Автономные транзакции PostgresPro Enterprise

Шурутов Михаил
m.shurutov@postgrespro.ru

Краткое содержание доклада

- Автономные транзакции: определение и область применения
- Автономные транзакции: реализация в СУБД основных производителей - Oracle, IBM DB2, SAP HANA
- Автономные транзакции: реализация в PostgresPro Enterprise
- Автономные транзакции: сравнение функциональных возможностей
- "Автономные" транзакции PostgreSQL
- Автономные транзакции: сравнение производительности

Автономные транзакции: определение и область применения

Автономная транзакция (ATX) - это независимая транзакция, выполняющаяся в собственном контексте, результат выполнения которой `(COMMIT/ROLLBACK)` не зависит от результата выполнения вызывающей транзакции `(COMMIT/ROLLBACK)`.

Область применения

- реализация интеграции между независимыми системами
- реализация аудита и журналирования
- реализация отладки хранимых процедур

Автономные транзакции основных производителей СУБД

Oracle

Автономные транзакции в Oracle определяются с помощью директивы: `PRAGMA AUTONOMOUS_TRANSACTION;`

Листинг 1: Автономная транзакция в Oracle

```
CREATE OR REPLACE PROCEDURE lower_salary
  (emp_id NUMBER, amount NUMBER)
AUTHID DEFINER AS
  PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
  UPDATE employees
  SET salary =  salary - amount
  WHERE employee_id = emp_id;

  COMMIT;
END lower_salary;
/
```

Автономные транзакции основных производителей СУБД

IBM DB2

Автономные транзакции в IBM DB2 имеют следующий вид:

Листинг 2: Автономная транзакция в IBM DB2

```
CREATE OR REPLACE PROCEDURE
   log_query (in queryingEmployee varchar(100),
              in accNumber integer, in when timestamp)
LANGUAGE SQL
AUTONOMOUS
BEGIN
   INSERT INTO log_table values (queryingEmployee, accNumber, when);
END %

COMMIT %
```

Автономные транзакции основных производителей СУБД

SAP HANA

Пример использования автономной транзакции в SAP HANA:

Листинг 3: Автономная транзакция в SAP HANA

```
CREATE PROCEDURE PROC1( 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_MESSAGE)
                        VALUES ( :p, :errCode, :errMsg);
        END;
        outtab = SELECT 1/:p as A FROM DUMMY;    -- DIVIDE BY ZERO Error if p=0
END
```

Синтаксис: SQL

Ниже показан синтаксис ATX в "**чистом**" SQL.

Листинг 4: Автономные транзакции в PostgreSQL. SQL.

```
-- Any SQL operators
BEGIN;
        -- Any SQL operators include other [autonomous] transactions
        BEGIN AUTONOMOUS [TRANSACTION] [ISOLATION LEVEL];
                -- Any of SQL operators include other [autonomous] transactions
        [COMMIT|ROLLBACK|END] [AUTONOMOUS] [TRANSACTION];
[COMMIT|ROLLBACK|END];
-- Any SQL operators
```

Автономная транзакция в "чистом" SQL должна вызываться из обычной транзакции, т.е. блок `BEGIN AUTONOMOUS; ... END;` должен быть вложен в блок `BEGIN; ... END;`, как показано в листинге выше.

SQL: Пример READ COMMITTED транзакции

```
test@test=> insert into temp (msg) values ('init record');
INSERT 0 1
test@test=> BEGIN;
BEGIN
test@test=> insert into temp (msg) values ('1-st record');
INSERT 0 1
test@test=> BEGIN AUTONOMOUS;
BEGIN
test@test=> insert into temp (msg) values ('2-nd record: ATX');
INSERT 0 1
test@test=> COMMIT;
COMMIT
test@test=> select * from temp;
 id |       msg
----+-----------------
  1 | init record
  2 | 1-st record
  3 | 2-nd record: ATX
(3 rows)

test@test=> ROLLBACK;
ROLLBACK
test@test=> select * from temp;
 id |       msg
----+-----------------
  1 | init record
  3 | 2-nd record: ATX
(2 rows)
```

SQL: Пример REPEATABLE READ транзакции

```
test@test=> BEGIN TRANSACTION ISOLATION LEVEL REPEATABLE READ;
BEGIN
test@test=> select * from temp;
 id |     msg
----+-------------
  1 | init record
(1 row)

test@test=> BEGIN AUTONOMOUS;
BEGIN
test@test=> insert into temp (msg) values ('1-st record: ATX');
INSERT 0 1
test@test=> COMMIT;
COMMIT
test@test=> select * from temp;
 id |     msg
----+-------------
  1 | init record
(1 row)

test@test=> ROLLBACK;
ROLLBACK
test@test=> select * from temp;
 id |       msg
----+------------------
  1 | init record
  2 | 1-st record: ATX
(2 rows)
```

Синтаксис: PL/PGSQL

Пример применения автономной транзакции в функции.

