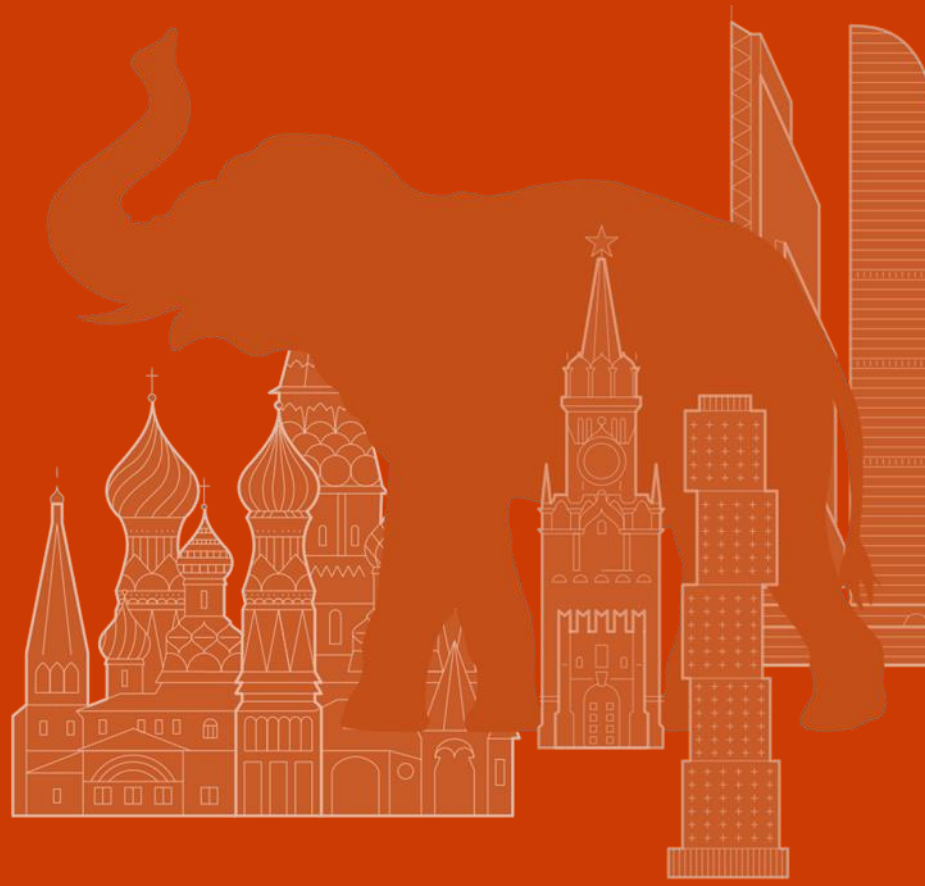
tantor



Автономные транзакции в PostgreSQL

Вадим Яценко, Иван Кушнаренко, Тантор Лабс

Disclaimer доклада

Доклад состоит из 2 частей:

- В первой части речь пойдет об общих подходах к реализации автономных транзакций в PostgreSQL;
- Во второй части будет проведена небольшая ретроспектива попыток реализации автономных транзакций в ядре PostgreSQL, а также сравнительный анализ различных подходов.

В данном докладе описываются некоторые из возможных вариантов реализации автономных транзакций (autonomous transactions, ATX) в PostgreSQL.

За время существования СУБД Postgres было предложена масса различных способов реализации ATX в ядре, часть из которых в итоге были имплементированы в коммерческих версиях в том или ином виде.

Мы расскажем о наиболее популярных подходах для реализации ATX в PostgreSQL, а также затронем некоторые из предложенных патчей.

Консенсус по тому, как должны быть реализованы ATX в PostgreSQL до сих пор не достигнут.

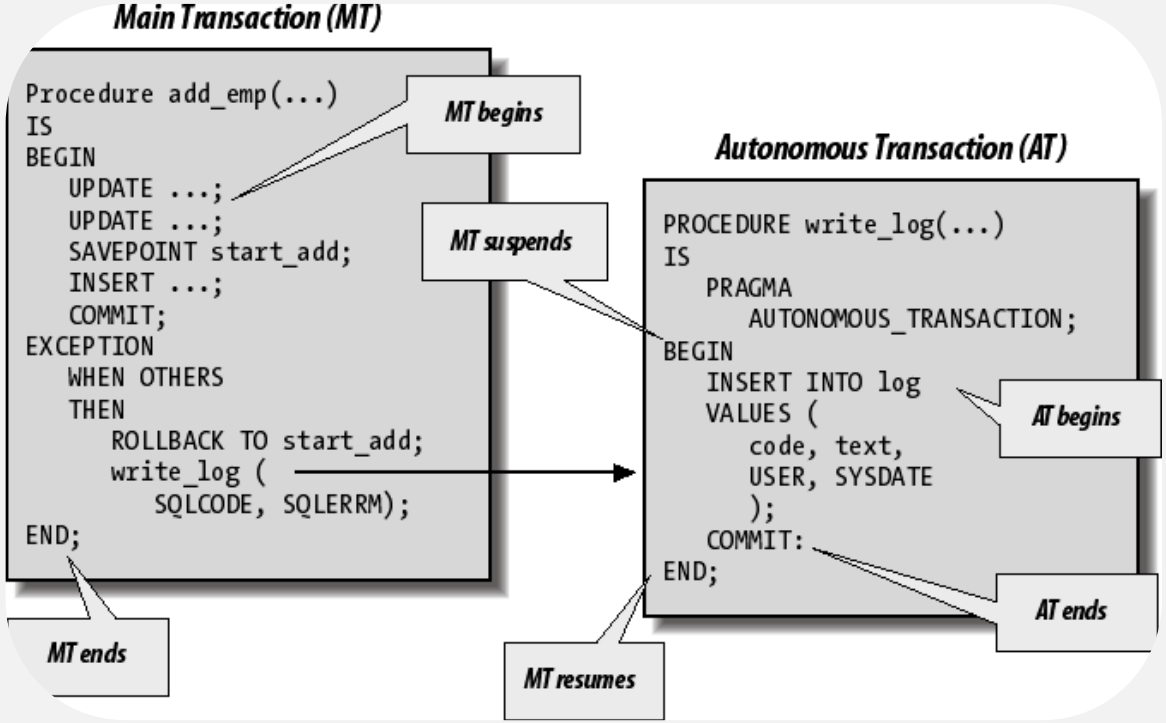
Часть 1. Что такое Autonomous transactions (ATX)?

