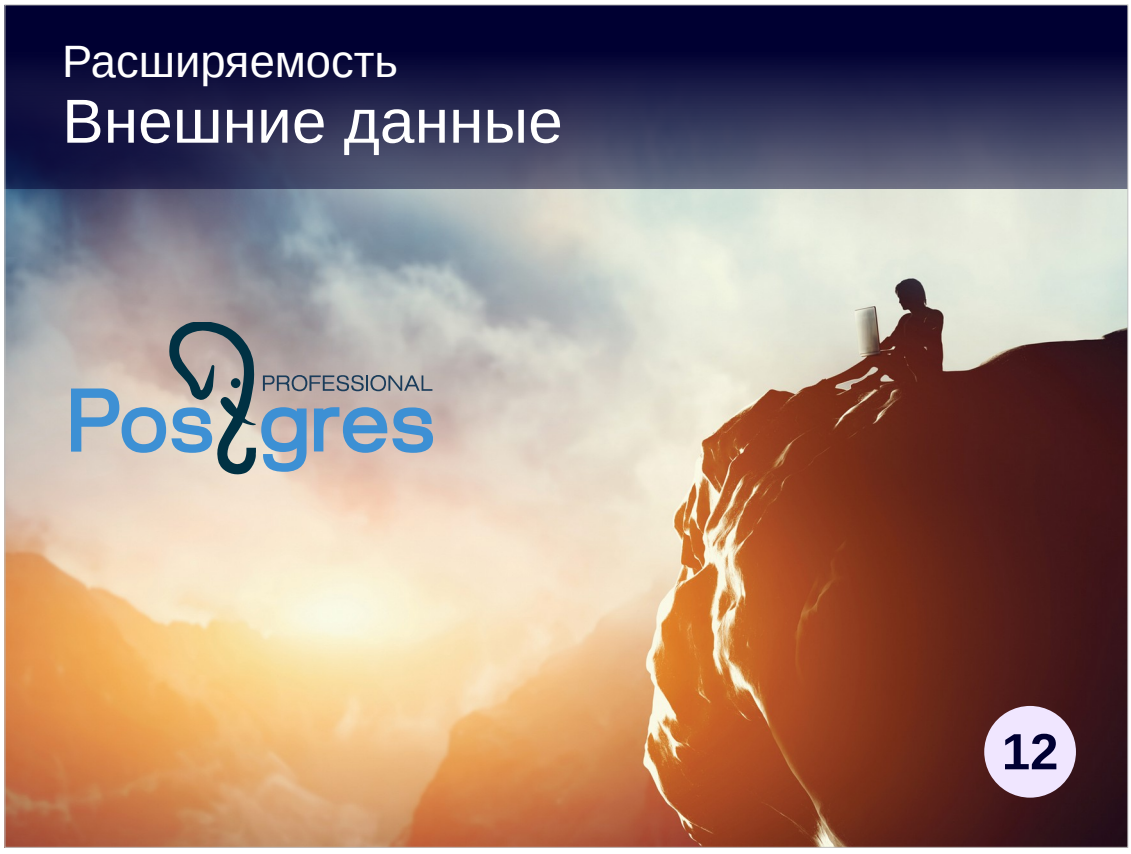

Авторы: Егор Рогов, Павел Лузанов

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

Темы

Назначение оберток сторонних данных

Настройка доступа к внешним данным

Примеры: postgres_fdw и file_fdw

Другие доступные обертки

Назначение

Представление внешних данных как обычных таблиц

- стандарт ISO/IEC 9075-9 (SQL/MED)
- поддерживаются SELECT, INSERT, UPDATE, DELETE
- триггеры
- обычное разграничение доступа (GRANT, REVOKE)

Способ миграции данных из других СУБД

В будущем — механизм для шардинга

- вместе с секционированием

В стандарте ISO/IEC 9075-9 (SQL/MED) определяется, каким образом базы данных SQL должны работать с внешними источниками. При этом доступ к внешним данным возможен как на чтение, так и на запись.

