Репликация
# Логическая репликация



Авторы: Егор Рогов, Павел Лузанов, Илья Баштанов



**Обратная связь**
Отзывы, замечания и предложения направляйте по адресу:
edu@postgrespro.ru

# Темы

Отличия логической репликации от физической

Публикации и подписки

Логическое декодирование и слоты логической репликации

Конфликты и их разрешение

Выполнение триггеров на подписчике

## Сравнение

### Физическая

- мастер-реплика: поток данных только в одну сторону
- трансляция потока журнальных записей или файлов журнала
- требуется двоичная совместимость серверов
- возможна репликация только всего кластера

### Логическая

- публикация-подписки: у сервера нет выделенной роли
- трансляция изменений табличных строк
- необходим уровень журнала logical
- требуется совместимость на уровне протокола
- возможна выборочная репликация отдельных таблиц



Как мы помним, при физической репликации серверы имеют назначенные роли: мастер и реплика. Мастер передает на реплику журнальные записи (в виде файлов или потока записей); реплика применяет эти записи к своим файлам данных. Применение происходит чисто механически, без «понимания смысла» изменений, поэтому важна двоичная совместимость между серверами (одинаковые платформы и основные версии PostgreSQL). Поскольку журнал общий для всего кластера, то и реплицировать можно только кластер целиком.

При логической репликации на одном сервере создается публикация, другие серверы могут на нее подписаться. У сервера нет выделенной роли: один и тот же сервер может как публиковать изменения, так и подписываться на другие (или даже свои) публикации. Подписке передается информация об изменениях строк в таблицах в платформо-независимом виде; двоичная совместимость не требуется. Для работы логической репликации в журнале публикующего сервера необходима дополнительная информация (параметр *wal_level* = logical). Логическая репликация позволяет транслировать не все изменения, а только касающиеся определенных таблиц.

Логическая репликация доступна, начиная с версии 10; более ранние версии должны были использовать расширение pglogical, либо организовывать репликацию с помощью триггеров.
https://www.2ndquadrant.com/en/resources-old/pglogical/

# Модель

## Публикация

- объект базы данных
- выдает изменения данных построчно
- изменения в порядке фиксации транзакций

## Подписка

- подписывается, получает и применяет изменения
- таблицы и столбцы сопоставляются по полным именам, строки — по логическим идентификаторам
- поддерживается «бесшовная» начальная синхронизация (экспорт снимка)
- могут возникать конфликты с локальными данными



Логическая репликация использует модель «публикация-подписка».

На одном сервере создается *публикация,* которая может включать ряд таблиц одной базы данных. Для репликации из нескольких баз данных потребуется создать несколько публикаций.

Публикация включает в себя изменения, происходящие с таблицами: эти изменения передаются на уровне строк («в таблице такой-то такая-то строка изменилась таким-то образом»).

Изменения выдаются не сразу, а только при фиксации транзакции.

https://postgrespro.ru/docs/postgresql/13/logical-replication-publication

Другие серверы могут создавать *подписки* на публикации, получать и применять изменения.

Применение изменений всегда происходит построчно. Хотя каждое изменение не требует разбора и планирования запроса, массовые изменения из-за этого будут выполняться медленнее.

Таблицы идентифицируются по полным именам (включая схему), столбцы также идентифицируются по именам. Это позволяет подписке использовать отличающуюся схему данных (например, иметь в таблице дополнительные столбцы).

По умолчанию при создании подписки выполняется начальная синхронизация содержимого таблиц. Она происходит «бесшовно» благодаря использованию механизма экспорта снимка данных.

https://postgrespro.ru/docs/postgresql/13/logical-replication-subscription

# Ограничения

Реплицируются не все изменения

- только команды INSERT, UPDATE, DELETE, TRUNCATE
- только базовые и секционированные таблицы
  (не реплицируются последовательности, материализованные представления)

Подписку и публикацию можно создать только на основном сервере

- не работают на физических репликах

Циклы в репликации не обрабатываются

- нельзя реплицировать одну и ту же таблицу
  с одного сервера на другой и обратно



Реплицируются только изменения содержимого таблиц, вызванные командами DML. TRUNCATE реплицируется, начиная с версии 11, секционированные таблицы — с версии 13.

