Репликация

Обзор логической репликации

Авторы: Егор Рогов, Павел Лузанов, Илья Баштанов, Алексей Береснев
Фото: Олег Бартунов (монастырь Пху и пик Бхрикути, Непал)

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

Темы

Логическая репликация

Уровни журнала

Публикации и подписки

Обнаружение и разрешение конфликтов

Варианты использования реплики

Логическая репликация

Публикующий сервер

- читает собственные журнальные записи
- декодирует их в изменения строк данных
- отправляет подписчикам

Сервер-подписчик

- получает записи из потока репликации
- применяет изменения к своим таблицам

Особенности

- публикация-подписка: поток данных возможен в обе стороны
- требуется совместимость на уровне протокола
- возможна выборочная репликация отдельных таблиц

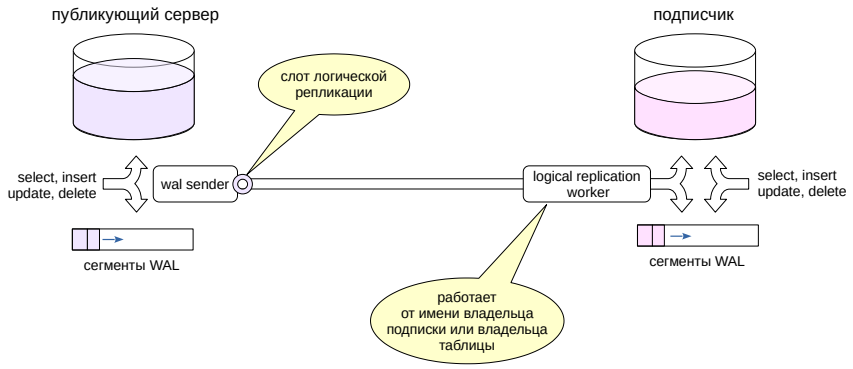
При физической репликации только один сервер (мастер) выполняет изменения и генерирует журнальные записи. Остальные серверы (реплики) только читают журнальные записи мастера и применяют их.

При логической репликации все серверы работают в обычном режиме, могут изменять данные и генерировать собственные журнальные записи. Любой сервер может *публиковать* свои изменения, а другие — *подписываться* на них. При этом один и тот же сервер может как публиковать изменения, так и подписываться на другие. Это позволяет организовать произвольные потоки данных между серверами.

Публикующий сервер читает собственные журнальные записи, но пересылает их подписчикам не в исходном виде (как при физической репликации), а предварительно декодирует их в «логический», независимый от платформы и версии PostgreSQL вид. Поэтому для логической репликации обычно не нужна двоичная совместимость, реплика должна лишь понимать протокол репликации. Также можно получать и применять изменения не всех объектов, а только отдельных таблиц.

https://postgrespro.ru/docs/postgresql/16/logical-replication

Логическая репликация



На публикующем сервере процесс wal_sender формирует журнальные записи, отражающие изменения в публикуемых данных, а фоновый процесс logical replication worker на сервере-подписчике получает информацию от публикующего сервера и применяет ее.

Чтобы гарантировать надежность передачи (отсутствие потерь и повторов), в обязательном порядке используется *слот логической репликации* (похожий на слот физической репликации).

https://postgrespro.ru/docs/postgresql/16/logical-replication-publication

https://postgrespro.ru/docs/postgresql/16/logical-replication-subscription

Публикации и подписки

Публикация

- включает одну или несколько таблиц базы данных
- обрабатывает команды INSERT, UPDATE, DELETE, TRUNCATE
- выдает изменения построчно после фиксации транзакции
- использует слот логической репликации

Подписка

- получает и применяет изменения
- возможна начальная синхронизация
- без разбора, переписывания и планирования — сразу выполнение
- возможны конфликты с локальными данными

Логическая репликация использует модель «публикация-подписка».

На одном сервере создается *публикация,* которая может включать ряд таблиц одной базы данных. Другие серверы могут *подписываться* на эту публикацию: получать и применять изменения.

Реплицируются только измененные строки таблиц (а не команды SQL). Команды DDL не передаются; таблицы-приемники на подписчике надо создавать вручную. Но есть возможность автоматической начальной синхронизации содержимого таблиц при создании подписки.