Autonomous transaction(ATX) - это транзакция, запущенная из другой основной транзакции для выполнения SQL-запросов, которые будут зафиксированы или отменены независимо от основной вызывающей транзакции.

Иллюстрация из книги Трое из Простоквашино. Эдуард Успенский

Введение. Что такое Autonomous transactions (ATX)?

- Не является стандартом SQL
- Термин пришел из “мира” Oracle и обозначает (под)транзакцию, которая не связана с основной транзакцией (по принципу «запустил и забыл»)
- Операции SQL, выполняемые в автономной транзакции, могут быть видны основной транзакции, когда они зафиксированы
- Когда автономная транзакция завершается неудачно, основной (вызывающей) транзакции не передается исключение (exception), что позволяет ей завершиться успешно



Oracle PL/SQL Programming, 5th Edition by Steven Feuerstein, Bill Pribyl

Введение. Когда полезны Autonomous transactions (ATX)?

- Логирование и журналирование действий
- Интеграция с внешними системами (alerting, вызов API)
- Отладка хранимых процедур
- Пакетные задания. Когда есть задача по манипулированию большими объемами данных, которые необходимо разбить на сегменты. При этом каждый сегмент фиксируется отдельно.

— Дедушка, а Баба Яга полезная или вредная?

— Всякая поганка в лесу к чему-нибудь назначена. Потому порядок.

© Приключения домовёнка Кузи

Введение. В каких коммерческих СУБД есть эта “Баба Яга”?

- PRAGMA AUTONOMOUS_TRANSACTION
- AUTONOMOUS
- AUTONOMOUS TRANSACTION
- “AUTONOMOUS”

© Приключения домовёнка Кузи

Введение. “Баба Яга” в Oracle

```
CREATE OR REPLACE PROCEDURE log_actions
(msg VARCHAR2, event_date TIMESTAMP)
AUTHID DEFINER AS
PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
        INSERT INTO table_tracking
        VALUES (msg, event_date);
COMMIT;
END log_actions;
```

Основные характеристики автономных транзакций:

- Автономные транзакции не зависят от состояния или возможного расположения основной транзакции. Например:
  - Автономная транзакция не видит никаких изменений, внесенных основной транзакцией.
  - Когда автономная транзакция фиксируется или откатывается, это не влияет на результат основной транзакции.
- Изменения, которые производит автономная транзакция, видны другим транзакциям, как только эта автономная транзакция фиксируется.
- Автономные транзакции могут запускать другие автономные транзакции.

Введение. “Баба Яга” в IBM DB2

```
CREATE PROCEDURE PROC( IN p INT , OUT
outtab TABLE (A INT)) LANGUAGE SQLSCRIPT
AS
BEGIN
DECLARE errCode INT;
DECLARE errMsg VARCHAR(5000);
DECLARE EXIT HANDLER FOR SQLEXCEPTION
BEGIN AUTONOMOUS TRANSACTION
errCode= ::SQL_ERROR_CODE;
errMsg= ::SQL_ERROR_MESSAGE ;
INSERT INTO ERR_TABLE
(PARAMETER,SQL_ERROR_CODE,SQL_ERROR_MESS
AGE) VALUES ( :p, :errCode, :errMsg);
END;
outtab = SELECT 1/:p as A FROM DUMMY; --
DIVIDE BY ZERO Error if p=0
END
```

Основные характеристики автономных транзакций:

- Появились в версии DB2 9.7:
  - Чтобы начать автономную транзакцию, необходимо указать ключевую фразу AUTONOMOUS TRANSACTION при использовании оператора CREATE PROCEDURE
  - Когда автономная транзакция фиксируется или откатывается, это не влияет на результат основной транзакции.
- Возможны взаимоблокировки(deadlocks) при использовании автономных транзакций
- Автономные транзакции могут запускать другие автономные транзакции.

