



**PG BootCamp Russia 2025 Ekaterinburg**

[PGBootCamp.ru](http://PGBootCamp.ru)

# Работа с логической репликацией

Александр Никитин. DBA Team

# Немного обо мне

- С базами начал работать где-то в 2001 г. (MS FoxPro, FireBird, Oracle, PostgreSQL).
- DBA с 2014 года.
- С 2020 г. выступаю на крупных конференциях. В
- консалтинге с 2021 года.
- Член Программного комитета PG BootCamp Russia

# Содержание

- Немного теории
- Создадим логическую репликацию таблицы
- Поработаем с конфликтами
- Произведём преобразование из физической реплики в логическую
- Нюансы

# Теория

Есть 2 типа репликации:

- Потокковая (физическая)
- Логическая

## Логическая:

Модель "публикация/подписчик"

Можно реплицировать отдельные таблицы

Можно реплицировать не все команды DML

Команды DDL не реплицируются

# Практика №1. Простые вещи.

Rocky Linux 8 (8\_1)  
(192.168.0.170)

Rocky Linux 8 (8\_2)  
(192.168.0.171)

# Практика №1

```
CREATE TABLE employees (  
    employee_id SERIAL PRIMARY KEY,  
    t VARCHAR(50) NOT NULL  
);  
  
insert into employees (t) values ('aaa');  
  
CREATE PUBLICATION my_pub FOR ALL TABLES;  
  
create user replica replication password 'ffffggg';  
  
grant pg_read_all_data to replica;  
  
Настроили wal_level, pg_hba.conf
```

# Практика №1

```
$ touch ~/.pgpass && chmod 600 ~/.pgpass  
$ echo "#hostname:port:database:username:password" >> ~/.pgpass  
$ echo "*:*:*:replica:ffffggg" >> ~/.pgpass
```

```
CREATE TABLE employees (  
    employee_id SERIAL PRIMARY KEY,  
    t VARCHAR(50) NOT NULL);
```

```
CREATE SUBSCRIPTION my_sub CONNECTION 'dbname=postgres  
user=replica host=192.168.0.170 port=5432' PUBLICATION my_pub;  
  
table employees;
```



# Практика №1

```
$ touch ~/.pgpass && chmod 600 ~/.pgpass
$ echo "#hostname:port:database:username:password" >> ~/.pgpass
$ echo "*:*:*:replica:ffffggg" >> ~/.pgpass
```

```
CREATE TABLE employees (
    employee_id SERIAL PRIMARY KEY,
    t VARCHAR(50) NOT NULL);
```

```
CREATE SUBSCRIPTION my_sub CONNECTION 'dbname=postgres
user=replica host=192.168.0.170 port=5432' PUBLICATION my_pub;
```

```
table employees;
```

<i>employee_id</i>	<i>t</i>
<hr/>	
1	aaa

# Практика №1



```
insert into employees (t) values ('bbb');
```



```
table employees;
```

# Практика №1



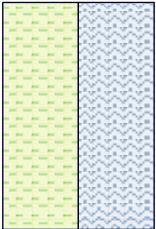
```
insert into employees (t) values ('bbb');
```




```
table employees;
```

<i>employee_id</i>	<i>t</i>
1	aaa
2	bbb

# Практика №1



```
CREATE TABLE emp2 (  
    employee_id SERIAL PRIMARY KEY,  
    t VARCHAR(50) NOT NULL  
);
```



```
Insert into emp2(t) values('ccc');
```

# Практика №1

Table emp2;

<i>employee_id</i>	<i>t</i>
1	ccc

# Практика №1

Table emp2;

<i>employee_id</i>	<i>t</i>
-----+-----	
1	ccc

Table emp2;

employee_id	t
-----+-----	
(0 rows)	

*CREATE PUBLICATION my\_pub FOR ALL TABLES;*

# Практика №1

Table emp2;

<i>employee_id</i>	<i>t</i>
-----+-----	
1	ccc

Table emp2;

employee_id	t
-----+-----	
(0 rows)	

ALTER SUBSCRIPTION my\_sub refresh publication;

Table emp2;

employee_id	t
-----+-----	
1	ccc

# Практика №1

```
postgres=# \dRp+
Publication my_pub
-[ RECORD 1 ]-----
Owner          | postgres
All tables     | t
Inserts        | t
Updates        | t
Deletes        | t
Truncates      | t
Via root       | f
```