После фиксации транзакции информация о строках, которые она изменила, извлекается из записей WAL на стороне публикации – эта процедура называется *логическим декодированием*. Сформированные сообщения пересылаются подписке по протоколу репликации в формате, независимом от платформы и версии сервера.

Для каждой подписки на ее стороне запускается процесс logical replication worker, он применяет полученные сообщения. При этом не требуется выполнять разбор, переписывание и планирование, поэтому применение происходит эффективно.

Уровни журнала

Параметр *wal_level*

minimal	<	replica	<	logical
восстановление после сбоя		восстановление после сбоя		восстановление после сбоя
		восстановление из резервной копии, репликация		восстановление из резервной копии, репликация
				логическая репликация

Чтобы публикующий сервер смог сформировать сообщения об изменениях на уровне табличных строк, в журнал необходимо помещать дополнительную информацию, в частности идентификаторы затрагиваемых при репликации строк и сообщения о фактах изменения определений таблиц и типов данных. Это позволяет подписчику в любой момент знать структуру объектов, участвующих в репликации.

Такой расширенный уровень журнала называется **logical**. Поскольку по умолчанию используется уровень replica, для логической репликации потребуется изменить значение параметра *wal_level* на публикующем сервере.

https://postgrespro.ru/docs/postgresql/16/protocol-logical-replication#PROTOCOL-LOGICAL-MESSAGES-FLOW

Логическая репликация

Пусть на первом сервере имеется таблица:

```
=> CREATE DATABASE replica_overview_logical_dba;

CREATE DATABASE

=> \c replica_overview_logical_dba

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

=> CREATE TABLE test(id integer PRIMARY KEY, descr text);

CREATE TABLE
```

Склонируем кластер с помощью автономной копии, как мы делали в теме «Обзор физической репликации», но команде pg_basebackup не будем передавать ключ -R, поскольку нам потребуется независимый сервер, а не реплика.

```
student$ rm -rf /home/student/tmp/backup

student$ pg_basebackup --pgdata=/home/student/tmp/backup --checkpoint=fast
```

Если второй сервер работает, остановим его.

```
student$ sudo pg_ctlcluster 16 replica stop

Cluster is not running.
```

Перемещаем резервную копию в каталог данных второго сервера, поменяв владельца файлов:

```
student$ sudo rm -rf /var/lib/postgresql/16/replica

student$ sudo mv /home/student/tmp/backup /var/lib/postgresql/16/replica

student$ sudo chown -R postgres: /var/lib/postgresql/16/replica
```

Запускаем второй сервер:

```
student$ sudo pg_ctlcluster 16 replica start
```

Получили два независимых сервера, на каждом из них есть пустая таблица test. Добавим в таблицу на первом сервере пару строк:

```
=> INSERT INTO test VALUES (1, 'Раз'), (2, 'Два');

INSERT 0 2
```

Теперь мы хотим настроить между серверами логическую репликацию. Для этого понадобится дополнительная информация в журнале публикующего сервера:

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

ALTER SYSTEM

student$ sudo pg_ctlcluster 16 main restart
```

На первом сервере создаем публикацию:

```
student$ psql -d replica_overview_logical_dba

=> CREATE PUBLICATION test_pub FOR TABLE test;

CREATE PUBLICATION

=> \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 эту публикацию:

```
student$ psql -p 5433 -d replica_overview_logical_dba
```

```
=> CREATE SUBSCRIPTION test_sub
CONNECTION 'port=5432 user=student dbname=replica_overview_logical_dba'
PUBLICATION test_pub;
```

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

```
=> \dRs
```

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

Проверяем репликацию:

```
=> INSERT INTO test VALUES (3, 'Три');
```

```
INSERT 0 1
```

```
=> SELECT * FROM test;
```

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

Состояние подписки можно посмотреть в представлении:

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

```
-[ RECORD 1 ]---------+------------------------------
subid                 | 24578
subname               | test_sub
pid                   | 81558
leader_pid            |
relid                 |
received_lsn          | 0/30042B8
last_msg_send_time    | 2024-07-08 14:57:38.382977+03
last_msg_receipt_time | 2024-07-08 14:57:38.383807+03
latest_end_lsn        | 0/30042B8
latest_end_time       | 2024-07-08 14:57:38.382977+03
```