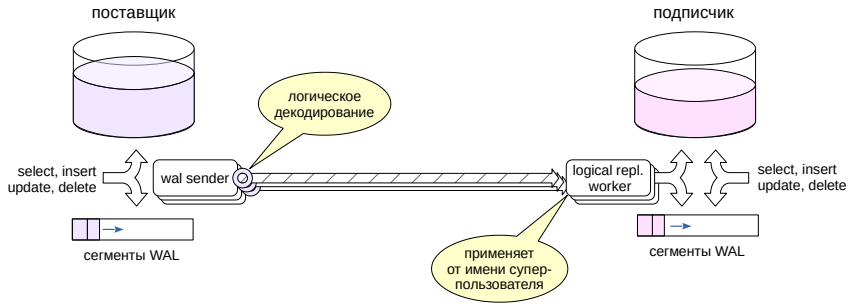
Не реплицируются команды DDL, что означает необходимость предварительно создать все необходимые таблицы на стороне подписки. Не реплицируются остальные объекты, объединяемые термином relation: последовательности, материализованные представления, внешние таблицы. Большие объекты (large objects) также не реплицируются.

Если используется физическая репликация, то и публикацию, и подписку можно создать только на основном сервере, так как команды DDL на реплике не поддерживаются.

Нет возможности организовать репликацию одной и той же таблицы между двумя серверами: изменения, сделанные на первом сервере, применяются вторым и тут же снова пересылаются первому, который скорее всего не сможет их применить из-за нарушения ограничений целостности.

https://postgrespro.ru/docs/postgresql/13/logical-replication-restrictions

# Схема работы




*max_wal_senders*
*max_replication_slots*

*max_logical_replication_workers*
*max_worker_processes*



Данные об изменениях таблиц передаются подписке тем же процессом wal sender, что и при обычной потоковой репликации. Так же, как и при потоковой репликации, этот процесс читает журнал предзаписи, но не просто транслирует прочитанные записи, а предварительно декодирует их. В отличие от физической репликации, в обязательном порядке используется слот логической репликации.

На стороне подписки информацию принимает фоновый процесс logical replication worker и применяет ее. В это же время сервер-подписчик принимает обычные запросы и на чтение, и на запись.

Обратите внимание, что на публикующем сервере может быть запущено много процессов wal sender — по одному на каждую подписку. Значения параметров *max_wal_senders* и *max_replication_slots* должны соответствовать нужному количеству процессов.

На сервере подписки необходимо установить параметры *max_logical_replication_workers* (для процессов, принимающих изменения по подписке) и в целом *max_worker_processes* (как минимум на единицу больше, так как есть еще процесс logical replication launcher, но вообще этот пул используется и для других нужд).

https://postgrespro.ru/docs/postgresql/13/logical-replication-architecture

### Логическая репликация

Мы собираемся настроить логическую репликацию таблицы test с сервера alpha на сервер beta.

Для начала создадим базу данных.

```
α=> CREATE DATABASE replica_logical;
```

```
CREATE DATABASE
```

```
α=> \c replica_logical
```

```
You are now connected to database "replica_logical" as user "student".
```

Второй кластер изначально будет копией первого, поэтому выполним резервное копирование в каталог PGDATA второго сервера.

```
student$ pg_basebackup --pgdata=/home/student/backup
```

```
student$ sudo pg_ctlcluster 13 beta status
```

```
Error: /var/lib/postgresql/13/beta is not accessible or does not exist
```

```
student$ sudo rm -rf /var/lib/postgresql/13/beta
```

```
student$ sudo mv /home/student/backup /var/lib/postgresql/13/beta
```

```
student$ sudo chown -R postgres:postgres /var/lib/postgresql/13/beta
```

Запускаем второй сервер.

```
student$ sudo pg_ctlcluster 13 beta start
```

Теперь на первом сервере создадим таблицу и заполним ее данными.

```
α=> CREATE TABLE test(id int PRIMARY KEY GENERATED BY DEFAULT AS IDENTITY, descr text);
```

```
CREATE TABLE
```

```
α=> INSERT INTO test(descr) VALUES ('Раз'), ('Два'), ('Три');
```

```
INSERT 0 3
```