# Практика №1

```
postgres=# \dRs+
List of subscriptions
-[ RECORD 1 ]-----+-----
Name                | my_sub
Owner                | postgres
Enabled              | t
Publication          | {my_pub}
Binary               | f
Streaming            | off
Two-phase commit     | d
Disable on error     | f
Origin               | any
Password required    | t
Run as owner?        | f
Failover             | f
Synchronous commit   | off
Conninfo              | dbname=postgres user=replica host=192.168.0.170 port=5432
Skip LSN              | 0/0
```

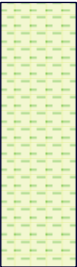
```
postgres=# \dRp+
Publication my_pub
-[ RECORD 1 ]-----
Owner          | postgres
All tables     | t
Inserts        | t
Updates        | t
Deletes        | t
Truncates      | t
Via root       | f
```

# Практика №1. Итоги

- Мы познакомились с репликацией.
- Узнали, что подписку можно обновлять.
- /dRp+ - список подписок
- /dRs+ - список публикаций

Почистим за собой и перейдём ко второй практике.


# Практика №2. Коллизии



```
CREATE TABLE t1 (id integer);
```

```
INSERT INTO t1 VALUES (5);
```

```
CREATE PUBLICATION my_pub FOR TABLE t1;
```



```
CREATE TABLE t1 (id integer UNIQUE);
```

```
CREATE SUBSCRIPTION my_sub CONNECTION 'dbname=postgres  
user=replica host=192.168.0.170 port=5432' PUBLICATION my_pub  
with (disable_on_error = true);
```

# Практика №2.


```
BEGIN; -- Txn1
  INSERT INTO t1 VALUES (1);
COMMIT;
```

```
BEGIN; -- Txn2
  INSERT INTO t1 VALUES (generate_series(2, 4));
  INSERT INTO t1 VALUES (5);
  INSERT INTO t1 VALUES (generate_series(6, 8));
COMMIT;
```

```
BEGIN; -- Txn3
  INSERT INTO t1 VALUES (9);
COMMIT;
```


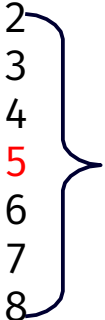
```
BEGIN; -- Txn4
  INSERT INTO t1 VALUES (10);
COMMIT;
```

# Практика №2.



```
table t1;  
id
```


-----  
5  
1  
2  
3  
4  
5  
6  
7  
8  
9  
10




```
table t1;  
id
```

-----  
5  
1

# Практика №2.



```
table t1;  
id
```



```
table t1;  
id
```

-----

-----

5

5

1

1

2

3

4

5

6

7

8

9

10



ERROR: duplicate key value violates unique constraint "t1\_id\_key"

**DETAIL:** Key (id)=(5) already exists.

**CONTEXT:** processing remote data for replication origin "pg\_24730" during message type "INSERT" for replication target relation "public.t1" in transaction 1026, finished at 0/43904C50

**LOG:** subscription "my\_sub" has been disabled because of an error

# Практика №2.

Запомним это состояние.

Рассмотрим пути разрешения коллизий.

# Практика №2.

## 1 Вариант:

DETAIL: Key (id)=(5) already exists.



# Практика №2.

## 1 Вариант:

DETAIL: Key (id)=(5) already exists.

```
delete from t1 where id = 5;
```

```
ALTER SUBSCRIPTION my_sub ENABLE;
```

```
table t1;
```

```
id
```

```
----
```

```
1
```

```
2
```

```
3
```

```
4
```

```
5
```

```
6
```

```
7
```

```
8
```

```
9
```

```
10
```

# Практика №2.


## 2 Вариант:

CONTEXT: processing remote data for replication origin "**pg\_24730**" during message type "INSERT" for replication target relation "public.t1" in transaction 1026, finished at **0/43904C50**

```
SELECT  
pg_replication_origin_advance('pg_24730',  
'0/43904C51'::pg_lsn);
```

```
ALTER SUBSCRIPTION my_sub ENABLE;
```

table t1; id	id
----	----
5	5
1	1
2	9
3	10
4	
<b>5</b>	
6	
7	
8	
9	
10	



# Практика №2.

2 Вариант:

finished at 0/43904C50

/usr/pgsql-17/bin/pg\_waldump 00000002000000000000000043

```
lsn: 0/43904A20, prev 0/439049E8, desc: INSERT off: 2, flags: 0x08, blkref #0: rel 1663/5/24734 blk 0
lsn: 0/43904A60, prev 0/43904A20, desc: COMMIT 2025-02-19 03:38:06.132078 EST
lsn: 0/43904A90, prev 0/43904A60, desc: INSERT off: 3, flags: 0x08, blkref #0: rel 1663/5/24734 blk 0
lsn: 0/43904AD0, prev 0/43904A90, desc: INSERT off: 4, flags: 0x08, blkref #0: rel 1663/5/24734 blk 0
lsn: 0/43904B10, prev 0/43904AD0, desc: INSERT off: 5, flags: 0x08, blkref #0: rel 1663/5/24734 blk 0
lsn: 0/43904B50, prev 0/43904B10, desc: INSERT off: 6, flags: 0x08, blkref #0: rel 1663/5/24734 blk 0
lsn: 0/43904B90, prev 0/43904B50, desc: INSERT off: 7, flags: 0x08, blkref #0: rel 1663/5/24734 blk 0
lsn: 0/43904BD0, prev 0/43904B90, desc: INSERT off: 8, flags: 0x08, blkref #0: rel 1663/5/24734 blk 0
lsn: 0/43904C10, prev 0/43904BD0, desc: INSERT off: 9, flags: 0x08, blkref #0: rel 1663/5/24734 blk 0
lsn: 0/43904C50, prev 0/43904C10, desc: COMMIT 2025-02-19 03:38:06.154881 EST
lsn: 0/43904C80, prev 0/43904C50, desc: INSERT off: 10, flags: 0x08, blkref #0: rel 1663/5/24734 blk 0
lsn: 0/43904CC0, prev 0/43904C80, desc: COMMIT 2025-02-19 03:38:06.162175 EST
lsn: 0/43904CF0, prev 0/43904CC0, desc: INSERT off: 11, flags: 0x08, blkref #0: rel 1663/5/24734 blk 0
lsn: 0/43904D30, prev 0/43904CF0, desc: COMMIT 2025-02-19 03:38:06.677102 EST
```

id  
---  
5  
1  
9  
10

У вас есть возможность прыгнуть не туда

# Практика №2.

## 3 Вариант:

С 15 версии

```
ALTER SUBSCRIPTION name SKIP ( skip_option = value )
```

```
alter subscription my_sub skip (lsn = '0/43904C50');
```

```
alter subscription my_sub enable;
```

id  
---  
5  
1  
9  
10

# Практика №2.

## 3 Вариант:

С 15 версии

```
ALTER SUBSCRIPTION name SKIP ( skip_option = value )
```

```
alter subscription my_sub skip (lsn = '0/43904C50');
```

```
alter subscription my_sub enable;
```

id  
---  
5  
1  
9  
10

<https://github.com/postgres/postgres/blob/83ea6c54025bea67bcd4949a6d58d3fc11c3e21b/src/backend/commands/subscriptioncmds.c#L1520C5-L1521C52>

```
/* ALTER SUBSCRIPTION ... SKIP supports only LSN option */
```

# Практика №2.

Как обнаружить, что у нас есть проблема?

# Практика №2.

Как обнаружить, что у нас есть проблема?

```
table pg_stat_subscription_stats; (с 15 версии PG)
```

subid	subname	apply_error_count	sync_error_count	stats_reset
24598	my_sub	1	0	

```
SELECT oid, subname, subenabled, subdisableonerr FROM pg_subscription;
```

oid	subname	subenabled	subdisableonerr
24598	my_sub	f	t

# Практика №2.

Как обнаружить, что у нас есть проблема?

```
table pg_stat_subscription_stats;  (с 15 версии PG)
subid | subname | apply_error_count | sync_error_count | stats_reset
-----+-----+-----+-----+-----
24598 | my_sub  |                  1 |                  0 |
```

```
Select pg_stat_reset_subscription_stats ( oid )
```

Oid или NULL, если для всех.



# Практика №2. Итоги

Получили коллизию, смотрим в лог, там вся информация, необходимая для разрешения коллизии.