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

```
student$ ps -o pid,command --ppid 81264
```

```
    PID COMMAND
  81265 postgres: 16/replica: checkpointer
  81266 postgres: 16/replica: background writer
  81269 postgres: 16/replica: walwriter
  81270 postgres: 16/replica: autovacuum launcher
  81271 postgres: 16/replica: logical replication launcher
  81525 postgres: 16/replica: student replica_overview_logical_dba [local] idle
  81558 postgres: 16/replica: logical replication apply worker for subscription 24578
```

Конфликты

Режимы идентификации для изменения и удаления

- столбцы первичного ключа (по умолчанию)
- столбцы указанного уникального индекса с ограничением NOT NULL
- все столбцы
- без идентификации (по умолчанию для системного каталога)

Конфликты — нарушение ограничений целостности

- репликация приостанавливается до устранения конфликта вручную

Вставка новых строк 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 при определении таблицы можно указать и другие способы (replica identity): использовать уникальный индекс или использовать все столбцы. Можно вообще отказаться от поддержки репликации для некоторых таблиц (по умолчанию — таблицы системного каталога).

Поскольку таблицы на сервере публикации и сервере подписки могут изменяться независимо друг от друга, при вставке новых версий строк возможны конфликты — нарушения ограничений целостности. В этом случае процесс применения записей приостанавливается до тех пор, пока конфликт не будет разрешен вручную.

Конфликты

Локальные изменения на подписчике не запрещаются. Вставим строку в таблицу на втором сервере:

```
=> INSERT INTO test VALUES (4, 'Четыре (локально)');
```

```
INSERT 0 1
```

Если теперь строку с таким же значением первичного ключа вставить на публикующем сервере, при ее применении на стороне подписки произойдет конфликт.

```
=> INSERT INTO test VALUES (4, 'Четыре');
```

```
INSERT 0 1
```

```
=> INSERT INTO test VALUES (5, 'Пять');
```

```
INSERT 0 1
```

Подписка не может применить изменение, репликация остановилась.

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

```
-[ RECORD 1 ]---------+---------
subid                 | 24578
subname               | test_sub
pid                   |
leader_pid            |
relid                 |
received_lsn          |
last_msg_send_time    |
last_msg_receipt_time |
latest_end_lsn        |
latest_end_time       |
```

```
=> SELECT * FROM test;
```

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

Чтобы разрешить конфликт, удалим строку на втором сервере и немного подождем...

```
=> DELETE FROM test WHERE id=4;
```

```
DELETE 1
```

```
=> SELECT * FROM test;
```

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

Репликация возобновилась.

Удаление подписки

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

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

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

Ограничения

Postgres PROFESSIONAL

Не реплицируются

- команды DDL
- значения последовательностей
- изменения представлений, материализованных представлений, сторонних таблиц

Не поддерживается

- автоматическое разрешение конфликтов

Логическая репликация имеет довольно много ограничений.

Не реплицируются команды DDL — все изменения схемы данных надо переносить вручную.

Реплицироваться могут только обычные базовые таблицы и секционированные таблицы. То есть нельзя реплицировать отношения другого рода, такие как последовательности, материализованные представления и внешние таблицы.

И, как уже говорилось, нет возможности автоматического разрешения конфликтов.

Эти ограничения уменьшают применимость логической репликации.

https://postgrespro.ru/docs/postgresql/16/logical-replication-restrictions

Особенности

Ссылочная целостность

для TRUNCATE нужно включать в публикацию все таблицы, на которые ссылаются внешние ключи

Нужно постоянное соединение

неактивный слот препятствует удалению сегментов WAL и удерживает горизонт очистки

