x)$

$w_2(y)$

$r_1(x) \leadsto w_2(x) w_1(y) \leadsto w_2(y)$

$r_1(x) \leadsto w_2(x) w_1(y) \curvearrowright w_2(y)$

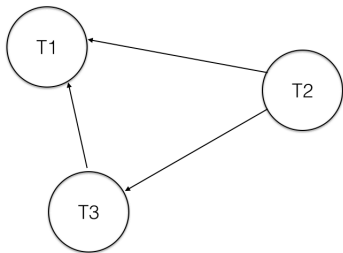
Две истории для двух транзакций

T1
 $r_1(x)$
 $w_1(y)$

T2
 $w_2(x)$
 $w_2(y)$

$r_1(x) \leadsto w_2(x) w_1(y) \leadsto w_2(y)$
 $r_1(x) \leadsto w_2(x) w_1(y) \curvearrowright w_2(y)$

Интуитивно понятно: стрелочки в одну сторону - хорошо



Граф конфликтов без циклов



история сериализуема

Понятно как решать задачу, но

- Искать циклы в графе затруднительно

Понятно как решать задачу, но

- Искать циклы в графе затруднительно
- Еще затруднительней - под OLTP нагрузкой

Понятно как решать задачу, но

- Искать циклы в графе затруднительно
- Еще затруднительней - под OLTP нагрузкой
- Надо искать более хитрые подходы

Понятно как решать задачу, но

- Искать циклы в графе затруднительно
- Еще затруднительней - под OLTP нагрузкой
- Надо искать более хитрые подходы Например блокировки

- Можно заблокировать все и сразу - перед началом любых действий заблокировать все что нужно

- Можно заблокировать все и сразу - перед началом любых действий заблокировать все что нужно
- Непонятно чем это лучше чем просто последовательное выполнение

- Можно заблокировать все и сразу - перед началом любых действий заблокировать все что нужно
- Непонятно чем это лучше чем просто последовательное выполнение
- К этому в конце концов приходят различные NoSQL

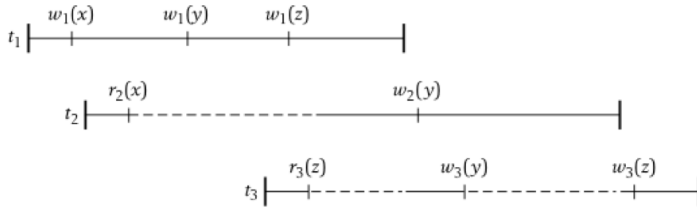
- Можно заблокировать все и сразу - перед началом любых действий заблокировать все что нужно
- Непонятно чем это лучше чем просто последовательное выполнение
- К этому в конце концов приходят различные NoSQL
- А ничем не лучше

- Блокировки берутся по мере необходимости
- Ни одна блокировка не снимается, покуда не взяты все необходимые блокировки



PGDAY'14
RUSSIA

2PL Диаграмма



from: Weikum, Vossen
Transactional Information Systems:
Theory, Algorithms, and the Practice of
Concurrency Control and Recovery

- Дедлоки

- Дедлоки
- Медленно - никто не хочет ждать блокировки

- Дедлоки
- Медленно - никто не хочет ждать блокировки
- Зато обеспечена сериализация

- Дедлоки
- Медленно - никто не хочет ждать блокировки
- Зато обеспечена сериализация **В любой нормальной базе 2PL основное средство обеспечения сериализации**
- * На самом деле бывают истории, которые 2PL разрулить не может в принципе

- Интуитивно понятно - чтобы не ждать блокировку, берем предыдущую версию

- Интуитивно понятно - чтобы не ждать блокировку, берем предыдущую версию
- PostgreSQL - версионник. MVCC это основа архитектуры

$$s = r_1(x)w_1(x)r_2(x)w_2(y)r_1(y)w_1(z)c_1c_2$$

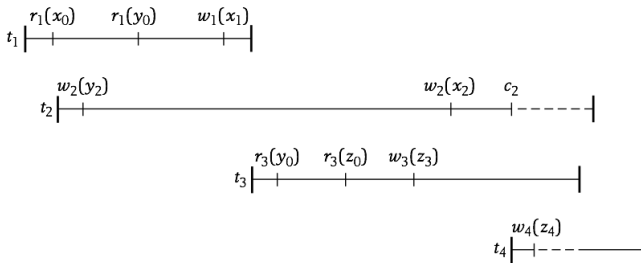
$$s = r_1(x)w_1(x)r_2(x)w_2(y)r_1(y)w_1(z)c_1c_2$$

- Если предыдущая версия у не доступна первой транзакции на чтение, историю сериализовать нельзя