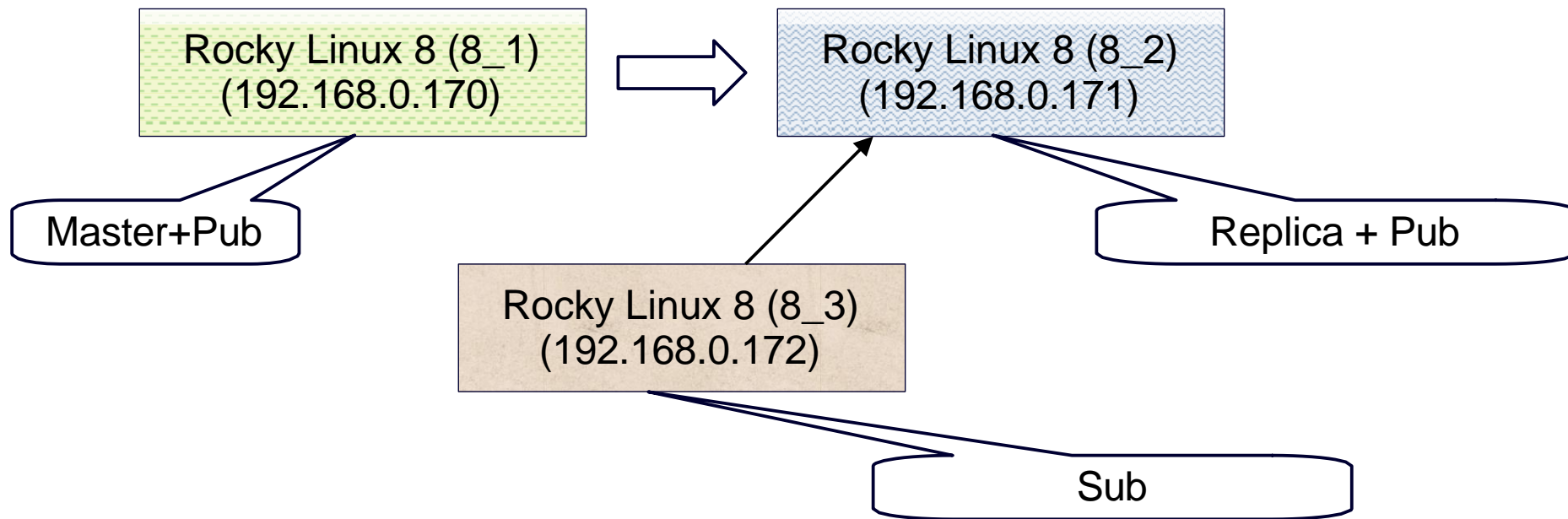
Варианты разрешения коллизий:

- 1) Удаление записи, мешающей накату изменений
- 2) `pg_replication_origin_advance`
- 3) `alter subscription ... skip (lsn = '...');` (с 15 версии).

Представления: `pg_stat_subscription_stats`, `pg_subscription`

Функция: `pg_stat_reset_subscription_stats`

# Практика №3. Что нового?



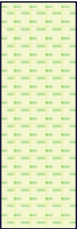
hot\_standby / hot\_standby\_feedback

# Практика №3



Создадим реплику:

```
pg_basebackup -D /var/lib/pgsql/17/data -R -X stream -c fast -  
P --no-slot --no-manifest -d "host=192.168.0.170 user=replica  
application_name=rocky8_2" -Z 'server-zstd:level=3'
```



```
CREATE TABLE emp (  
    employee_id SERIAL PRIMARY KEY,  
    t VARCHAR(50) NOT NULL  
);  
insert into emp (t) values ('aaa');
```



Table emp;

employee_id	t
1	aaa

# Практика №3

Публикация для всех таблиц у нас уже создана

```
CREATE TABLE emp (  
    employee_id SERIAL PRIMARY KEY,  
    t VARCHAR(50) NOT NULL  
);  
CREATE SUBSCRIPTION my_sub2 CONNECTION 'dbname=postgres user=replica  
host=192.168.0.171 port=5432' PUBLICATION my_pub;
```

`select pg_log_standby_snapshot();` - снимок выполняющихся транзакций или checkpoint, или дождаться, когда bgwriter запишет снимок.

```
pg_log_standby_snapshot
```

```
-----  
0/45026268
```

# Практика №3

Остановим postgresql.

Select slot\_name, slot\_type, active, inactive\_since from  
pg\_replication\_slots;

slot_name	slot_type	active	inactive_since
my_sub2	logical	f	2025-02-21 05:19:17.093171-05

Запустим postgresql.