Могут вызвать проблемы

массовые изменения данных

изменения, сделанные долгими транзакциями

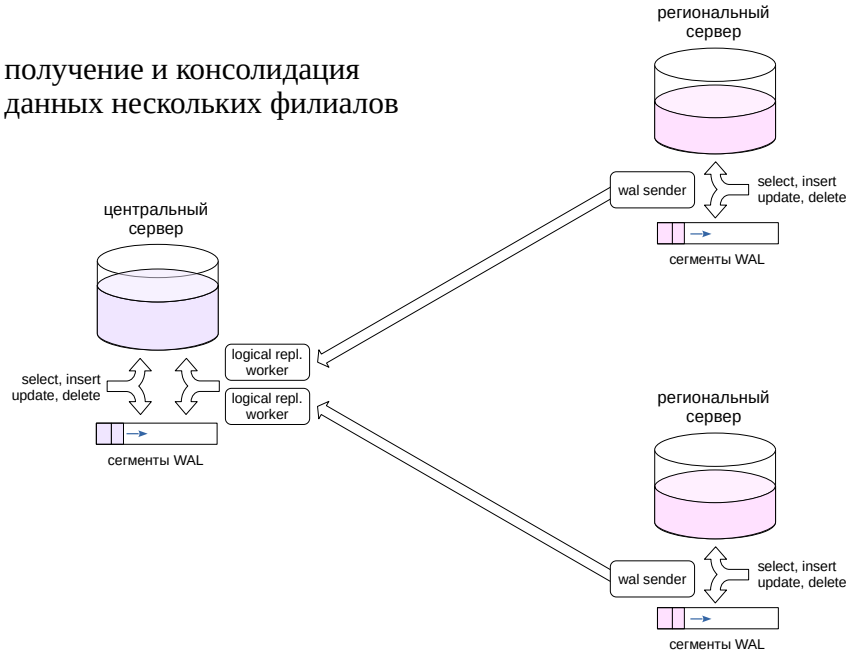
Для правильной репликации команды TRUNCATE нужно включать в ту же публикацию все таблицы, на которые ссылаются внешние ключи опустошаемой таблицы.

Соединение между серверами публикации и подписки должно быть достаточно стабильным. При разрыве слот репликации становится неактивным и сервер вынужден хранить сегменты WAL до момента восстановления соединения. Неактивный слот также удерживает горизонт очистки.

Изменения реплицируются построчно, поэтому выполнение на публикующем сервере команды SQL, затрагивающей много строк, приведет к повышенной нагрузке на подписчик.

Конфигурация по умолчанию не очень хорошо справляется с долгими транзакциями — они увеличивают нагрузку на публикующий сервер, поскольку публикация передает изменения только после фиксации транзакции. Параметр подписки streaming позволяет сгладить проблему: подписка может получать изменения строк без задержки и либо накапливать их во временном файле, либо сразу же применять, если есть свободный фоновый процесс.

1. Консолидация



Рассмотрим несколько задач, которые можно решить с помощью логической репликации.

Пусть имеются несколько региональных филиалов, каждый из которых работает на собственном сервере PostgreSQL. Задача состоит в консолидации части данных на центральном сервере.

Для решения задачи на региональных серверах создаются публикации необходимых данных. Центральный сервер подписывается на эти публикации. Полученные данные можно обрабатывать с помощью триггеров на стороне центрального сервера (например, приводя данные к единому виду).

Такая же схема, развернутая наоборот, позволяет, например, передавать справочную информацию от центрального сервера региональным.

С точки зрения бизнес-логики также есть множество особенностей, требующих внимания. В каких-то случаях может быть проще периодически передавать данные в пакетном режиме.

На иллюстрации: на центральном сервере работают два процесса приема журналов, по одному на каждую подписку.

2. Обновление серверов

обновление основной версии
без прерывания обслуживания

Задача: обеспечить обновление основной версии сервера без прерывания обслуживания клиентов.

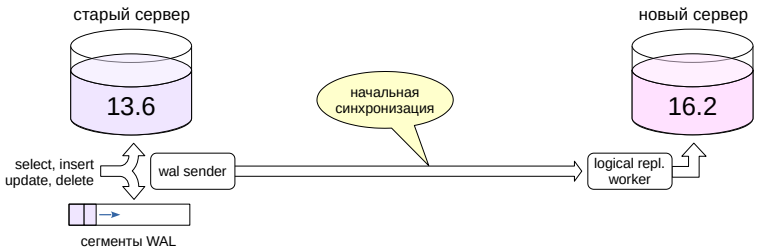
Поскольку между разными основными версиями нет двоичной совместимости, физическая репликация не помогает. Однако логическая репликация дает возможности для решения этой задачи.

Как обычно, требуются внешние средства для переключения пользователей между серверами.

Вначале создается новый сервер с желаемой версией PostgreSQL.