Введение. “Баба Яга” в SAP HANA

```
CREATE OR REPLACE PROCEDURE log_actions
(msg VARCHAR2, event_date TIMESTAMP)
AUTHID DEFINER AS
PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
INSERT INTO table_tracking VALUES (msg,
event_date);
COMMIT;
END log_actions;
```

Основные характеристики автономных транзакций:

- Автономные транзакции не зависят от состояния или возможного расположения основной транзакции. Например:
  - Автономная транзакция не видит никаких изменений, внесенных основной транзакцией.
  - Когда автономная транзакция фиксируется или откатывается, это не влияет на результат основной транзакции.
- Изменения, которые производит автономная транзакция, видны другим транзакциям, как только эта автономная транзакция фиксируется.
- Автономные транзакции могут запускать другие автономные транзакции.

Введение. “Баба Яга” в MS SQL

```
BEGIN TRAN OuterTran
INSERT TABLE1
BEGIN "AUTONOMOUS" TRAN InnerTran
INSERT TABLE2
COMMIT "AUTONOMOUS" TRAN InnerTran
ROLLBACK TRAN OuterTran
```

Основные характеристики автономных транзакций:

- Автономные транзакций как таковых нет, но их можно эмулировать:
  - Доступно с версии SQL Server 2008.
  - Реализуется с помощью опции remote proc transaction promotion
- Когда remote proc transaction promotion равно false локальная транзакция не будет повышена до распределенной транзакции

PostgreSQL. А как у нас дела с Autonomous transactions (ATX)?

Автономные транзакции в PostgreSQL отсутствуют.

© Приключения домовёнка Кузи

PostgreSQL. Workaround-ы для реализации ATX

- Обходные пути эмуляции внутренними инструментами:
  - “Old school”, с помощью независимого соединения, с использованием расширения dblink
  - С помощью PL/Python + драйвер psycopg2
  - Экзотические, с помощью COPY + PROGRAM
- С помощью стороннего расширения pg_background, с использованием динамических фоновых процессов

PostgreSQL. Миграция АТХ из Oracle с помощью ora2pg

Автономные транзакции в PostgreSQL чаще всего требуются при миграции из Oracle с помощью ora2pg

- Опция AUTONOMOUS_TRANSACTION управляет преобразованием:
  - FALSE - использовать функцию, как обычную
  - TRUE - переводить в функцию-обертку с помощью расширения dblink или pg_background
- Опция PG_BACKGROUND - доступна с версии 9.5, если установлено расширение pg_background
- Опция DBLINK_CONN - использовать dblink расширение, как обертку для имитации АТХ (поведение по умолчанию)

ora2pg

Ora2Pg. Пример миграции ATX из Oracle с использованием dblink

```
CREATE PROCEDURE log_action (username
VARCHAR2, event_date DATE, msg VARCHAR2)
IS
   PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
   INSERT INTO table_tracking VALUES
(log_seq.nextval, username, event_date,
msg);
   COMMIT;
END log_action;
```

Исходная процедура в Oracle:

- Создает автономную транзакцию, используя директиву PRAGMA AUTONOMOUS_TRANSACTION
- Регистрирует в таблице table_tracking все действия, вне зависимости от результата транзакции

Ora2Pg. Пример миграции ATX из Oracle с использованием dblink

```
CREATE OR REPLACE FUNCTION
log_action_atx (
          username text, event_date
timestamp, msg text
) RETURNS VOID AS
$body$
BEGIN
   INSERT INTO table_tracking VALUES
(nextval('log_seq'), username,
event_date, msg);
END;
$body$
LANGUAGE PLPGSQL
```

Создается хранимая функция:

- К имени функции добавляется суффикс _atx
- INSERT практически идентичен исходному в Oracle

Ora2Pg. Пример миграции ATX из Oracle с использованием dblink

```
CREATE OR REPLACE FUNCTION log_action (
  username text, event_date timestamp, msg text
) RETURNS VOID AS
$body$
DECLARE
      v_conn_str  text := 'port=5432 dbname=testdb
         host=localhost user=pguser password=pgpass';
      v_query     text;
BEGIN
v_query := 'SELECT true FROM log_action_atx ( ' ||
quote_nullable(username) || ',' ||
quote_nullable(event_date) || ',' ||
quote_nullable(msg) || ' )';
PERFORM * FROM dblink(v_conn_str, v_query) AS p (ret boolean);

END;
$body$
LANGUAGE plpgsql SECURITY DEFINER;
```

Создается хранимая функция:

- Затем создается функция обертка с использованием dblink
- Существует также некоторая проблема безопасности: пароль подключения находится в коде в открытом виде.
- Производительность при таком подходе тоже не идеальна, так как нам по сути необходимо устанавливать каждый раз новое соединение
- Как рекомендация, можно использовать pgBouncer, чтобы сократить издержки на установку соединений, что существенно может повысить производительность.

Ora2Pg. Пример миграции ATX из Oracle с использованием pg_background

```
CREATE OR REPLACE FUNCTION log_action (
          username text, event_date timestamp, msg text
) RETURNS VOID AS
$body$
DECLARE
        v_query     text;
BEGIN
        v_query := 'SELECT true FROM log_action_atx ( ' ||
 quote_nullable(username) || ',' || quote_nullable(event_date) ||
',' || quote_nullable(msg) || ' )';
      PERFORM * FROM pg_background_result(
           pg_background_launch(v_query)
      ) AS p (ret boolean);
END;
$body$
LANGUAGE plpgsql SECURITY DEFINER;
```

Используем функции расширения pg_background:

- Функция log_action_atx остается аналогичной dblink реализации
- В ora2pg используется автоматическое преобразование, поэтому подход такой же как и для dblink
- Регистрирует в таблице table_tracking все действия, вне зависимости от результата транзакции с помощью фонового процесса

pg_background. Что еще предоставляет расширение?