# Практика №3

Выполним свитчвер (`rocky8_1` → `rocky8_2`)

Checkpoint; (мастер и затем реплика)

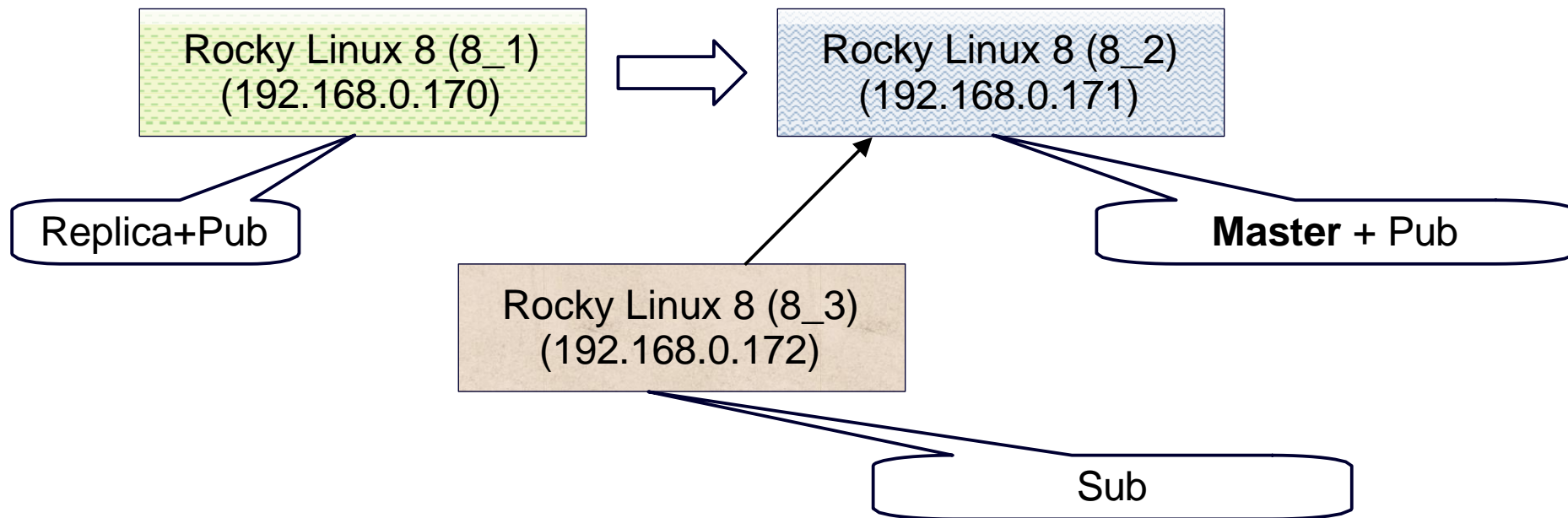
Проверка длительных активностей, лага репликации

Удаление и создание `standby.signal`


Перенастройка `primary_conninfo`

Останов `rocky_8_1`, подъём `rocky8_2`, подъём `rocky8_1`

# Практика №3.



# Практика №3



```
insert into emp(t) values ('bbb');
```



```
Table emp;
```

employee_id	t
1	aaa
34	bbb



# Практика №3

- Binary (с 16 версии)

# Практика №3

- Binary (с 16 версии)
- `CREATE SUBSCRIPTION test_sub CONNECTION 'host=127.0.0.1 dbname=postgres port=5432' PUBLICATION test_pub WITH (binary=true);`

# Практика №3

- Binary (с 16 версии)
- `CREATE SUBSCRIPTION test_sub CONNECTION 'host=127.0.0.1 dbname=postgres port=5432' PUBLICATION test_pub WITH (binary=true);`
- Обязательное совпадение типов данных (кроме текста).

# Практика №3

- Binary (с 16 версии)
- `CREATE SUBSCRIPTION test_sub CONNECTION 'host=127.0.0.1 dbname=postgres port=5432' PUBLICATION test_pub WITH (binary=true);`
- Обязательное совпадение типов данных (кроме текста).
- Ускорение инициализации до 40%

# Практика №3

- Streaming (до 16 версии) true/false

# Практика №3

- Streaming (с 16 версии) off/on/parallel

# Практика №3

- Streaming (с 16 версии) off/on/parallel
- Off – транзакция полностью декодируется и только потом передаётся.

# Практика №3

- Streaming (с 16 версии) off/on/parallel
- Off – транзакция полностью декодируется и только потом передаётся.
- On – запись во временные файлы и применение информации из них после подтверждения



# Практика №3

- Streaming (с 16 версии) off/on/parallel
- Off – транзакция полностью декодируется и только потом передаётся.
- On – запись во временные файлы и применение информации из них после подтверждения
- Parallel – рабочий параллельный процесс применяет изменения напрямую, если его нет, то так же работа через временные файлы.