2. Обновление серверов

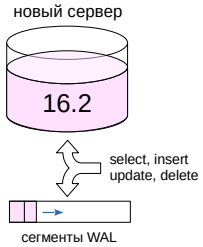
обновление основной версии
без прерывания обслуживания



Затем между серверами настраивается логическая репликация всех необходимых баз и данные синхронизируются. Это возможно благодаря тому, что для логической репликации не требуется двоичной совместимости между серверами.

2. Обновление серверов

обновление основной версии
без прерывания обслуживания

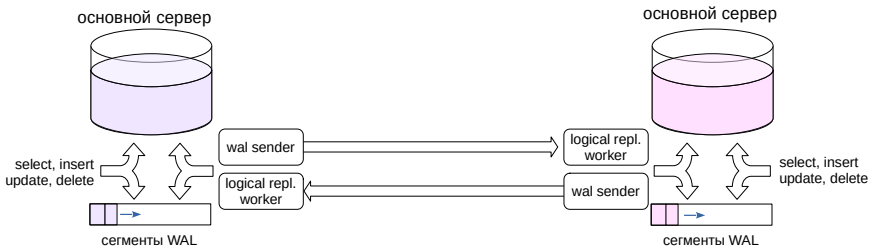


После этого клиенты переключаются на новый сервер, а старый выключается.

На самом деле процесс обновления с помощью логической репликации гораздо более сложен и сопряжен со значительными трудностями. Несколько подробнее он рассматривается в курсе DBA2, тема «Обновление сервера».

3. Мастер-мастер

кластер, в котором
данные могут изменять несколько серверов



Задача: обеспечить надежное хранение данных на нескольких серверах с возможностью записи на любом сервере (что полезно, например, для георасределенной системы).

Для решения можно использовать двунаправленную репликацию, передавая изменения в одних и тех же таблицах от одного сервера к другому и обратно.

Двунаправленная репликация появилась в PostgreSQL 16: можно одновременно опубликовать таблицу и подписаться на ее изменения на другом сервере.

Конечно, прикладная система должна быть построена таким образом, чтобы избегать конфликтов при изменении данных в одних и тех же таблицах. Например, использовать глобальные уникальные идентификаторы или гарантировать, что разные серверы работают с разными диапазонами ключей.

Надо учитывать, что система мастер-мастер, построенная на логической репликации, не обеспечивает сама по себе выполнение глобальных распределенных транзакций и, следовательно, согласованность данных между серверами. Кроме того, никаких средств для автоматизации обработки сбоев, подключения или удаления узлов из кластера и т. п. в PostgreSQL не предусмотрено — эти задачи должны решаться внешним средствами.

Итоги

Логическая репликация передает изменения строк отдельных таблиц

- разнонаправленная
- совместимость на уровне протокола

Модель публикация — подписчик

Следует учитывать имеющиеся ограничения

Практика

1. Настройте логическую репликацию произвольной таблицы в другую на том же самом сервере.
2. Настройте двунаправленную логическую репликацию одной и той же таблицы на двух разных серверах.

1. Если попробовать выполнить обычные действия, команда CREATE SUBSCRIPTION «повиснет». Внимательно изучите документацию:

https://postgrespro.ru/docs/postgresql/16/sql-createsubscription

2. Склонируйте второй сервер из резервной копии, как было показано в демонстрации.

При создании подписок на обоих серверах укажите следующие параметры: copy_data = false, origin = none.

1. Репликация на одном сервере

Тестовая таблица:

```
=> CREATE DATABASE replica_overview_logical_dba;
```

```
CREATE DATABASE
```

```
=> \c replica_overview_logical_dba
```

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

```
=> CREATE TABLE test (
    id int
);
```

```
CREATE TABLE
```

Копия базы с таблицей:

```
=> \c student
```

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

```
=> CREATE DATABASE replica_overview_logical_dba2 TEMPLATE replica_overview_logical_dba;
```

```
CREATE DATABASE
```

И установим необходимый уровень журнала:

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

```
ALTER SYSTEM
```

```
=> \q
```

```
student$ sudo pg_ctlcluster 16 main restart
```

Подключаемся к базам данных:

```
student$ psql replica_overview_logical_dba
```

```
student$ psql replica_overview_logical_dba2
```

В первой базе создадим публикацию:

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

```
CREATE PUBLICATION
```

Команда CREATE SUBSCRIPTION по умолчанию создает слот репликации, а для этого дожидается завершения всех транзакций, активных на момент начала создания слота. Одной из таких транзакция является та, которая выполняет команду CREATE SUBSCRIPTION, что приводит к бесконечному ожиданию.