- Если мы хотим дождаться результата автономной транзакции и использовать его:

```
SELECT * FROM
pg_background_result(pg_background_launch('SELECT
log_action(...)')) AS p (ret text);
```

- Если мы хотим позже использовать результат автономного транзакции, запущенной в фоновом режиме, нам нужно сохранить pid, возвращаемый функцией pg_background_launch():

```
SELECT INTO at_pid pg_background_launch …
... do something …
SELECT INTO at_result * FROM
pg_background_result(at_pid)
```

© Приключения домовёнка Кузи

pg_background. Еще больше примеров

```
postgres=# BEGIN;
BEGIN
postgres=# SELECT * FROM pg_background_result(
   pg_background_launch($$SELECT log_action(
   'Vadim','now','PG BootCamp Russia 2023')$$)
   ) AS p (result text);
                     result
------------------------------------------------
 Message inserted into table_tracking with id: 1
(1 row)

postgres=# ROLLBACK;
ROLLBACK

postgres=# SELECT * FROM table_tracking;
 id | username |         event_date         |           msg
----+----------+----------------------------+-------------------------
  1 | Vadim    | 2023-10-04 01:06:08.933085 | PG BootCamp Russia 2023
(1 row)
```

Описание последовательности действий:

- Открыли транзакцию с помощью оператора **BEGIN**
- Выполнили вызов функции логирования действий с помощью **pg_background_launch**
- Откатили транзакцию с помощью оператора **ROLLBACK**
- Запись в таблице **table_tracking** все равно присутствует

plpython3u. Реализация ATX с помощью psycopg2

```
CREATE EXTENSION plpython3u;

CREATE FUNCTION public.log_python(username text, event_date timestamp,
    msg text
)
 RETURNS void
 LANGUAGE plpython3u
AS $function$
  import psycopg2
  from datetime import datetime
  conn = psycopg2.connect(host='/var/run/postgresql', port=5432)
  conn.autocommit = True
  cur = conn.cursor()
  cur.execute("SELECT log_action(%s, %s, %s)", (username, event_date, msg))
$function$;
```

Описание последовательности действий:

- Создаем расширение **plpython3u**
- Устанавливаем соединение с помощью драйвера plpython3u
- Вызываем функцию **log_action** для записи в таблицу **table_tracking**

plpython3u. Проверяем работу АТХ

```
postgres=# BEGIN;
BEGIN
postgres=# SELECT * FROM log_python('Vadim'::text,
 '2023-10-05 10:00:00'::timestamp, 'PG BootCamp Russia 2023'::text);
 log_python
------------
(1 row)

postgres=# ROLLBACK;
ROLLBACK

postgres=# SELECT * FROM table_tracking;
 id | username |     event_date      |          msg
----+----------+---------------------+-------------------------
  3 | Vadim    | 2023-10-05 10:00:00 | PG BootCamp Russia 2023
(1 row)
```

Описание последовательности действий:

- Открыли транзакцию с помощью оператора **BEGIN**
- Выполнили вызов функции логирования действий с помощью **log_python**
- Откатили транзакцию с помощью оператора **ROLLBACK**
- Запись в таблице **table_tracking** все равно присутствует

Завершение Части 1. Промежуточные итоги

dblink

Плюсы:

- Есть в contrib модуле
- Надежное и проверенное решение
- Возможно использовать pgBouncer для повышения производительности
- Поддерживается нативно утилитой ora2pg при миграции из Oracle

Минусы:

- Создает дополнительные соединения к БД
- Пароль хранится в открытом виде
- Производительность не оптимальна, за счет необходимости в дополнительных коннектах

pg_background

Плюсы:

- Не требует создания дополнительных соединений
- Более высокая производительность(тесты далее в докладе)
- Более гибкое решение для ATX
- Более безопасный способ. Не требует хранения пароля.

Минусы:

- Установка дополнительного расширения
- При большом количестве транзакций возможна ошибка выделения сегмента общей памяти

plpython3u

Плюсы:

- Является расширением процедурного языка
- Более высокая производительность по сравнению с dblink
- Более безопасный способ. Не требует хранения пароля.

Минусы:

- Требует знаний pl/python
- Требует дополнительных настроек и полного доступа суперпользователя

ATX в ядре PostgreSQL?

Часть 2. Реализация автономных транзакций в PostgreSQL

Глоссарий: основные термины

Автономная сессия (AS) - сессия, которая группирует множественные связанные SQL команды в одну транзакцию. Когда выполняется функция с PRAGMA AUTONOMOUS_TRANSACTION, то создается автономная сессия, которая получает SQL команды от бекенда и выполняет их в автономной транзакции.

Основная транзакция (родительская) (BT)- в которой инициализируется автономная транзакция. Автономная транзакция (autonomous transaction) - независимая транзакция, запускаемая внутри родительской транзакции при работе автономной сессии.

Автономная функция (AF)- функция с PRAGMA AUTONOMOUS_TRANSACTION. В ней создается автономная сессия.

Фоновый воркер (background worker)(BGW) - фоновый процесс, который выполняет какие-то действия в фоне, без участия пользователя.

DSM - динамическая общая память (dynamic shared memory).

shm_mq - очередь сообщений с общей памятью (shared memory message queue).