Внешние данные представлены в PostgreSQL как таблицы, которые называются *внешними* или *сторонними*. С ними можно работать при помощи обычных команд DML: INSERT, UPDATE, DELETE, SELECT. На внешние таблицы можно создавать триггеры. А для разграничения прав используются команды GRANT, REVOKE.

Основное отличие внешних таблиц от обычных в том, что данные физически не хранятся в БД PostgreSQL, а загружаются из внешней системы (отправляются во внешнюю систему) при выполнении запросов.

Обертки сторонних данных могут быть весьма полезны и при миграции данных в PostgreSQL из других СУБД.

Поддержка внешних данных активно развивается, поскольку (вместе с секционированием) является одним их компонентов для реализации шардинга. Идея в том, чтобы секции основной таблицы были представлены внешними таблицами, физически находящимися на других серверах. Это можно сделать уже сейчас, но пока не хватает поддержки двухфазной фиксации, параллельного доступа к секциями и других составляющих.

https://postgrespro.ru/docs/postgresql/12/ddl-foreign-data

Компоненты

1. Обертка сторонних данных
   - каков тип внешнего источника данных?
   - postgres_fdw: внешний сервер PostgreSQL

Чтобы настроить доступ к внешним данным, нужно создать несколько объектов базы данных.

Все начинается с *обертки сторонних данных* (foreign data wrapper, FDW). Обертка определяет тип внешнего источника данных. Например, postgres_fdw реализует доступ к базам данных PostgreSQL, а file_fdw — доступ к файлам операционной системы.

https://postgrespro.ru/docs/postgresql/12/postgres-fdw

https://postgrespro.ru/docs/postgresql/12/sql-createforeigndatawrapper

Обертка сторонних данных

```
=> CREATE DATABASE ext_foreign;

CREATE DATABASE

=> \c ext_foreign

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

Обертка создается при создании расширения:

```
=> CREATE EXTENSION postgres_fdw;

CREATE EXTENSION

=> \dew

                     List of foreign-data wrappers
     Name     |  Owner  |       Handler        |       Validator
--------------+---------+----------------------+------------------------
 postgres_fdw | student | postgres_fdw_handler | postgres_fdw_validator
(1 row)
```

Компоненты

1. Обертка сторонних данных

   каков тип внешнего источника данных?

2. Внешний сервер

   как подключиться к внешнему источнику?

   postgres_fdw: узел, порт, база данных

Следующий объект — *внешний сервер*. Сервер определяет, как подключаться к внешнему источнику данных.

Например, если источником является СУБД, то надо знать узел, порт, имя базы данных. (Но не имя пользователя и пароль. Эта информация появится чуть позже.)

Если требуется подключение к нескольким внешним источникам (одного типа, который определен оберткой), то для каждого из них нужно создать отдельный сервер.

https://postgrespro.ru/docs/postgresql/12/sql-createserver

Внешний сервер

В качестве внешнего сервера выберем базу данных книжного магазина на нашем же локальном сервере.

```
=> CREATE SERVER remote_server
FOREIGN DATA WRAPPER postgres_fdw
OPTIONS (
    host 'localhost',
    port '5432',
    dbname 'bookstore2'
);
```

```
CREATE SERVER
```

Обратите внимание, что в параметрах сервера не указывается роль.

```
=> \x
```

```
Expanded display is on.
```

```
=> \des+
```

```
List of foreign servers
-[ RECORD 1 ]--------+-----------------------------------------------------
Name                 | remote_server
Owner                | student
Foreign-data wrapper | postgres_fdw
Access privileges    |
Type                 |
Version              |
FDW options          | (host 'localhost', port '5432', dbname 'bookstore2')
Description          |


=> \x
```

```
Expanded display is off.
```

Компоненты

1. Обертка сторонних данных

   каков тип внешнего источника данных?

2. Внешний сервер

   как подключиться к внешнему источнику?

3. Сопоставление ролей

   как локальные роли связаны с разграничением доступа, используемым внешним источником?

   postgres_fdw: и там, и там — обычные роли PostgreSQL

Далее следует *сопоставление ролей*.

Внешний источник может выполнять аутентификацию при подключении. В таких случаях необходимо определить, какие локальные роли могут подключаться к каким учетным записям на внешнем сервере.

Если используется аутентификация по паролю, то здесь же указывается и пароль.

https://postgrespro.ru/docs/postgresql/12/sql-createusermapping

Сопоставление ролей

Настроим соответствие: пусть локальная роль student подключается к внешнему серверу как postgres.