---

Для работы логической репликации понадобится изменить уровень журнала.

```
student$ psql -U postgres -c "ALTER SYSTEM SET wal_level = logical"
```

```
ALTER SYSTEM
```

```
student$ sudo pg_ctlcluster 13 alpha restart
```

На втором сервере таблицы test нет. Поскольку команды DDL не реплицируются, таблицу необходимо создать вручную. При этом таблица подписчика может содержать и дополнительные столбцы, если это необходимо.

```
student$ /usr/lib/postgresql/13/bin/psql -p 5433 -d replica_logical
```

```
β=> CREATE TABLE test(id int PRIMARY KEY GENERATED BY DEFAULT AS IDENTITY, descr text, additional text);
```

```
CREATE TABLE
```

На первом сервере создаем публикацию для таблицы test. Публикация относится к конкретной базе данных; в нее можно включить и несколько таблиц, а можно даже все таблицы сразу (FOR ALL TABLES).

```
student$ /usr/lib/postgresql/13/bin/psql -d replica_logical
```

```
α=> CREATE PUBLICATION test_pub FOR TABLE test;
```

```
CREATE PUBLICATION
```

```
α=> \dRp+
```

```
                          Publication test_pub
  Owner  | All tables | Inserts | Updates | Deletes | Truncates | Via root
---------+------------+---------+---------+---------+-----------+----------
 student | f          | t       | t       | t       | t         | f
Tables:
    "public.test"
```

На втором сервере подписываемся на публикацию. При этом на публикующем сервере будет создан слот логической репликации.

Подписку может создать только суперпользователь. А роль для подключения к публикующему серверу должна

иметь атрибуты REPLICATION и LOGIN, и также право чтения публикуемых таблиц — роль student подходит под эти требования.

```
β=> \c - postgres

You are now connected to database "replica_logical" as user "postgres".

β=> CREATE SUBSCRIPTION test_sub
CONNECTION 'port=5432 user=student dbname=replica_logical'
PUBLICATION test_pub;

NOTICE:  created replication slot "test_sub" on publisher
CREATE SUBSCRIPTION

β=> \c - student

You are now connected to database "replica_logical" as user "student".

β=> \dRs

            List of subscriptions
   Name   |  Owner   | Enabled | Publication
----------+----------+---------+-------------
 test_sub | postgres | t       | {test_pub}
(1 row)
```

По умолчанию данные сначала синхронизируются между серверами, и только после этого запускается процесс репликации. Это выполняется «бесшовно» с гарантией того, что никакие изменения не будут потеряны.

```
β=> SELECT * FROM test;

 id | descr | additional
----+-------+------------
(0 rows)
```

Проверим, как работает репликация изменений.

```
α=> INSERT INTO test(descr) VALUES ('Четыре');

INSERT 0 1

β=> SELECT * FROM test;

 id | descr  | additional
----+--------+------------
  1 | Раз    |
  2 | Два    |
  3 | Три    |
  4 | Четыре |
(4 rows)
```

Состояние подписки можно посмотреть в представлении:

```
β=> SELECT * FROM pg_stat_subscription \gx

-[ RECORD 1 ]---------+------------------------------
subid                 | 32778
subname               | test_sub
pid                   | 79800
relid                 |
received_lsn          | 0/50282C0
last_msg_send_time    | 2022-06-13 16:06:58.43766+03
last_msg_receipt_time | 2022-06-13 16:06:58.438562+03
latest_end_lsn        | 0/50282C0
latest_end_time       | 2022-06-13 16:06:58.43766+03
```

- received_lsn — позиция в журнале, до которой получены изменения;
- latest_end_lsn — позиция в журнале, подтвержденная процессу wal sender.