Вопросы при реализации

- Синтаксис:
  - PRAGMA AUTONOMOUS_TRANSACTION в функциях и процедурах
  - BEGIN AUTONOMOUS в основной транзакции
- Видимость изменений между основной и автономной транзакциями
- Обнаружение взаимоблокировки (deadlock detection)
- Как передавать данные между основной и автономной транзакциями?
- Выполнение АТ по отношению к ОТ (основной транзакции):
  - синхронное
  - асинхронное
- Обработка исключений в автономных транзакциях
- Обработку в АТ расширенного протокола запросов
- Вложенность вызовов АТ
- Объём и опасность рефакторинга существующего кода ядра
- Логи
- Тесты

1-ые обсуждения в pgsql-hackers (без кода)

1) 2008
https://www.postgresql.org/message-id/flat/1A6E6D554222284AB25ABE3229A9276271549A%40nrtexcus702.int.asurion.com

2) 2010
https://www.postgresql.org/message-id/flat/AANLkTi%3DuogmYxLKWmUfFSg-Ki2bejsQiO2g5GTMxvdW2%40mail.gmail.com

3) 2011
https://www.postgresql.org/message-id/flat/1303399444.9126.8.camel%40vanquo.pezone.net

4) 2011
https://wiki.postgresql.org/wiki/Autonomous_subtransactions

5) 2011
https://www.postgresql.org/message-id/flat/20111218082812.GA14355%40leggeri.gi.lan
https://wiki.postgresql.org/wiki/Autonomous_subtransactions

Патч Rajeev

В 2014 году Rajeev Rastogi
https://www.postgresql.org/message-id/flat/BF2827DCCE55594C8D7A8F7FFD3AB7713DDDEF59%40SZXEML508-MBX.china.huawei.com

- PL/pgSQL: PRAGMA AUTONOMOUS_TRANSACTION

- В 2015 новая тема, продолжение обсуждения семантики поведения AT и их синтаксис
https://www.postgresql.org/message-id/flat/BF2827DCCE55594C8D7A8F7FFD3AB7715990499A%40szxeml521-mbs.china.huawei.com

Патч Peter

В 2016 году Peter Eisentraut
https://www.postgresql.org/message-id/flat/659a2fce-b6ee-06de-05c0-c8ed6a01979e%402ndquadrant.com

- PL/Python и PL/pgSQL
  - PL/pgSQL: PRAGMA AUTONOMOUS_TRANSACTION
  - PL/Python: plpy.autonomous()
- Реализация с помощью BGW
- Синхронное выполнение
- Клиент-серверный протокол Postgres
- На каждый вызов AF с нуля создаётся BGW
- По завершении выполнения AF он уничтожается
- Допустимы бесконечные вложенные вызовы AF
- Если в AF вызывается другая AF, в начале синхронно выполняется 2-ая, а потом довыполняется 1-я.

Синтаксис

- PL/pgSQL: **PRAGMA AUTONOMOUS_TRANSACTION**

```
AS $$
DECLARE
  PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
  FOR i IN 0..9 LOOP
    START TRANSACTION;
    INSERT INTO test1 VALUES (i);
    IF i % 2 = 0 THEN
        COMMIT;
    ELSE
        ROLLBACK;
    END IF;
  END LOOP;
  RETURN 42;
END;
$$;
```

Синтаксис

- PL/Python: **plpy.autonomous()**

```
with plpy.autonomous() as a:
    for i in range(0, 10):
        a.execute("BEGIN")
        a.execute("INSERT INTO test1 (a) VALUES (%d)" % i)
        if i % 2 == 0:
            a.execute("COMMIT")
        else:
            a.execute("ROLLBACK")
```

Патч Peter: Преимущества

- Не нужен рефакторинг существующего кода (на момент 2016г)
- Общепринятая прагма AUTONOMOUS_TRANSACTION
- Ревьюеры не против!
- Функционал работает!

Патч Peter: Недостатки

- Допустимы только в функциях и процедурах через прагму

  => отдельные блоки тоже неплохо бы иметь, вне функций
- На каждый вызов AF с нуля создаётся BGW. По завершении выполнения он уничтожается.

  => сделать пул
- Существует лимит на количество BGW, определяется настройкой.

  => ничего не поделать, если реализация через BGW
- Переключение контекста между процессами

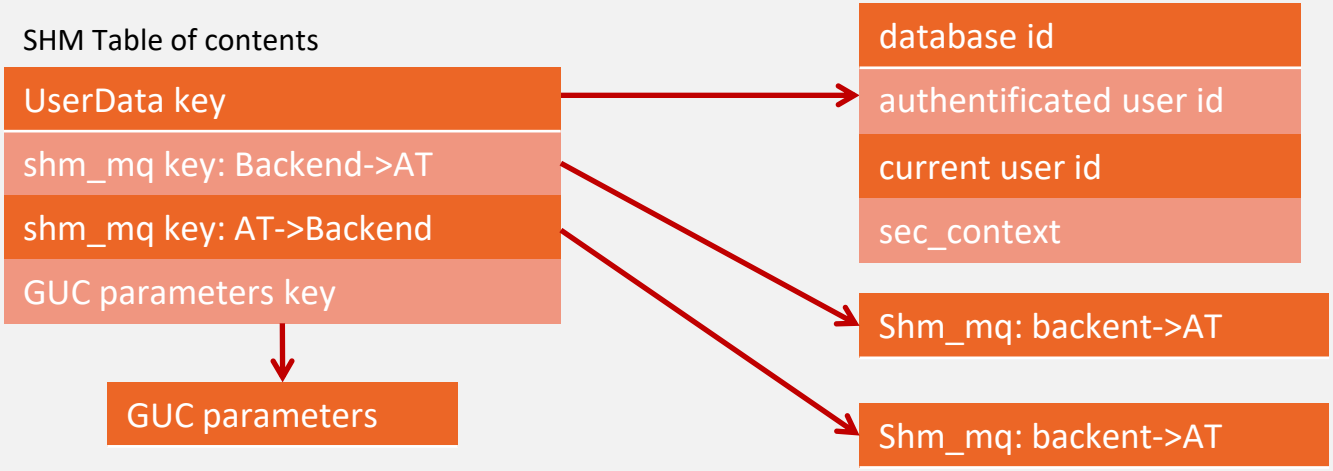
  => ничего не поделать, если реализация через BGW
- Незначительные недоработки
- Адаптирован для версии PostgreSQL 9.6