```
=> CREATE USER MAPPING FOR student
SERVER remote_server
OPTIONS (
    user 'postgres',
    password 'postgres'
);
```

```
CREATE USER MAPPING
```

```
=> \deu+
```

```
                        List of user mappings
    Server     | User name |               FDW options
---------------+-----------+-----------------------------------------
 remote_server | student   | ("user" 'postgres', password 'postgres')
(1 row)
```

Разумеется, можно настраивать несколько соответствий для разных ролей.

Компоненты

1. Обертка сторонних данных
   - каков тип внешнего источника данных?
2. Внешний сервер
   - как подключиться к внешнему источнику?
3. Сопоставление ролей
   - как локальные роли связаны с ролями внешнего источника?
4. Внешние таблицы
   - какова структура внешних данных?
   - postgres_fdw: данные в обычных таблицах

И, наконец, можно приступить к созданию *внешних таблиц*. Внешняя таблица определяет, как представить данные из внешнего источника в виде обычной реляционной таблицы.

Для внешних таблиц можно добавлять ограничения целостности NOT NULL и CHECK. Эти ограничения не будут применяться при выполнении команд, ведь в любом случае PostgreSQL не сможет обеспечить соблюдение ограничений на внешнем источнике. Но планировщик может использовать информацию об ограничениях при построении плана выполнения запроса.

https://postgrespro.ru/docs/postgresql/12/sql-createforeigntable

https://postgrespro.ru/docs/postgresql/12/sql-importforeignschema

Внешние таблицы

Внешнюю таблицу можно создать явным образом, при необходимости указывая названия объектов, если они отличаются:

```
=> CREATE FOREIGN TABLE remote_users (
    id integer OPTIONS (column_name 'user_id') NOT NULL,
    username text NOT NULL,
    email text NOT NULL
)
SERVER remote_server
OPTIONS (
    schema_name 'public',
    table_name 'users'
);
```

```
CREATE FOREIGN TABLE
```

Можно указать только некоторые ограничения целостности, но в любом случае они не проверяются локально и просто отражают ограничения, накладываемые внешней системой.

```
=> \det
```

```
        List of foreign tables
 Schema |    Table     |    Server
--------+--------------+---------------
 public | remote_users | remote_server
(1 row)
```

```
=> SELECT * FROM remote_users WHERE id = 1;
```

```
 id | username |      email
----+----------+-----------------
  1 | alice    | alice@localhost
(1 row)
```

Как выполняется такой запрос? Иными словами, передается ли предикат внешнему серверу, чтобы он сам выбрал эффективный способ выполнения, или фильтрацией занимается локальный сервер?

```
=> EXPLAIN (verbose, costs off)
    SELECT * FROM remote_users WHERE id = 1;
```

```
                                      QUERY PLAN
---------------------------------------------------------------------------------------
 Foreign Scan on public.remote_users
   Output: id, username, email
   Remote SQL: SELECT user_id, username, email FROM public.users WHERE ((user_id = 1))
(3 rows)
```

Планировщик умеет распределять работу между серверами. Также ничто не мешает использовать в одном запросе как внешние, так и локальные данные.

```
=> EXPLAIN (costs off)
    SELECT username FROM remote_users
    UNION ALL
    SELECT rolname FROM pg_roles;
```

```
               QUERY PLAN
----------------------------------------
 Append
   ->  Subquery Scan on "*SELECT* 1"
         ->  Foreign Scan on remote_users
   ->  Subquery Scan on "*SELECT* 2"
         ->  Seq Scan on pg_authid
(5 rows)
```

Другой способ — не создавать внешние таблицы по одной, а импортировать внешнюю схему (всю или выборочно несколько таблиц), если обертка это позволяет.