PGDAY'14
RUSSIA

MV2PL Диаграмма



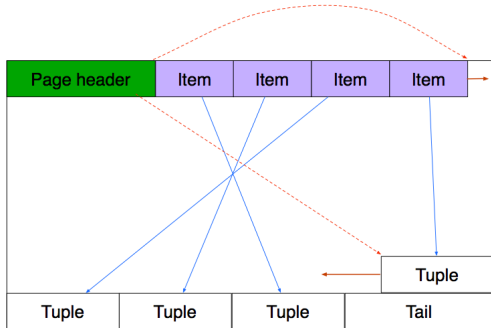
from: Weikum, Vossen
Transactional Information Systems:
Theory, Algorithms, and the Practice of
Concurrency Control and Recovery

- Tuple = Кортеж
- Page = Страница

- Tuple = Кортеж
- Page = Страница
- Страничная модель удобна для реализации транзакционных алгоритмов

- Tuple = Кортеж
- Page = Страница
- Страничная модель удобна для реализации транзакционных алгоритмов
- Безотносительно реляционной теории

Как устроена страница в PostgreSQL



- UPDATE = INSERT + DELETE

- UPDATE = INSERT + DELETE
- DELETE убирает из области видимости

- UPDATE = INSERT + DELETE
- DELETE убирает из области видимости
- Убранное из области видимости рано или поздно подчищает vacuum

```
tt=# INSERT into test(id) values(5);
```

```
INSERT 0 1
```

```
tt=# select *,xmin,xmax from test;
```

id	xmin	xmax
5	1266	0

(5 rows)

```
tt=# select txid_current();
```

txid_current
1267

(1 row)

```
tt=# begin;  
BEGIN  
tt=# UPDATE test set id=5 where id=4;  
UPDATE 1
```

В другой сессии:

```
tt=# select *,xmin,xmax from test;  
 id | xmin | xmax  
----+-----+-----  
  4 | 1264 | 1270  
(3 rows)
```

- Weikum, Vossen Transactional Information Systems: Theory, Algorithms, and the Practice of Concurrency Control and Recovery
- <http://momjian.us/main/writings/pgsql/mvcc.pdf> - Примеры и специфика PostgreSQL

Следующая важная задача

Что делать,
если в момент операции списания мы упали

- Для чего писать лог?

- Для чего писать лог?
- Что именно писать в лог?

- Для чего писать лог?
- Что именно писать в лог?
- Что потом с этим делать?

- Если упадем, "грязные" страницы из памяти будут потеряны
- Дампить сразу на диск в датафайл может быть затруднительно
- Будем писать их в лог

Например

- Type: **UPDATE** ID: 172 Undo: $x \leftarrow x_{old}$ Redo: $x \leftarrow x_{new}$
- Type: **OUTCOME** ID: 172 Status: **COMMITTED**

- Сканируем лог назад
- Ищем
- Winners: статус COMMITTED или ABORTED
- Losers: все остальные
- Redo: COMMITTED winners
- Undo: losers

- Redo
- Undo - версии прямо в датафайле

- Это была теория

- Это была теория
- Что об этом должен знать DBA?

- Это была теория
- Что об этом должен знать DBA?
- Да по-хорошему не только DBA

```
[pg@tr ~]$ ps ax | grep post
2199  ??  Ss   17:05.79 postgres: writer process      (postgres)
2200  ??  Ss    6:29.36 postgres: wal writer process  (postgres)
2201  ??  Ss    2:56.25 postgres: autovacuum launcher process (postgres)
2202  ??  Ss   12:49.19 postgres: stats collector process (postgres)
14046 ??  Ss    0:18.53 postgres: user mydb 127.0.0.1(48543) (postgres)
17252 ??  Is    0:11.93 postgres: user mydb 127.0.0.1(48800) (postgres)
26361 ??  Ss    0:01.20 postgres: user mydb 127.0.0.1(49512) (postgres)
1590 v0-  S   17:47.29 /home/pg/pg_bin/bin/postgres -D /db/postgres/db01/data
```

```
$PG_HOME/PG_VERSION  
$PG_HOME/base  
$PG_HOME/global  
$PG_HOME/pg_clog  
$PG_HOME/pg_hba.conf  
$PG_HOME/pg_ident.conf  
$PG_HOME/pg_multixact  
$PG_HOME/pg_notify  
$PG_HOME/pg_serial  
$PG_HOME/pg_stat_tmp  
$PG_HOME/pg_subtrans  
$PG_HOME/pg_tblspc  
$PG_HOME/pg_twophase  
$PG_HOME/pg_xlog -> /db/postgres/db01/db_wal  
$PG_HOME/postgresql.conf
```

```
pg@pglect01:~> ls pg_lecture/base/
```

```
1 12489 12497 16385
```

```
tt=> SELECT datname,oid FROM pg_database WHERE datname='postgres';
```

```
datname | oid
```

```
-----+-----
```

```
postgres | 12497
```

```
(1 row)
```

```
tt=> SELECT datname,oid FROM pg_database WHERE datname='tt';
```

```
datname | oid
```

```
-----+-----
```

```
tt      | 16385
```

```
(1 row)
```

```
pg@pglect01:~> ls pg_lecture/base/
1 12489 12497 16385
tt=> select relname,oid,relfilenode from pg_class where
relname='test';
 relname | oid | relfilenode
-----+-----+-----
 test   | 16388 |          16388
(1 row)
pg@pglect01:~> ls -l pg_lecture/base/16385/*
.....
-rw----- 1 pg pg   8192 Apr 27 01:46 pg_lecture/base/16385/16386
-rw----- 1 pg pg      0 Apr 27 01:42 pg_lecture/base/16385/16388
-rw----- 1 pg pg   512 Apr 27 01:41 pg_lecture/base/16385/pg_filenode.map
-rw----- 1 pg pg 106804 Apr 27 01:41 pg_lecture/base/16385/pg_internal.init
-rw----- 1 pg pg     4 Apr 27 01:41 pg_lecture/base/16385/PG_VERSION
```