План действий

- На основе идей Peter-a реализовать автономные транзакции в PostgreSQL 15
  - развить функциональные возможности
  - предусмотреть больше тестов
  - повысить производительность
- Реализовать пул автономных сессий
- Предусмотреть отложенную инициализацию сессий, по мере необходимости
- Поддержать уничтожение или пересоздание сессии
- Предусмотреть настройки пула воркеров в postgresql.conf:
  - время жизни сессий в пуле
  - размера пула
  - и т.п.
- Добавить автономные сессии в PL/Python

Создание Автономной сессии

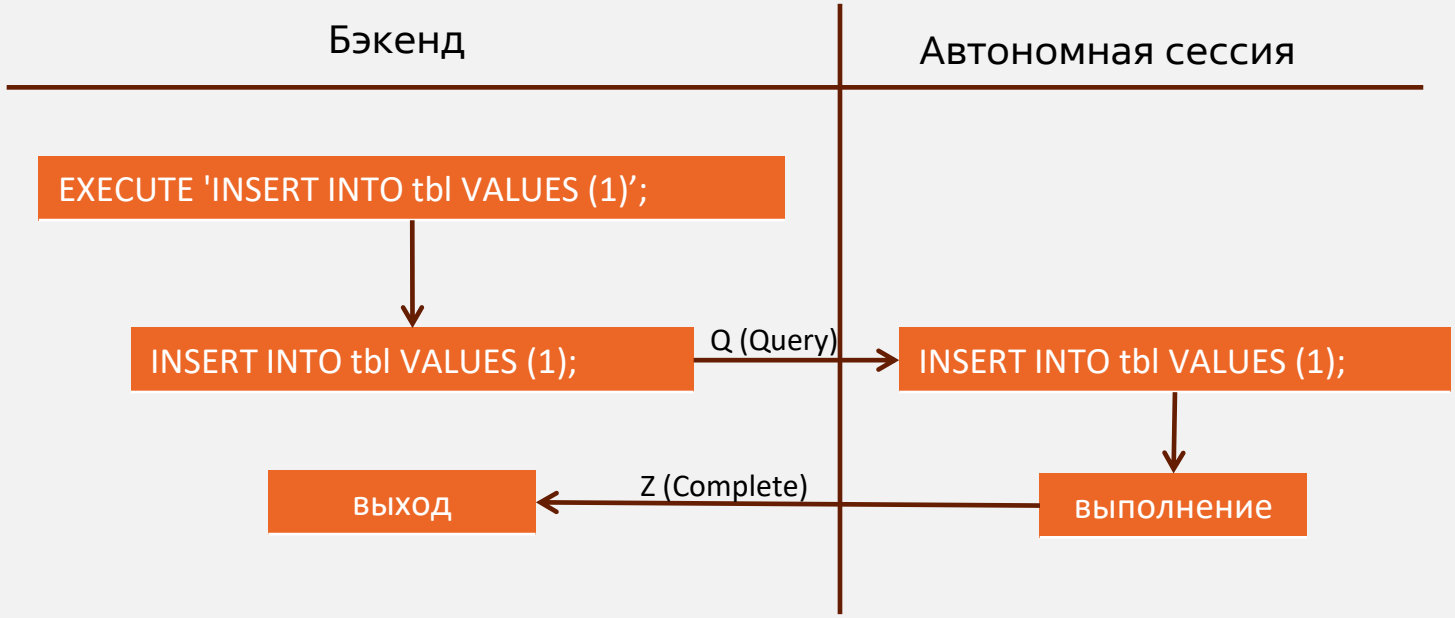
1. Формирование сегмента **Dynamic Shared Memory**



2. Создание BGW с **функцией-обработчиком** -> **Autonomous Session**
3. **Autonomous Session** извещает бэкенд, что готова выполнять SQL запросы

Обработка SQL-запроса

```
CREATE FUNCTION autonomous_test() RETURNS void
LANGUAGE plpgsql
AS $$
DECLARE
  PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
  EXECUTE 'INSERT INTO my_table0 VALUES (1)';
END
$$;
```