```
=> CREATE SCHEMA bookstore2;
```

```
CREATE SCHEMA
```

```
=> IMPORT FOREIGN SCHEMA public
    LIMIT TO (users, sessions)
    FROM SERVER remote_server
    INTO bookstore2;

IMPORT FOREIGN SCHEMA

=> SELECT * FROM bookstore2.users;

 user_id | username |       email
---------+----------+-------------------
       1 | alice    | alice@localhost
       2 | bob      | bob@localhost
       3 | charlie  | charlie@localhost
(3 rows)
```

Обертка postgres_fdw позволяет и изменять данные:

```
=> UPDATE remote_users
SET email = 'alice@gmail.com'
WHERE id = 1;

UPDATE 1
```

Соединения и транзакции

Управление соединениями

- открывается при первом обращении к внешним данным
- остается открытым до конца сеанса

Управление удаленными транзакциями

- фиксируется или прерывается автоматически, когда фиксируется или прерывается локальная транзакция
- Repeatable Read для локальных транзакций уровней Read Committed и Repeatable Read
- Serializable для локальных транзакций Serializable
- согласованность не гарантируется (нет двухфазной фиксации)

При работе с внешней СУБД важно понимать, как происходит управление соединениями и транзакциями.

Расширение postgres_fdw открывает соединение автоматически при первом обращении к внешним данным (в рамках одного внешнего сервера — одно на каждую роль), и соединение остается открытым до конца локального сеанса. С этим надо быть аккуратным, чтобы не исчерпать ограничение числа подключений.

Удаленные транзакции «привязаны» к локальным:

- расширение автоматически начинает удаленную транзакцию, когда локальная обращается к внешним данным;

- удаленная транзакция автоматически завершается с завершением локальной (фиксацией или обрывом).

Тем не менее, локальная и удаленная транзакции — это *разные* транзакции, а не одна глобальная. Расширение postgres_fdw пока не позволяет использовать двухфазную фиксацию, поэтому согласованность не гарантируется. Например, локальная транзакция может быть зафиксирована, а удаленная — нет из-за сетевого сбоя.

https://postgrespro.ru/docs/postgresql/12/postgres-fdw#id-1.11.7.42.11

Сравнение с dblink

	postgres_fdw	dblink
доступные команды SQL	SELECT, INSERT, UPDATE, DELETE	любые
совмещение в запросе локальных и внешних данных	да	сложно
управление соединением	автоматическое	ручное
управление транзакциями	автоматическое	ручное
глобальные транзакции	нет	нет
стандартизованный способ	да	нет

В теме «Фоновые процессы» рассматривалось расширение dblink, которое предназначено для выполнения запросов на удаленном сервере PostgreSQL и предоставляет возможности, схожие с возможностями postgres_fdw.

Обертка postgres_fdw позволяет выполнять ограниченный набор команд SQL, но в большинстве случаев более удобна: она сама управляет соединениями и транзакциями («привязывая» их к локальному соединению и транзакции). Она позволяет использовать в одном запросе как локальные данные, так и внешние; при этом автоматически формируется распределенный план выполнения запроса. Синтаксис доступа к данным определяется стандартом SQL.

Зато расширение dblink позволяет выполнять любые команды SQL. В некоторых случаях может оказаться важным управление соединениями и транзакциями вручную.

Оба способа не предоставляют глобальных (распределенных) транзакций.

Таким образом, для доступа к внешним данным в большинстве случаев следует выбрать postgres_fdw. Расширение dblink стоит использовать, если нужно сделать что-то, не укладывающееся в рамки оберток сторонних данных.

Расширение file_fdw

1. Обертка сторонних данных

   тип внешнего источника — текстовый файл с разделителями

2. Внешний сервер

   как подключиться — не требуется дополнительной информации

3. Сопоставление ролей

   не имеет смысла, т. к. во внешнем источнике нет ролей

4. Внешние таблицы

   структура данных — описание полей файла в виде столбцов таблицы

Другой пример обертки сторонних данных дает расширение file_fdw, предназначенное для доступа к файлам в ФС сервера PostgreSQL.

Доступ возможен только на чтение и выполняется с помощью того же механизма, что и SQL-команда COPY.

https://postgrespro.ru/docs/postgresql/12/file-fdw

Расширение file_fdw

Расширения file_fdw позволяет обращаться к любому текстовому файлу и задействует тот же механизм, что и в команде COPY. Поэтому для демонстрации мы запишем имеющуюся таблицу в файл, а затем прочитаем данные из файла.