К процессам сервера добавился logical replication worker (его номер указан в pg_stat_subscription.pid):

```
student$ ps -o pid,command --ppid `sudo head -n 1 /var/lib/postgresql/13/beta/postmaster.pid`

    PID COMMAND
  79508 postgres: 13/beta: checkpointer
  79509 postgres: 13/beta: background writer
  79510 postgres: 13/beta: walwriter
  79511 postgres: 13/beta: autovacuum launcher
  79512 postgres: 13/beta: stats collector
  79513 postgres: 13/beta: logical replication launcher
  79800 postgres: 13/beta: logical replication worker for subscription 32778
```

```
79821 postgres: 13/beta: student replica_logical [local] idle
```

# Логическое декодирование

Postgres PROFESSIONAL

## Переупорядочивающий буфер

wal sender читает журнальные записи и накапливает их в буфере, раскладывая по транзакциям

буфер в локальной памяти; при необходимости сбрасывается на диск

## Модуль вывода

получает накопленные записи при фиксации транзакции

декодирует записи, формируя сообщения об операциях над табличными строками в платформо-независимом формате

фильтрует сообщения, на которые подписан получатель

## Слот логической репликации

гарантирует, что подписка не пропустит изменения



Полезно представлять внутреннее устройство логической репликации.

Журнальные записи читаются процессом wal sender и раскладываются по отдельным транзакциям в специальном буфере в оперативной памяти. Это делается для того, чтобы при фиксации транзакции можно было взять все изменения, сделанные именно этой транзакцией, и передать их подписчику. При превышении определенного порога буфер начинает сбрасываться на диск (в каталог PGDATA/pg_replslots).

Заметим, что при наличии нескольких подписчиков и, следовательно, нескольких процессов wal sender, *каждый* из этих процессов будет самостоятельно читать WAL: буфер, упорядочивающий записи, находится в *локальной* памяти каждого процесса wal sender.

Когда транзакция фиксируется, ее изменения передаются *модулю вывода,* который *декодирует* их и представляет в платформо-независимом (текстовом) формате. Процесс wal sender передает эти декодированные сообщения подписчику (если он на них подписан) через *слот логической репликации.* Этот слот похож на обычный репликационный слот, но к нему привязан *модуль вывода.*

В журнал на уровне logical дополнительно записывается информация, необходимая для логического декодирования, в частности:
- новые значения всех столбцов для UPDATE, а не только измененных;
- старые значения столбцов, входящих в логический идентификатор, для UPDATE и DELETE;
- OID базы данных для COMMIT.

https://postgrespro.ru/docs/postgresql/13/logicaldecoding

### Слот и логическое декодирование

Что происходит при логическом декодировании?

Создадим вручную слот логической репликации. Для передачи изменений подписчику используется модуль вывода pgoutput, а для наблюдения удобнее модуль test_decoding:

```
α=> SELECT pg_create_logical_replication_slot('test_slot','test_decoding');

 pg_create_logical_replication_slot
------------------------------------
 (test_slot,0/50282F8)
(1 row)
```

Теперь на сервере alpha два слота:

```
α=> SELECT slot_name, plugin, slot_type, active FROM pg_replication_slots;

 slot_name |    plugin     | slot_type | active
-----------+---------------+-----------+--------
 test_sub  | pgoutput      | logical   | t
 test_slot | test_decoding | logical   | f
(2 rows)
```

В отдельном сеансе начнем транзакцию и вставим строку в таблицу:

```
student$ /usr/lib/postgresql/13/bin/psql -d replica_logical

α=> BEGIN;

BEGIN

α=> INSERT INTO test(descr) VALUES ('Пять');

INSERT 0 1
```

Модуль вывода может запросить у слота изменения:

```
α=> SELECT * FROM pg_logical_slot_get_changes('test_slot', NULL, NULL);

 lsn | xid | data
-----+-----+------
(0 rows)
```

Транзакция не завершена, модуль вывода пока ничего не получил. Сделаем еще изменение и завершим транзакцию.

```
α=> UPDATE test SET descr = 'Beş' WHERE id = 5;

UPDATE 1

α=> COMMIT;

COMMIT

α=> SELECT * FROM pg_logical_slot_get_changes('test_slot', NULL, NULL);

    lsn    | xid |                            data
-----------+-----+-----------------------------------------------------------------
 0/50282F8 | 492 | BEGIN 492
 0/50282F8 | 492 | table public.test: INSERT: id[integer]:5 descr[text]:'Пять'
 0/5028380 | 492 | table public.test: UPDATE: id[integer]:5 descr[text]:'Beş'
 0/5028400 | 492 | COMMIT 492
(4 rows)
```

Теперь модуль вывода получил изменения.