# Практика №3. Итоги

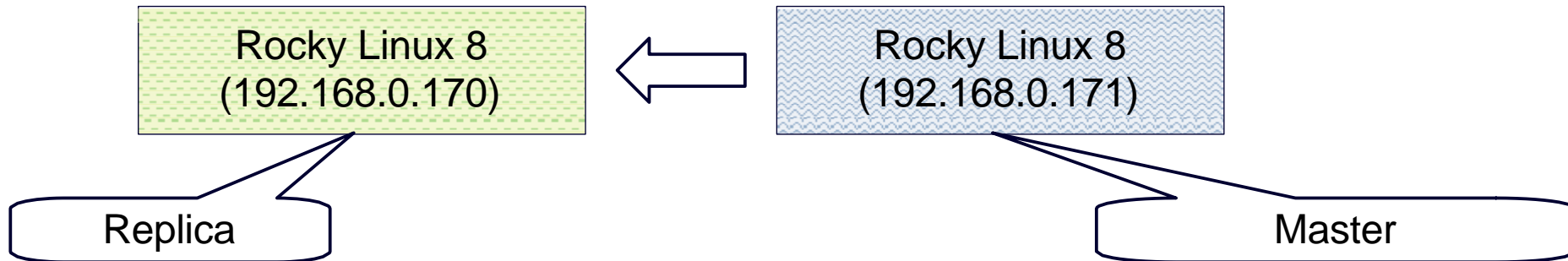
Изменение роли сервера, на который оформлена подписка, не влияет на работоспособность логической репликации

Binary в некоторых случаях может дать прирост производительности инициализации.

Streaming позволяет ускорить процесс логической репликации

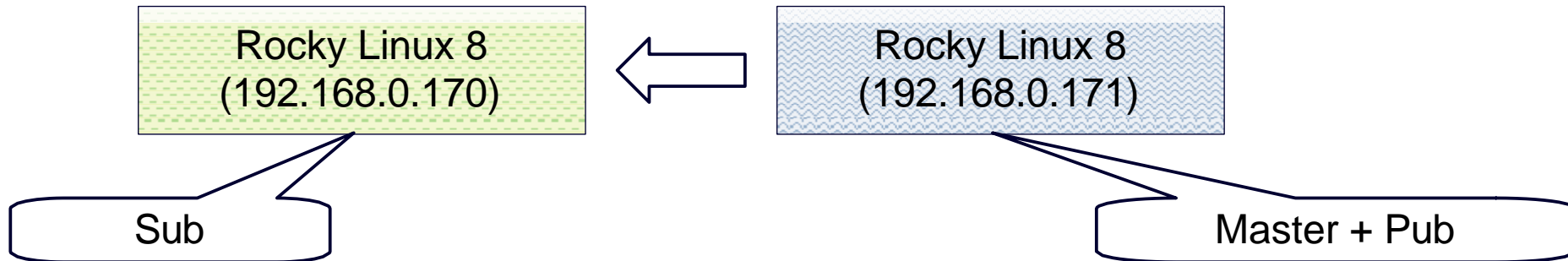
# Практика №4. Преобразования

Попробуем преобразовать физическую реплику в логическую.



# Практика №4.

Попробуем преобразовать физическую реплику в логическую.



# Практика №4.



Пропустим момент создания реплики. (см 34й слайд)



Проверяем наличие слота репликации:

```
select slot_name, slot_type, active from pg_replication_slots;
```

Создаём слот логической репликации:

```
select pg_create_logical_replication_slot('logical_slot', 'pgoutput');
```

Создаём публикацию:

```
CREATE PUBLICATION my_pub FOR ALL TABLES;
```

Не обязательный шаг:

```
Checkpoint;
```

# Практика №4.




Checkpoint;

```
Select pg_promote();  
Select pg_is_in_recovery();
```

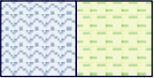
Идём в логи и ищем строку: redo done at **0/47004CB0**



```
select pg_replication_slot_advance('logical_slot', '0/47004CB0');
```



```
CREATE SUBSCRIPTION my_sub CONNECTION 'user=replica dbname=postgres  
host=192.168.0.171 port=5432' PUBLICATION my_pub WITH  
(copy_data=false, slot_name='logical_slot', create_slot=false);
```



Проверка работоспособности + удаление слота физической репликации, если он был.