(с помощью расширения *pg_buffercache*)

```
tt=# select reldatabase,relfilenode,relblocknumber,  
isdirty,usagecount from pg_buffercache WHERE  
relfilenode=16388;
```

reldatabase	relfilenode	relblocknumber	isdirty	usagecount
16385	16388	0	t	1

```
tt=# CHECKPOINT ;  
CHECKPOINT
```

```
tt=# select reldatabase,relfilenode,relblocknumber,isdirty,usagecount from pg_buffercache  
WHERE relfilenode=16388;
```

reldatabase	relfilenode	relblocknumber	isdirty	usagecount
16385	16388	0	f	1

- shared memory
- proc memory

```
postgres=# select name, setting, context from pg_settings where name ~ '(shared_b|work_mem)';
```

name	setting	context
maintenance_work_mem	16384	user
shared_buffers	16384	postmaster
work_mem	1024	user

(3 rows)

```
postgres=# select name, setting, context from pg_settings where name ~ '(shared_b|work_mem)';
```

name	setting	context
maintenance_work_mem	16384	user
shared_buffers	16384	postmaster
work_mem	1024	user

(3 rows)

- Много не бывает

- Много не бывает
- На самом деле зависит от дисков и размера базы

- Все-таки linux

- Все-таки linux
- И тем не менее linux

Какие еще важные настройки есть в postgresql.conf (о которых нельзя забывать)

- autovacuum
- WAL

- Одна из наихудших идей - отключить autovacuum вообще
- "Коробочные" настройки недостаточно агрессивны

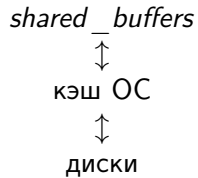
```
postgres=# select name, setting, context from pg_settings where category ~ 'Autovacuum';
```

name	setting	context
autovacuum	on	sighup
autovacuum_analyze_scale_factor	0.1	sighup
autovacuum_analyze_threshold	50	sighup
autovacuum_freeze_max_age	200000000	postmaster
autovacuum_max_workers	3	postmaster
autovacuum_multixact_freeze_max_age	400000000	postmaster
autovacuum_naptime	60	sighup
autovacuum_vacuum_cost_delay	20	sighup
autovacuum_vacuum_cost_limit	-1	sighup
autovacuum_vacuum_scale_factor	0.2	sighup
autovacuum_vacuum_threshold	50	sighup

```
(11 rows)
```

- Одна из наиболее распространенных проблем производительности
- Настройка тесно завязана на внутренности ОС

WAL - постановка проблемы



- *wal_buffers* (768kB → 16Mb)
- *checkpoint_segments* (3 - чекпойнт каждые 48Mb → 256 - 4Gb)
- *checkpoint_timeout* (что первое наступит)
- *checkpoint_completion_target* (для распределения чекпойнтов)

```
postgres=# select pg_stat_reset_shared('bgwriter');
-[ RECORD 1 ]-----+-
pg_stat_reset_shared |
```

```
postgres=# select * from pg_stat_bgwriter ;
-[ RECORD 1 ]-----+-----
checkpoints_timed      | 0
checkpoints_req        | 0
checkpoint_write_time  | 0
checkpoint_sync_time   | 0
buffers_checkpoint     | 0
buffers_clean          | 0
maxwritten_clean       | 0
buffers_backend        | 0
buffers_backend_fsync  | 0
buffers_alloc          | 0
stats_reset            | 2014-03-15 00:08:54.931266-04
```