```
=> COPY (SELECT * FROM remote_users)
TO '/var/lib/postgresql/users.txt'
WITH (
    format 'text',
    delimiter '/'
);
```

```
COPY 3
```

```
postgres$ cat ~/users.txt
```

```
2/bob/bob@localhost
3/charlie/charlie@localhost
1/alice/alice@gmail.com
```

```
=> CREATE EXTENSION file_fdw;
```

```
CREATE EXTENSION
```

```
=> CREATE SERVER file_server
    FOREIGN DATA WRAPPER file_fdw;
```

```
CREATE SERVER
```

```
=> CREATE FOREIGN TABLE file_users (
    id integer,
    username text,
    email text
)
SERVER file_server
OPTIONS (
    filename '/var/lib/postgresql/users.txt',
    format 'text',
    delimiter '/'
);
```

```
CREATE FOREIGN TABLE
```

```
=> SELECT * FROM file_users;
```

```
 id | username |       email
----+----------+-------------------
  2 | bob      | bob@localhost
  3 | charlie  | charlie@localhost
  1 | alice    | alice@gmail.com
(3 rows)
```

Другие доступные обертки

Базы данных

- ODBC, JDBC
- различные реляционные и NoSQL СУБД

Файлы различных форматов

Геоданные и научные данные

Веб-ресурсы

- стандартные протоколы (RSS, IMAP, WWW...)
- для различных сайтов

Можно написать собственную обертку

- на C или Python (с помощью расширения Multicorn)

Рассмотренные обертки сторонних данных, file_fdw и postgres_fdw, поставляются вместе с PostgreSQL в виде contrib-модулей.

Помимо этого существует множество других оберток, которые предоставляют доступ ко многим популярным базам данных и не только. Информацию о них можно получить:

- на страничке wiki: https://wiki.postgresql.org/wiki/Fdw;
- на сайте PGXN: http://pgxn.org/tag/fdw/.

Если нужную обертку не удалось найти, то ее можно написать самостоятельно. Для этого потребуется язык C: https://postgrespro.ru/docs/postgresql/12/fdwhandler

Другой вариант — воспользоваться расширением Multicorn (https://multicorn.org/), которое позволяет написать обертку на языке Python.

Итоги

Обертки сторонних данных — стандартный способ работы с данными любых внешних источников

- доступ к PostgreSQL и файлам «из коробки»
- есть реализации оберток для огромного числа систем

Альтернатива — расширение dblink для выполнения произвольных запросов на удаленном сервере PostgreSQL

Практика

1. Поставщик информирует магазин о новинках книжного рынка, предоставляя CSV-файл определенного формата. Загрузите информацию из файла 2020.csv, расположенного в домашнем каталоге пользователя student, в таблицы базы данных. Формат файла описан в комментариях к слайду.

   Учтите, что авторы, указанные в файле, могут уже существовать в базе данных, но в файле могут быть допущены опечатки.
2. Проверьте в приложении, что новые книги успешно переданы на логическую реплику, и для них работает полнотекстовый поиск.

1. CSV-файл содержит следующие поля:

- фамилия автора;
- имя автора;
- отчество автора;
- название книги;
- формат издания;
- количество страниц;
- ISBN;
- аннотация;
- издательство;
- год выпуска;
- гарнитура;
- номер издания;
- серия;
- имя файла с обложкой (файл находится в каталоге covers).

Если у книги несколько авторов, то они следуют в файле в том порядке, в котором указаны в выходных сведениях. При этом для второго и следующих авторов все поля, относящиеся к книге — пустые.

Воспользуйтесь расширением file_fdw, указав для файла формат CSV.

Для борьбы с опечатками вспомните про расширение pg_trgm, с которым вы познакомились в практике к теме «Классы операторов».

2. Логическая реплика с подпиской на изменения книг настраивалась в задании к теме «Логическая репликация».