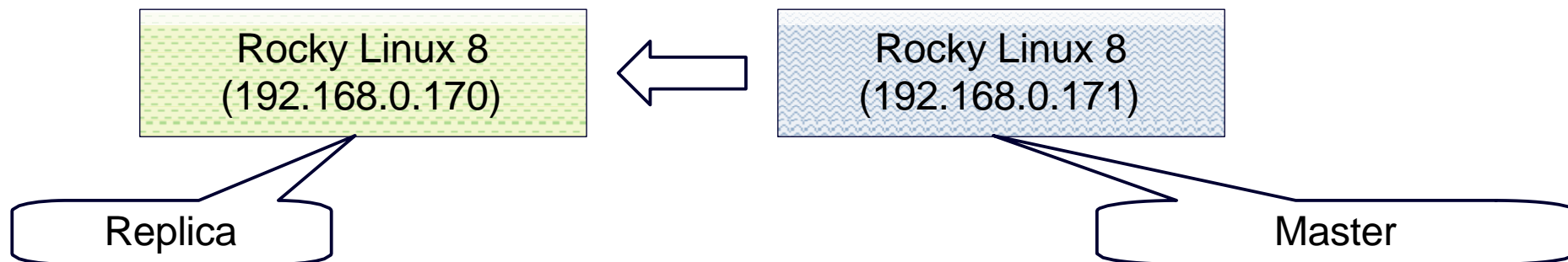
# Практика №4.

Вроде бы всё?

# Практика №4.

Ну конечно нет :)

17 PostgreSQL – утилита `pg_createsubscriber`





# Практика №4.

Останавливаем PostgreSQL

```
pg_createsubscriber -D /var/lib/pgsql/17/data  
--publisher-server='user=postgres dbname=postgres host=192.168.0.171  
port=5432'  
--database postgres  
--subscriber-username=postgres  
--replication-slot=postgres_db_slot  
--publication=postgres_db_publication  
--subscription=postgres_db_subscription  
--verbose
```

Запускаем PostgreSQL

# Практика №4.

Проверки, которые необходимо выполнить перед изменением типа репликации:

1) `select name , setting from pg_settings where name in ('wal_level','max_replication_slots','max_wal_senders');`

2) `select name , setting from pg_settings where name in ('max_worker_processes','max_replication_slots','max_logical_replication_workers');`

# Практика №4. Итоги

Изменение типа репликации из физической в логическую возможно.

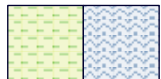
В 17 версии эта процедура значительно упростилась за счёт появления отдельной утилиты `pg_createsubscriber`.

# Практика №5. Подводные камни

Rocky Linux 8 (8\_1)  
(192.168.0.170)

Rocky Linux 8 (8\_2)  
(192.168.0.171)

# Практика №5.



```
create table fff(i integer);  
insert into fff values (3),(3),(5);
```

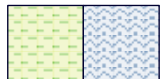
Создаём публикацию, создаём таблицу и подписку.



```
select ctid, * from fff;
```

ctid	i
(0,1)	3
(0,2)	3
(0,3)	5

# Практика №5.



```
create table fff(i integer);
```

```
insert into fff values (3),(3),(5);
```

Создаём публикацию, создаём таблицу и подписку.

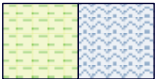
```
select ctid, * from fff;
```

ctid	i
(0,1)	3
(0,2)	3
(0,3)	5

```
delete from fff where ctid = '(0,2)';
```



# Практика №5.



```
create table fff(i integer);  
insert into fff values (3),(3),(5);
```

Создаём публикацию, создаём таблицу и подписку.



```
select ctid, * from fff;
```

ctid	i
(0,1)	3
(0,2)	3
(0,3)	5

```
delete from fff where ctid = '(0,2)';
```

*ERROR: cannot delete from table "fff"  
because it does not have a replica identity  
and publishes deletes*

*HINT: To enable deleting from the table, set  
REPLICA IDENTITY using ALTER TABLE.*

# Практика №5.

<https://www.postgresql.org/docs/17/sql-altertable.html>

```
REPLICA IDENTITY { DEFAULT | USING INDEX имя_индекса | FULL | NOTHING }
```

USING INDEX index\_name

Records the old values of the columns covered by the named index, that must be unique, not partial, not deferrable, and include only columns marked NOT NULL. If this index is dropped, the behavior is the same as NOTHING.



# Практика №5.

```
alter table fff replica identity full;
```

```
delete from fff where ctid = '(0,2)';
```

```
postgres=# select ctid, * from fff;
```

ctid	i
(0,1)	3
(0,3)	5

```
postgres=# select ctid, * from fff;
```

ctid	i
(0,2)	3
(0,3)	5

# Практика №5.

[https://github.com/dataegret/pg-utils/blob/master/sql/check\\_all\\_tables\\_have\\_pk.sql](https://github.com/dataegret/pg-utils/blob/master/sql/check_all_tables_have_pk.sql)



# Практика №5.

А может быть создать РК?