Отдельные транзакции

```
CREATE TABLE tbl (a integer CHECK (a > 500));

CREATE FUNCTION sep_trans()
RETURNS VOID
AS $$
DECLARE
  PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
  INSERT INTO tbl(a) VALUES(700);    -- 1ая автономная транзакция
  INSERT INTO tbl(a) VALUES(100);    -- 2ая автономная транзакция
END
$$ LANGUAGE plpgsql;

select sep_trans();
>>
    ERROR:  Failing row contains (100).new row for relation "tbl"
    violates check constraint "tbl_a_check"

select * from tbl
>>
    700
```

Единый блок транзакции

```
TRUNCATE TABLE tbl;

CREATE FUNCTION one_trans()
RETURNS VOID
AS $$
DECLARE
  PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
  START TRANSACTION; -- одна автономная транзакция
    INSERT INTO tbl(a) VALUES(800);
    INSERT INTO tbl(a) VALUES(200);
  COMMIT;
END
$$ LANGUAGE plpgsql;

SELECT one_trans();
>>
    ERROR:  Failing row contains (200).new row for relation "tbl"
    violates check constraint "tbl_a_check"

select * from tbl
>>
    none
```

Вызов нетранзакционных команд

Запрещены к выполнению в функции:

- VACUUM
- ALTER SYSTEM
- DISCARD/LISTEN/UNLISTEN
- REINDEX/CLUSTER
- **CREATE/ALTER/DROP** ROLE/DATABASE/TABLESPACE/SUBSCRIPTION
- и т.д.

```
DO $$
BEGIN
   EXECUTE 'VACUUM VERBOSE tbl';
END $$;
>>
      ERROR:  VACUUM cannot be executed from a function


DO $$
DECLARE
  PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
   EXECUTE 'VACUUM VERBOSE tbl';
END $$;
>>
      INFO: vacuuming "test.public.tbl"
```

Пример обработки исключения

```
CREATE OR REPLACE FUNCTION f_exception ()
RETURNS VOID AS $$
DECLARE
      PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
      raise notice 'Call f_exception';
      raise exception 'Test exception' USING ERRCODE = 'AB001';
END;
$$
LANGUAGE plpgsql;
```

Пример обработки исключения

```
CREATE OR REPLACE FUNCTION f_main ()
RETURNS VOID AS $$
BEGIN
  BEGIN
    PERFORM f_exception();
  EXCEPTION
    WHEN OTHERS THEN
      IF SQLSTATE = 'AB001' THEN
        RAISE NOTICE 'Caught custom exception with SQLSTATE AB001.';
      ELSE
        RAISE NOTICE 'An exception occurred: %', SQLERRM;
      END IF;
  END;
END;
$$
LANGUAGE plpgsql;
```

```
select * from f_main();
>>
     NOTICE:  Call f_exception
     NOTICE:  Caught custom exception with SQLSTATE AB001.
```

Особенности хранимых процедур

```
truncate table tbl;

CREATE PROCEDURE p2()
AS $$
DECLARE
BEGIN
    INSERT INTO tbl(a) VALUES(700);
    COMMIT;

    INSERT INTO tbl(a) VALUES(800);
    ROLLBACK;
END
$$ LANGUAGE plpgsql;

CALL p2();

SELECT * FROM tbl;
>>
    700
```

Вызов процедуры, управляющей транзакциями, из функции

```
CREATE FUNCTION f1 ()
RETURNS VOID AS $$
BEGIN
  CALL p2();
END
$$ LANGUAGE plpgsql;

SELECT * FROM f1();
>>
     ERROR:  invalid transaction termination
     CONTEXT:  PL/pgSQL function p2() line 5 at COMMIT
     SQL statement "call p2()"
```

```
begin;
INSERT INTO tbl(a) VALUES(900);
CALL p2();
COMMIT;
>>
     ERROR:  invalid transaction termination
     CONTEXT:  PL/pgSQL function p2() line 5 at COMMIT
```

Ограничения хранимых процедур

- Транзакции выполняются последовательно
- Процедуры не могут полноценно возвращать значения, они только выполняют операции (можно использовать INOUT, подходит для всех типов, кроме **набора_записей** SETOF)
- Хранимые **процедуры** вызываются с помощью команды CALL, в то время как **функции** вызываются с использованием SELECT или как часть другого выражения (напрямую в SQL запросы подставить результат вызова хранимой процедуры невозможно)
- Вызов **процедуры**, управляющей транзакциями, из **функции** недоступен

Тоже самое с нетранзакционными запросами

```
CREATE OR REPLACE PROCEDURE p2()
AS $$
DECLARE
BEGIN
      execute 'vacuum verbose tbl';
END
$$ LANGUAGE plpgsql;

call p2()
>>
          ERROR:  VACUUM cannot be executed from a function
          CONTEXT:  SQL statement "vacuum verbose tbl"
```