```
student$ psql bookstore2
```

1. Загрузка новых поступлений

```
=> CREATE EXTENSION file_fdw;
```

```
CREATE EXTENSION
```

```
=> CREATE SERVER file_server
    FOREIGN DATA WRAPPER file_fdw;
```

```
CREATE SERVER
```

```
=> CREATE FOREIGN TABLE new_books (
    last_name text,
    first_name text,
    middle_name text,
    title text,
    format text,
    pages integer,
    isbn text,
    abstract text,
    publisher text,
    year text,
    typeface text,
    edition integer,
    series text,
    cover_filename text
)
SERVER file_server
OPTIONS (
    filename '/home/student/2020.csv',
    format 'csv'
);
```

```
CREATE FOREIGN TABLE
```

Здесь предполагается, что несколько процедур загрузки книг не будут запускаться параллельно. Если такой гарантии нет, в коде надо отдельно предусмотреть эту возможность, устанавливая необходимые блокировки.

При сопоставления авторов используется нечеткий поиск с помощью триграмм. Оператор % отсекает совсем непохожих (предел похожести по умолчанию 0.3 слишком низок для нашего случая, поэтому увеличиваем его). Из оставшихся выбираем лучшее совпадение, упорядочивая кандидатов с помощью функции similarity.

```
=> CREATE EXTENSION pg_trgm;
```

```
CREATE EXTENSION
```

```
=> SELECT set_limit(0.8); -- предел похожести для оператора %
```

```
 set_limit
-----------
       0.8
(1 row)
```

```
=> DO $$
DECLARE
    i new_books;
    book_id bigint;
    author_id bigint;
    author_cause text;
    seq_num integer;
BEGIN
    FOR i IN SELECT * FROM new_books
    LOOP
        IF i.title IS NOT NULL THEN
            INSERT INTO books AS b(
                title, format, pages, additional, cover
            ) VALUES (
                i.title,
                i.format::text::book_format,
                i.pages,
                jsonb_build_object(
                    'ISBN',         i.isbn,
                    'Аннотация',    i.abstract,
                    'Издательство', i.publisher,
                    'Год выпуска',  i.year,
                    'Гарнитура',    i.typeface,
                    'Издание',      i.edition,
                    'Серия',        i.series
                ),
                pg_read_binary_file(
                    '/home/student/covers/'||i.cover_filename
                )
            )
            RETURNING b.book_id INTO book_id;
            seq_num := 1;
            RAISE NOTICE 'Книга: % (%)', i.title, book_id;
        END IF;

        -- есть ли точное совпадение с имеющимся автором?
        SELECT a.author_id, 'точное совпадение'
        INTO author_id, author_cause
        FROM authors a
        WHERE a.last_name   = i.last_name
          AND a.first_name  = i.first_name
          AND a.middle_name = i.middle_name;
        -- если нет, то может найдется очень похожий?
        IF author_id IS NULL THEN
            SELECT a.author_id, 'НЕточное совпадение'
            INTO author_id, author_cause
            FROM authors a
            WHERE (a.last_name || a.first_name || a.middle_name) %
                  (i.last_name || i.first_name || i.middle_name)
            ORDER BY similarity(
                  (a.last_name || a.first_name || a.middle_name),
                  (i.last_name || i.first_name || i.middle_name)
            ) DESC
            LIMIT 1;
        END IF;
        -- если и похожего нет, то считаем автора новым
        IF author_id IS NULL THEN
            INSERT INTO authors AS a(first_name,last_name,middle_name)
                VALUES (i.first_name, i.last_name, i.middle_name)
                RETURNING a.author_id, 'новый'
                INTO author_id, author_cause;
        END IF;
        RAISE NOTICE '  Автор: %, % (%)',
            i.last_name, author_cause, author_id;