Проблема решается созданием слота вручную и указанием его имени при создании подписки:

```
=> SELECT pg_create_logical_replication_slot('test_slot','pgoutput');
```

```
 pg_create_logical_replication_slot
------------------------------------
 (test_slot,0/216BAC8)
(1 row)
```

```
=> CREATE SUBSCRIPTION test
CONNECTION 'user=student dbname=replica_overview_logical_dba'
PUBLICATION test WITH (slot_name = test_slot, create_slot = false);
```

```
CREATE SUBSCRIPTION
```

Проверим:

```
=> INSERT INTO test SELECT * FROM generate_series(1,100);
```

```
INSERT 0 100
```

```
=> SELECT count(*) FROM test;
```

```
 count
-------
   100
(1 row)
```

Репликация работает.

Удалим публикацию, подписку и вторую базу данных.

```
=> DROP PUBLICATION test;
```

```
DROP PUBLICATION
```

```
=> DROP SUBSCRIPTION test;
```

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

```
=> DROP DATABASE replica_overview_logical_dba2 (FORCE);
```

```
DROP DATABASE
```

2. Двунаправленная репликация

Клонируем сервер с помощью резервной копии:

```
student$ pg_basebackup --pgdata=/home/student/tmp/backup --checkpoint=fast
```

Убеждаемся, что второй сервер остановлен, и выкладываем резервную копию:

```
student$ sudo pg_ctlcluster 16 replica stop
```

```
Cluster is not running.
```

```
student$ sudo rm -rf /var/lib/postgresql/16/replica
```

```
student$ sudo mv /home/student/tmp/backup /var/lib/postgresql/16/replica
```

```
student$ sudo chown -R postgres: /var/lib/postgresql/16/replica
```

Запускаем второй сервер:

```
student$ sudo pg_ctlcluster 16 replica start
```

База данных с таблицей и настройка уровня журнала также были склонированы:

```
student$ psql -p 5433 replica_overview_logical_dba
```

```
=> SHOW wal_level;
```

```
 wal_level
-----------
 logical
(1 row)
```

```
=> SELECT count(*) FROM test;
```

```
 count
-------
   100
(1 row)
```

Публикуем таблицу на обоих серверах:

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

```
CREATE PUBLICATION
```

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

```
CREATE PUBLICATION
```

Подписываемся:

```
=> CREATE SUBSCRIPTION test
CONNECTION 'port=5433 user=student dbname=replica_overview_logical_dba'
PUBLICATION test WITH (copy_data = false, origin = none);
```

```
NOTICE:  created replication slot "test" on publisher
CREATE SUBSCRIPTION
```

```
=> CREATE SUBSCRIPTION test
CONNECTION 'port=5432 user=student dbname=replica_overview_logical_dba'
PUBLICATION test WITH (copy_data = false, origin = none);
```

```
NOTICE:  created replication slot "test" on publisher
CREATE SUBSCRIPTION
```

Изменяем строки таблицы:

```
=> UPDATE test SET id = id + 100;
```

```
ERROR:  cannot update table "test" because it does not have a replica identity and
publishes updates
HINT:  To enable updating the table, set REPLICA IDENTITY using ALTER TABLE.
```

Чтобы изменения и удаления реплицировались, нужно задать идентификацию строк. По умолчанию строки идентифицируются по первичному ключу.

```
=> ALTER TABLE test ADD PRIMARY KEY (id);
```

```
ALTER TABLE
```

```
=> ALTER TABLE test ADD PRIMARY KEY (id);
```

```
ALTER TABLE
```

Повторяем попытку:

```
=> UPDATE test SET id = id + 100;
```

```
UPDATE 100
```

```
=> UPDATE test SET id = id + 100;
```

```
UPDATE 100
```

Смотрим результат:

```
=> SELECT min(id), max(id) FROM test;
```

```
 min | max
-----+-----
 201 | 300
(1 row)
```

```
=> SELECT min(id), max(id) FROM test;
```

```
 min | max
-----+-----
 201 | 300
(1 row)
```