# Практика №5.

```
ALTER TABLE your_table ADD PRIMARY KEY (column_name);
```

# Практика №5.

Есть и более интересные советы:

```
create unique index concurrently tbl_name_column_idx on  
tbl_name(column);
```

```
alter table tbl_name add constraint tbl_name_column_pkey primary key  
using index tbl_name_column_idx;
```

# Практика №5.

Но, что делать, если у нас в таблице нет поля или комбинации полей, которые мы можем принять за уникальный индекс?

# Практика №5.

[https://github.com/Nikitin-Alexandr/utils/blob/main/create\\_PK.sql](https://github.com/Nikitin-Alexandr/utils/blob/main/create_PK.sql)



# Практика №5.

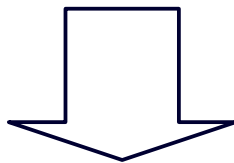
```
\set schema_name public  
\set tbl_name largetable  
\set new_colname id  
\set seq_name largetable_seq_id
```





# Практика №5.

```
\set schema_name public  
\set tbl_name largetable  
\set new_colname id  
\set seq_name largetable_seq_id
```



create\_PK\_largetable.sql



# Практика №5.

```
alter table public.largetable add column id bigint;  
create sequence public.largetable_seq_id as bigint owned by largetable.id;  
alter table public.largetable alter column id set default nextval($$largetable_seq_id$$);  
alter sequence public.largetable_seq_id owner to postgres;
```

```
set lock_timeout to '100ms';  
set session_replication_role to 'replica';  
set deadlock_timeout to '600s';  
set vacuum_cost_delay to 0;  
select now() as start_time \gset  
\set cnt_err_vac 0
```

```
update public.largetable set id = nextval($$largetable_seq_id$$) where id is null and ctid  
>='(0,0)' and ctid<'(1000,0)';  
update public.largetable set id = nextval($$largetable_seq_id$$) where id is null and ctid  
>='(1000,0)' and ctid<'(2000,0)';  
update public.largetable set id = nextval($$largetable_seq_id$$) where id is null and ctid  
>='(2000,0)' and ctid<'(3000,0)';  
...
```

# Практика №5.

```
set statement_timeout to '100ms';
alter table public.targetable add constraint targetable_95984 check (id is not null)
not valid;
reset statement_timeout;
alter table public.targetable validate constraint targetable_95984;
set statement_timeout to '100ms';
alter table public.targetable alter column id set not null;
alter table public.targetable drop constraint targetable_95984;
reset statement_timeout;
create unique index concurrently targetable_id_idx on public.targetable(id);
set statement_timeout to '100ms';
alter table public.targetable add constraint targetable_id_pkey primary key using
index targetable_id_idx;
reset statement_timeout;
\d public.targetable
```

# Практика №5. Итоги

При работе с логической репликацией вы можете столкнуться с некоторыми сложностями.

Но, теперь вы знаете, как с ними поступить :)

# Выводы

1) Логическая репликация это инструмент, который может помочь вам в каких-то случаях.

# Выводы

- 1) Логическая репликация это инструмент, который может помочь вам в каких-то случаях.
- 2) Этот инструмент развивается и пользоваться им становится всё удобнее.

# Выводы



Чем ты занимаешься после работы?



Я PostgreSQL DBA...



- 1) Логическая репликация – это инструмент, который может помочь вам в каких-то случаях.
- 2) Этот инструмент развивается и пользоваться им становится всё удобнее.
- 3) Если вам после выступления захотелось попробовать всё, о чём я рассказал и что-то не получится – обратитесь к вашим ДБА :)

# DBA Team

**Ваши данные - наша забота!**

- **Поддержка PostgreSQL 24/7**
- **Настройка кластеров конкретно под ваш профиль нагрузки;**
- **Помощь в оптимизации SQL-запросов;**
- **Выполнение миграций данных без простоя системы;**
- **Миграция БД на новые сервера или в другой ДЦ с минимальным простоем;**
- **Своевременное обновление PostgreSQL (как минорное, так и мажорное);**
- **Настройка резервного копирования и восстановления;**
- **Консультации по архитектуре и производительности БД;**
- **Проведение регулярных проверок ваших кластеров;**





**PG BootCamp Russia 2025 Ekaterinburg**

[PGBootCamp.ru](https://PGBootCamp.ru)

**Вопросы?**

Александр Никитин

E-mail: [contact@dba.team](mailto:contact@dba.team)

tg: @anikitindba

Бонус: знаем команду DevOps.