---

Удалим тестовый слот, иначе он будет препятствовать удалению сегментов WAL.

```
α=> SELECT pg_drop_replication_slot('test_slot');

 pg_drop_replication_slot
--------------------------

(1 row)
```

# Конфликты

Режимы идентификации для изменения и удаления

`ALTER TABLE ... REPLICA IDENTITY ...`

а) столбцы первичного ключа (по умолчанию)

б) столбцы указанного уникального индекса с ограничением NOT NULL

в) все столбцы

г) без идентификации (по умолчанию для системного каталога)

Конфликты — нарушение ограничений целостности

репликация приостанавливается до устранения конфликта

требуется вручную исправить данные на стороне подписки



Вставка новых строк на стороне подписки происходит достаточно просто.

Интереснее обстоит дело при изменениях и удалениях — в этом случае надо как-то идентифицировать старую версию строки. По умолчанию для этого используются столбцы первичного ключа, но для таблицы можно указать и другие способы: по уникальному индексу или по всем столбцам. В первом случае для поиска строки будет использоваться соответствующий индекс, во втором — полное сканирование таблицы (что крайне неэффективно для больших таблиц).

Можно вообще отказаться от поддержки репликации для некоторых таблиц (по умолчанию так работают таблицы системного каталога).

Поскольку таблицы на публикующем сервере и на подписчике могут изменяться независимо друг от друга, при вставке новых версий строк возможно возникновение конфликта — нарушение ограничения целостности. В этом случае процесс применения записей приостанавливается до тех пор, пока конфликт не будет разрешен. Автоматического разрешения пока не существует; нужно вручную исправить данные на подписчике так, чтобы устранить конфликт.

https://postgrespro.ru/docs/postgresql/13/logical-replication-conflicts

### Конфликты

Заметим, что последовательности не реплицируются. На втором сервере создалась своя собственная последовательность:

```
β=> INSERT INTO test(descr) VALUES ('Шесть - локально');
```

```
ERROR:  duplicate key value violates unique constraint "test_pkey"
DETAIL:  Key (id)=(1) already exists.
```

А вот так получится:

```
β=> INSERT INTO test VALUES (6, 'Шесть - локально');
```

```
INSERT 0 1
```

Что произойдет, если значение с таким же ключом (6) появится на публикующем сервере?

```
α=> INSERT INTO test(descr) VALUES ('Шесть');
```

```
INSERT 0 1
```

```
α=> INSERT INTO test(descr) VALUES ('Семь');
```

```
INSERT 0 1
```

При репликации возникнет конфликт, и она будет приостановлена.

```
β=> SELECT * FROM test;
```

```
 id |      descr       | additional
----+------------------+------------
  1 | Раз              |
  2 | Два              |
  3 | Три              |
  4 | Четыре           |
  5 | Beş              |
  6 | Шесть - локально |
(6 rows)
```

Фактически, процесс logical replication worker будет периодически перезапускаться, проверяя, не устранен ли конфликт.
Поэтому информация в pg_stat_subscription пропадает:

```
β=> SELECT * FROM pg_stat_subscription \gx
```

```
-[ RECORD 1 ]---------+---------
subid                 | 32778
subname               | test_sub
pid                   |
relid                 |
received_lsn          |
last_msg_send_time    |
last_msg_receipt_time |
latest_end_lsn        |
latest_end_time       |
```

В журнал сообщений будут попадать записи о нарушении ограничений целостности:

```
student$ sudo tail -n 3 /var/log/postgresql/postgresql-13-beta.log
```

```
2022-06-13 16:07:00.396 MSK [79800] ERROR:  duplicate key value violates unique constraint "test_pkey"
2022-06-13 16:07:00.396 MSK [79800] DETAIL:  Key (id)=(6) already exists.
2022-06-13 16:07:00.400 MSK [79506] LOG:  background worker "logical replication worker" (PID 79800) exited with exit code 1
```