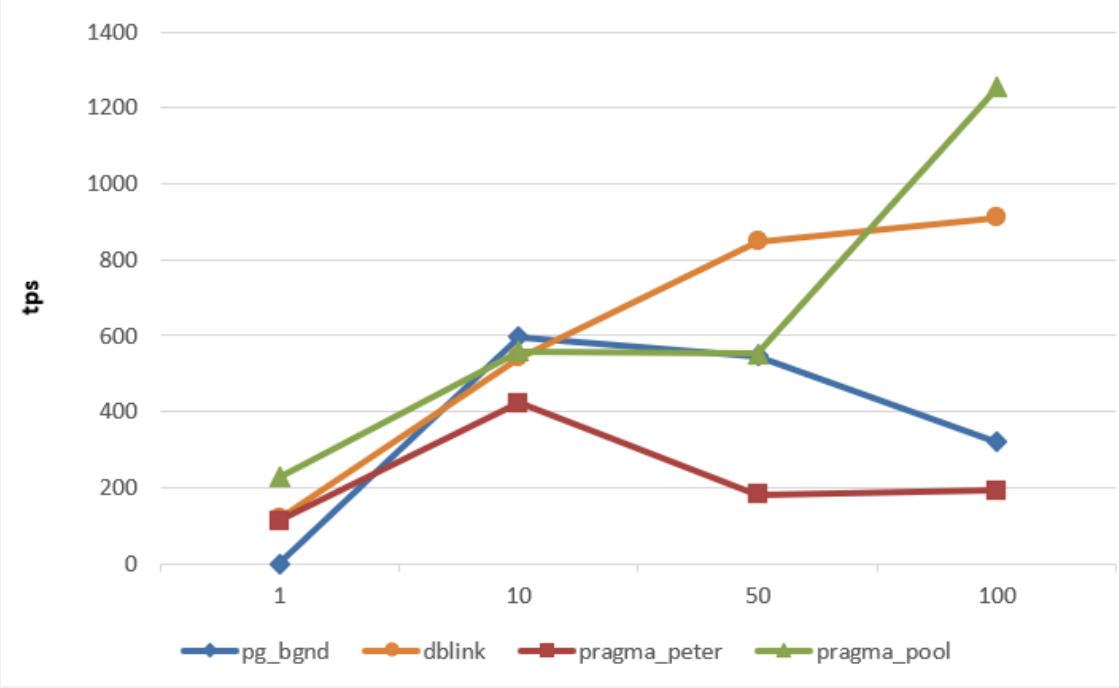
Какие возможности открывают автономные транзакции

- Позволяют делать то же самое, что и процедуры (по части управления транзакциями)
- Выполнять нетранзакционные команды внутри функций
- Реализация сложной логики, включающей возможность "выглянуть" за пределы текущей транзакции без использования dblink
- Высокопроизводительная альтернатива dblink и pg_background

Бенчмарки

- **pragma_pool** - прагма, реализация через пул потоков
- **pragma_peter** - прагма, патч Peter Eisentraut
- **dblink** - реализация через dblink
- **pg_bgnd** – реализация через pg_background

```
SELECT log_action(
         'some text',
         now(),
         'autonomous_transaction');
```



Бенчмарки: PRAGMA AUTONOMOUS_TRANSACTION

```
DROP TABLE IF EXISTS table_tracking;

CREATE TABLE table_tracking (id integer, username text, event_date timestamp, msg text);

CREATE SEQUENCE log_seq START 1;

CREATE OR REPLACE FUNCTION log_action_atx (
        username text, event_date timestamp, msg text
) RETURNS VOID AS
$body$
DECLARE
   PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
   INSERT INTO table_tracking VALUES (nextval('log_seq'), username, event_date, msg);
END;
$body$
LANGUAGE PLPGSQL;
```

Бенчмарки: PRAGMA AUTONOMOUS_TRANSACTION

```
CREATE OR REPLACE FUNCTION log_action (
        username text, event_date timestamp, msg text
) RETURNS VOID AS
$body$
BEGIN
       PERFORM * FROM log_action_atx(username, event_date, msg);
END;
$body$
LANGUAGE plpgsql;


-- Запрос выполняемый в pgbench
SELECT log_action('some text', now()::timestamp, 'autonomous_transaction');
```

Запрос использовался для (названия легенд):

- **pragma_pool** - прагма, реализация через пул потоков
- **pragma_peter** - прагма, патч Peter Eisentraut

Бенчмарки: dblink

Изменена только реализация log_action_atx:

```
CREATE OR REPLACE FUNCTION log_action_atx (
        username text, event_date timestamp, msg text
) RETURNS VOID AS
$body$
DECLARE
        v_conn_str  text := 'port=5432 dbname=at_dblink host=localhost user=postgres';
        v_query     text;

BEGIN
        v_query := 'SELECT true FROM log_action_atx ( ' || quote_nullable(username) ||
                ',' || quote_nullable(event_date) || ',' || quote_nullable(msg) || ' )';
        PERFORM * FROM dblink(v_conn_str, v_query) AS p (ret boolean);
END;
$body$
LANGUAGE PLPGSQL;
```

- **dblink** - реализация через dblink

Бенчмарки: pg_background

Изменена только реализация log_action_atx:

```
CREATE OR REPLACE FUNCTION log_action_atx (
        username text, event_date timestamp, msg text
) RETURNS VOID AS
$body$
DECLARE
        v_query     text;
BEGIN
        v_query := 'SELECT true FROM log_action_atx ( ' || quote_nullable(username) || ',' ||
quote_nullable(event_date) || ',' || quote_nullable(msg) || ' )';
        PERFORM * FROM pg_background_result(pg_background_launch(v_query)) AS p (ret boolean);
END;
$body$
LANGUAGE PLPGSQL;
```

- pg_bgnd - реализация через pg_background

Дальнейшие планы

- Улучшить логирование
- Обкатать реализацию на большем количестве случаев
- Отправить патч на CommitFest
- Пройти все ревью и вмёржить в стандартный Postgres

Вопросы и замечания?

Спасибо за внимание!

tantor