        INSERT INTO authorships(book_id, author_id, seq_num)
            VALUES (book_id, author_id, seq_num);
        seq_num := seq_num + 1;
    END LOOP;
END;
$$;
```

```
NOTICE:  Книга: Sapiens. Краткая история человечества (97)
NOTICE:    Автор: Харари, новый (156)
NOTICE:  Книга: Генетический детектив. От исследования рибосомы к Нобелевской премии (98)
NOTICE:    Автор: Рамакришнан, новый (157)
NOTICE:  Книга: Основы технологий баз данных (99)
NOTICE:    Автор: Новиков, точное совпадение (1)
NOTICE:    Автор: Горшкова, НЕточное совпадение (2)
NOTICE:    Автор: Графеева, новый (158)
NOTICE:  Книга: Код. Тайный язык информатики (100)
NOTICE:    Автор: Петцольд, новый (159)
DO

=> DROP FOREIGN TABLE new_books;

DROP FOREIGN TABLE
```

Практика

1. В двух разных базах данных находятся две таблицы с одинаковым набором столбцов. Выведите строки, которые присутствуют в первой таблице, но отсутствуют во второй. Решите задачу с помощью postgres_fdw и с помощью dblink и сравните два способа.
2. Как можно проверить, какой уровень изоляции использует обертка postgres_fdw и как она управляет соединениями и транзакциями?

1. Расширение dblink демонстрировалось в теме «Фоновые процессы».

2. Настройте на втором сервере журнал сообщений таким образом, чтобы в него записывалась информация

- о подключениях (*log_connections*);

- об отключениях (*log_disconnections*);

- о выполняемых командах (*log_statement*).

Обратитесь к таблице на втором сервере с помощью postgres_fdw и проверьте, что попало в журнал сообщений.

1. Сравнение таблиц в разных базах

Таблица в первой базе данных:

```
=> CREATE DATABASE ext_foreign;
```

```
CREATE DATABASE
```

```
=> \c ext_foreign
```

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

```
=> CREATE TABLE test (
    s text
);
```

```
CREATE TABLE
```

```
=> INSERT INTO test VALUES ('foo'),('bar'),('baz');
```

```
INSERT 0 3
```

И во второй (расположим ее на втором сервере):

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

```
=> CREATE DATABASE ext_foreign;
```

```
CREATE DATABASE
```

```
=> \c ext_foreign
```

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

```
=> CREATE TABLE test (
    s text
);
```

```
CREATE TABLE
```

```
=> INSERT INTO test VALUES ('foo'),('bar');
```

```
INSERT 0 2
```

Создаем стороннюю таблицу для доступа к таблице во второй базе данных:

```
=> CREATE EXTENSION postgres_fdw;
```

```
CREATE EXTENSION
```

```
=> CREATE SERVER remote_server
FOREIGN DATA WRAPPER postgres_fdw
OPTIONS (
    host 'localhost',
    port '5433',
    dbname 'ext_foreign'
);
```

```
CREATE SERVER
```

```
=> CREATE USER MAPPING FOR student
SERVER remote_server
OPTIONS (
    user 'student',
    password 'student'
);
```

```
CREATE USER MAPPING
```

```
=> CREATE FOREIGN TABLE test2 (
    s text
)
SERVER remote_server
OPTIONS (
    schema_name 'public',
    table_name 'test'
);
```

```
CREATE FOREIGN TABLE
```

Сравниваем:

```
=> SELECT * FROM test
EXCEPT
SELECT * FROM test2;
```

```
  s
-----
 baz
(1 row)
```

С помощью расширения dblink задачу можно решить следующим образом:

```
=> CREATE EXTENSION dblink;
```

```
CREATE EXTENSION
```

```
=> SELECT * FROM test
EXCEPT
SELECT * FROM dblink(
    'host=localhost port=5433 dbname=ext_foreign user=student password=student',
    'SELECT * FROM test'
) AS (s text);
```

```
  s
-----
 baz
(1 row)
```

В этом случае подготовительные действия практически не требуются, но сам запрос выглядит сложнее.

---

2. Проверка работы postgres_fdw

На втором сервере настроим журнал сообщений так, чтобы в него попадала информация о подключениях и отключениях, и о выполняемых командах.