Чтобы разрешить этот конфликт, удалим конфликтующую строку из таблицы и немного подождем:

```
β=> DELETE FROM test WHERE id = 6;
```

```
DELETE 1
```

Проверим:

```
β=> SELECT * FROM test;
```

```
 id | descr  | additional
----+--------+------------
  1 | Раз    |
  2 | Два    |
  3 | Три    |
  4 | Четыре |
  5 | Beş    |
  6 | Шесть  |
  7 | Семь   |
(7 rows)
```

Данные появились, репликация восстановлена.

---

### Триггеры на подписчике

На подписчике могут выполняться триггеры, но если просто создать триггер, то он не отработает. Это удобно, если на обоих серверах созданы одинаковые таблицы с одинаковым набором триггеров: в таком случае триггер уже отработал на публикующем сервере, его не надо выполнять на подписчике.

Попробуем.

```
β=> CREATE OR REPLACE FUNCTION change_descr() RETURNS trigger AS $$
BEGIN
  NEW.additional := 'из публикации';
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

```
CREATE FUNCTION
```

```
β=> CREATE TRIGGER test_before_row
BEFORE INSERT OR UPDATE ON test
FOR EACH ROW
EXECUTE FUNCTION change_descr();
```

```
CREATE TRIGGER
```

```
α=> INSERT INTO test(descr) VALUES ('Восемь');
```

```
INSERT 0 1
```

```
β=> SELECT * FROM test;
```

```
 id | descr  | additional
----+--------+------------
  1 | Раз    |
  2 | Два    |
  3 | Три    |
  4 | Четыре |
  5 | Beş    |
  6 | Шесть  |
  7 | Семь   |
  8 | Восемь |
(8 rows)
```

Можно изменить таблицу, чтобы триггер срабатывал только при репликации:

```
β=> ALTER TABLE test ENABLE REPLICA TRIGGER test_before_row;
```

```
ALTER TABLE
```

Или в обоих случаях: и при репликации, и при локальных изменениях.

```
β=> ALTER TABLE test ENABLE ALWAYS TRIGGER test_before_row;
```

```
ALTER TABLE
```

Различить эти ситуации можно с помощью параметра session_replication_role:

```
β=> CREATE OR REPLACE FUNCTION change_descr() RETURNS trigger AS $$
BEGIN
  NEW.additional := current_setting('session_replication_role');
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

```
CREATE FUNCTION
```

Добавляем по одной строке на каждом сервере:

```
α=> INSERT INTO test(descr) VALUES ('Девять');
```

```
INSERT 0 1
```

```
β=> INSERT INTO test(id,descr) VALUES (10,'Десять');
```

```
INSERT 0 1
```

```
β=> SELECT * FROM test;
```

```
 id | descr  | additional
----+--------+------------
  1 | Раз    |
  2 | Два    |
  3 | Три    |
  4 | Четыре |
  5 | Beş    |
  6 | Шесть  |
  7 | Семь   |
  8 | Восемь |
  9 | Девять | replica
 10 | Десять | origin
(10 rows)
```

Триггер сработал в обоих случаях, причем понятно, откуда пришла строка.

---

### Удаление подписки

Если репликация больше не нужна, надо удалить подписку — иначе на публикующем сервере останется открытым репликационный слот.

```
β=> \c - postgres
```

```
You are now connected to database "replica_logical" as user "postgres".
```

```
β=> DROP SUBSCRIPTION test_sub;
```

```
NOTICE:  dropped replication slot "test_sub" on publisher
DROP SUBSCRIPTION
```

# Итоги

Логическая репликация: модель «публикация–подписка»

Передаются изменения табличных строк

Возможна выборочная репликация отдельных таблиц

Не требуется двоичная совместимость серверов

# Практика

1. Создайте две базы данных на одном сервере.