- Графики!
- %util лучше чем iops

```
root@tr:~# iostat -d -x 1
```

```
Linux 3.2.0-4-amd64 (tr) 03/14/2014 _x86_64_ (1 CPU)
```

Device:	rrqm/s	wrqm/s	r/s	w/s	rkB/s	wkB/s	avgrq-sz	avgqu-sz	await	r_await	w_await	svctm	%util
sda	0.00	0.69	0.03	0.64	0.65	7.98	25.82	0.00	0.40	0.82	0.38	0.28	0.02

Device:	rrqm/s	wrqm/s	r/s	w/s	rkB/s	wkB/s	avgrq-sz	avgqu-sz	await	r_await	w_await	svctm	%util
sda	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Device:	rrqm/s	wrqm/s	r/s	w/s	rkB/s	wkB/s	avgrq-sz	avgqu-sz	await	r_await	w_await	svctm	%util
sda	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

- ext4/xfs barrier
- checkpoint это важно - не загружайте диски попусту другими задачами

```
postgres=# select name, setting, context, max_val, min_val from pg_settings where name ~ 'bgwr';
```

name	setting	context	max_val	min_val
bgwriter_delay	200	sighup	10000	10
bgwriter_lru_maxpages	100	sighup	1000	0
bgwriter_lru_multiplier	2	sighup	10	0

(3 rows)

- "Честный" RAID
- Если нужна производительность - BBU
- 2.5" SAS
- не все SSD одинаково полезны (intel dc 3700 vs desktop class)
- Дисковый массив хорошо, но нужно уметь настраивать

- Покупать хорошие (я серьезно)
- *synchronous_commit* → *off* (но надо представлять себе риски)
- больше воркеров автовакуума

```
postgres=# show commit_delay;
```

```
commit_delay
```

```
-----
```

```
0
```

```
(1 row)
```

```
postgres=# show commit_siblings;
```

```
commit_siblings
```

```
-----
```

```
5
```

```
(1 row)
```

- Дамп != backup
- backup - некая специальная копия, с которой можно восстановиться на момент последней удачно завершившейся перед аварией транзакции
- На практике дампы лучше все равно снимать (помимо backup'а)
- Единственный способ валидировать backup - восстановиться с него
- DBA может не знать SQL (это плохо, но что делать), знать backup/recovery - обязан
- Конфиги! (лучше вообще в забэкапленном git'е)

- *SELECT pg_start_backup('label', true);*
- rsync и магия
- *SELECT pg_stop_backup();*

- `SELECT pg_start_backup('label', true);`
- rsync и магия
- `SELECT pg_stop_backup();`
- Эффективно `pg_basebackup` делает то же самое

- recovery.conf
- *restore_command* = 'cp /mnt/server/archivedir/%f %p'

- recovery.conf
- *restore_command* = 'cp /mnt/server/archivedir/%f %p'
- магия!

- HotStandby и все остальное
- Насчет всего остального лучше 100500 раз подумать
- Репликация путем пересылки WAL есть суть постоянное восстановление из backup'а на другом сервере
- Почему репликация это плохо
- Почему Master-Master/Active-Active это очень плохо
- Настройки

- Правильная постановка задачи
- Master-Master vs HotStandby для отказоустойчивости

```
postgres=# select name, setting, context from pg_settings where category ~ 'Repl';
```

name	setting	context
hot_standby	off	postmaster
hot_standby_feedback	off	sighup
max_standby_archive_delay	30000	sighup
max_standby_streaming_delay	30000	sighup
max_wal_senders	0	postmaster
synchronous_standby_names		sighup
vacuum_defer_cleanup_age	0	sighup
wal_keep_segments	0	sighup
wal_receiver_status_interval	10	sighup
wal_receiver_timeout	60000	sighup
wal_sender_timeout	60000	sighup

```
(11 rows)
```

- Что вообще сейчас происходит?
- В порядке-ли базовые настройки
- Стейтменты
- Раскопки лога

```
pg@pglect01:~/dellstore2-normal-1.0> psql dellstore2 -c 'explain (analyze on, buffers on) select count(*) from customers'
```

QUERY PLAN

Aggregate (cost=738.00..738.01 rows=1 width=0) (actual time=8.357..8.357 rows=1 loops=1)

Buffers: shared hit=488

-> Seq Scan on customers (cost=0.00..688.00 rows=20000 width=0) (actual time=0.026..5.728 rows=20000 loops=1)

Buffers: shared hit=488

Total runtime: 8.607 ms

(5 rows)

"Этот запрос не будет работать быстро никогда"

- `count(*)`
- join на 300 таблиц
- Запрос возвращает клиенту 1 000 000 000 строк

Несколько слов о connection pool

- pgbouncer
- pgpoolII
- разное

- nginx-style
- три режима пула - transaction, statement, session
- must have