```
=> ALTER SYSTEM SET log_connections = on;
```

```
ALTER SYSTEM
```

```
=> ALTER SYSTEM SET log_disconnections = on;
```

```
ALTER SYSTEM
```

```
=> ALTER SYSTEM SET log_statement = 'all';
```

```
ALTER SYSTEM
```

```
=> SELECT pg_reload_conf();
```

```
 pg_reload_conf
----------------
 t
(1 row)
```

Начнем новое соединение и локальную транзакцию Read Committed.

```
=> \q
```

```
student$ psql ext_foreign
```

```
=> BEGIN;
```

```
BEGIN
```

Обращаемся к сторонней таблице:

```
=> SELECT * FROM test2;
```

```
  s
-----
 foo
 bar
(2 rows)
```

В журнале второго сервера видим:

- установлено новое соединение;
- в нем выставлены некоторые параметры;
- начата транзакция с уровнем изоляции Repeatable Read;
- с помощью курсора прочитаны записи из таблицы test.

```
postgres$ tail -n 12 /var/log/postgresql/postgresql-12-replica.log
```

```
2023-07-26 11:32:25.446 MSK [117979] [unknown]@[unknown] LOG:  connection received: host=127.0.0.1 port=35598
2023-07-26 11:32:25.449 MSK [117979] student@ext_foreign LOG:  connection authorized: user=student database=ext_foreign application_name=postgres_fdw SSL enabled (protocol=TLSv1.3, cip
2023-07-26 11:32:25.450 MSK [117979] student@ext_foreign LOG:  statement: SET search_path = pg_catalog
2023-07-26 11:32:25.450 MSK [117979] student@ext_foreign LOG:  statement: SET timezone = 'UTC'
2023-07-26 11:32:25.452 MSK [117979] student@ext_foreign LOG:  statement: SET datestyle = ISO
2023-07-26 11:32:25.452 MSK [117979] student@ext_foreign LOG:  statement: SET intervalstyle = postgres
2023-07-26 11:32:25.452 MSK [117979] student@ext_foreign LOG:  statement: SET extra_float_digits = 3
2023-07-26 11:32:25.452 MSK [117979] student@ext_foreign LOG:  statement: START TRANSACTION ISOLATION LEVEL REPEATABLE READ
2023-07-26 11:32:25.452 MSK [117979] student@ext_foreign LOG:  execute <unnamed>: DECLARE c1 CURSOR FOR
        SELECT s FROM public.test
2023-07-26 11:32:25.452 MSK [117979] student@ext_foreign LOG:  statement: FETCH 100 FROM c1
2023-07-26 11:32:25.453 MSK [117979] student@ext_foreign LOG:  statement: CLOSE c1
```

Завершаем локальную транзакцию.

```
=> COMMIT;
```

```
COMMIT
```

В журнале видим, что удаленная транзакция также завершена:

```
postgres$ tail -n 1 /var/log/postgresql/postgresql-12-replica.log
```

```
2023-07-26 11:32:25.644 MSK [117979] student@ext_foreign LOG:  statement: COMMIT TRANSACTION
```

Завершаем сеанс.

```
=> \q
```

В этот момент завершается и соединение с удаленным узлом:

```
postgres$ tail -n 2 /var/log/postgresql/postgresql-12-replica.log
```

```
2023-07-26 11:32:25.732 MSK [117979] student@ext_foreign LOG:  could not receive data from client: Connection reset by peer
2023-07-26 11:32:25.732 MSK [117979] student@ext_foreign LOG:  disconnection: session time: 0:00:00.286 user=student database=ext_foreign host=127.0.0.1 port=35598
```

Журнал сообщений можно использовать, чтобы разобраться, как работа внешних средств выглядит с точки зрения СУБД.