   В первой базе данных создайте таблицу с первичным ключом и добавьте в нее несколько строк.
2. Перенесите определение созданной таблицы во вторую базу данных с помощью логической резервной копии.
3. Настройте логическую репликацию таблицы из первой базы данных во вторую.
4. Проверьте работу репликации.
5. Удалите подписку.



2. Воспользуйтесь утилитой pg_dump с ключом --schema-only.

3. Если попробовать выполнить это обычным образом, команда создания подписки «повиснет» из-за того, что она должна дождаться завершения активных транзакций на публикующем сервере, то есть и самой себя в том числе. В таком случае необходимо заранее создать слот логической репликации, как описано в документации: https://postgrespro.ru/docs/postgresql/13/sql-createsubscription

## 1. Базы данных и таблица

Сначала установим уровень журнала logical.

```
α=> \c - postgres
```

```
You are now connected to database "student" as user "postgres".
```

```
α=> ALTER SYSTEM SET wal_level = logical;
```

```
ALTER SYSTEM
```

```
student$ sudo pg_ctlcluster 13 alpha restart
```

Базы данных:

```
student$ /usr/lib/postgresql/13/bin/psql
```

```
α=> CREATE DATABASE replica_logical_1;
```

```
CREATE DATABASE
```

```
α=> CREATE DATABASE replica_logical_2;
```

```
CREATE DATABASE
```

Таблица в первой базе:

```
α=> \c replica_logical_1
```

```
You are now connected to database "replica_logical_1" as user "student".
```

```
α=> CREATE TABLE test(id int PRIMARY KEY GENERATED BY DEFAULT AS IDENTITY, descr text);
```

```
CREATE TABLE
```

```
α=> INSERT INTO test(descr) VALUES ('Раз'), ('Два'), ('Три');
```

```
INSERT 0 3
```

## 2. Перенос таблицы во вторую БД

Воспользоваться логической резервной копией особенно удобно, когда таблиц много.

```
student$ pg_dump --schema-only replica_logical_1 | psql -d replica_logical_2
```

```
SET
SET
SET
SET
SET
 set_config
------------

(1 row)

SET
SET
SET
SET
SET
SET
CREATE TABLE
ALTER TABLE
ALTER TABLE
ALTER TABLE
```

Переносить данные с помощью pg_dump не имеет смысла, поскольку в процессе переноса таблицы могут измениться.

```
student$ /usr/lib/postgresql/13/bin/psql -d replica_logical_2
```

```
α=> \d test;
```

```
                            Table "public.test"
 Column |  Type   | Collation | Nullable |             Default
--------+---------+-----------+----------+---------------------------------
 id     | integer |           | not null | generated by default as identity
 descr  | text    |           |          |
Indexes:
    "test_pkey" PRIMARY KEY, btree (id)
```

### 3. Логическая репликация

Публикация:

```
α=> CREATE PUBLICATION test_pub FOR TABLE test;
```

```
CREATE PUBLICATION
```

Поскольку репликация будет настроена на одном и том же сервере, вначале вручную создаем слот логической репликации.

```
α=> SELECT pg_create_logical_replication_slot('testslot','pgoutput');
```

```
 pg_create_logical_replication_slot
------------------------------------
 (testslot,0/3044D20)
(1 row)
```

И затем создаем подписку:

```
α=> \c - postgres
```

```
You are now connected to database "replica_logical_2" as user "postgres".
```

```
α=> CREATE SUBSCRIPTION test_sub
CONNECTION 'user=student dbname=replica_logical_1'
PUBLICATION test_pub
WITH (create_slot = false, slot_name = testslot);
```

```
CREATE SUBSCRIPTION
```

### 4. Проверка

```
α=> INSERT INTO test(descr) VALUES ('Четыре');
```

```
INSERT 0 1
```

```
α=> SELECT * FROM test;
```

```
 id | descr
----+--------
  1 | Раз
  2 | Два
  3 | Три
  4 | Четыре
(4 rows)
```

### 5. Удаление подписки

```
α=> DROP SUBSCRIPTION test_sub;
```

```
NOTICE:  dropped replication slot "testslot" on publisher
DROP SUBSCRIPTION
```