Листинг 5: Автономные транзакции в PostgreSQL. PL/PGSQL.

```
CREATE OR REPLACE FUNCTION test01_insert(newname name, newsum int)
RETURNS void AS $ti$
DECLARE
        tid int;
BEGIN
        tid:=nextval('test01_test01_id_seq');
        BEGIN AUTONOMOUS
                INSERT INTO test01_audit (test01_id,uname,sum,what_do)
                        VALUES (tid,newname,newsum,'INSERT');
        END;
        INSERT INTO test01 (test01_id,uname,sum) VALUES (tid,newname,newsum);
        EXCEPTION
                WHEN unique_violation THEN
                        RAISE EXCEPTION 'User % is exists', newname;
END;
$ti$ LANGUAGE plpgsql;
```

Синтаксис: PL/PGSQL. Неименованный блок DO

Пример применения вложенной автономной транзакции в неименованном блоке DO.

Листинг 6: Автономные транзакции в PostgreSQL. PL/PGSQL. DO.

```
DO $TU$
DECLARE newmsg text;
        tid int;
        currname name;
BEGIN
        tid:=nextval('test01_test01_id_seq');
        newmsg='dfjhvwljbvwebvkjebvwejbv';
        currname=current_user;
        BEGIN AUTONOMOUS
                INSERT INTO test_audit (test_id,uname,msg,op)
                        VALUES (tid,currname,newmsg,'INSERT');
        END;
        INSERT INTO test(test_id,msg) VALUES(tid,newmsg);
        EXCEPTION
                WHEN unique_violation THEN
                        RAISE EXCEPTION 'User % is exist', newname;
END $TU$;
```

Сравнение функциональных возможностей

Автономные транзакции основных производителей и PostgresPro Enterprise

	Oracle	IBM DB2	SAP HANA	PgPro Enterprise
Автономные процедуры, функции, триггеры и неименованные блоки	да	да	да	да
Автономные блоки в теле программных единиц	нет	нет	да	да
Автономные транзакции в "чистом" SQL	нет	нет	нет	да

"Автономные" транзакции PostgreSQL

"Автономные" транзакции PostgreSQL реализуются с помощью расширения dblink. Плюсы и минусы этого решения:

- **Минусы**
  - необходимость дополнительного конфигурирования сервера: аутентификация и количество соединений
  - серьезное падение производительности на создание дополнительного подключения
- **Плюсы**
  - не обнаружено

Сравнение производительности

Условия проведения тестирования

- Процессор: Intel(R) Core(TM) i5-4690K CPU @ 3.50GHz, 4 ядра
- Память: 16 Гб
- Диск: 3 Гб в tmpfs
- Тесты: набор из pgbech с изменением вставки в pgbench_history
- Коэффициент масштабирования при инициализации БД: 10

Сравнение производительности

Условия проведения тестирования: исходный скрипт

```
\set aid random(1, 100000 * :scale)
\set bid random(1, 1 * :scale)
\set tid random(1, 10 * :scale)
\set delta random(-5000, 5000)
BEGIN;
UPDATE pgbench_accounts SET abalance = abalance + :delta WHERE aid = :aid;
SELECT abalance FROM pgbench_accounts WHERE aid = :aid;
UPDATE pgbench_tellers SET tbalance = tbalance + :delta WHERE tid = :tid;
UPDATE pgbench_branches SET bbalance = bbalance + :delta WHERE bid = :bid;
-- Begin of code for replace
INSERT INTO pgbench_history (tid, bid, aid, delta, mtime)
        VALUES (:tid, :bid, :aid, :delta, CURRENT_TIMESTAMP);
-- End of code for replace
END;
```

Сравнение производительности

dblink

Вставка в таблицу pgbench_history тестов pgbench с помощью dblink:

```
SELECT dblink_exec('hostaddr=127.0.0.1 dbname=test user=test password=testpass',
        'INSERT INTO pgbench_history (tid, bid, aid, delta, mtime)
                VALUES (:tid, :bid, :aid, :delta, CURRENT_TIMESTAMP);');
```

```
scaling factor: 10
query mode: simple
number of clients: 8
number of threads: 1
duration: 20 s
number of transactions actually processed: 4481
latency average = 35.797 ms
tps = 223.484718 (including connections establishing)
tps = 223.510504 (excluding connections establishing)
```

Сравнение производительности

dblink+pgbouncer

Вставка в таблицу pgbench_history тестов pgbench с помощью dblink+pgbouncer:

```
SELECT dblink_exec('hostaddr=127.0.0.1 port=6432 dbname=test user=test password=testpass',
        'INSERT INTO pgbench_history (tid, bid, aid, delta, mtime)
                VALUES (:tid, :bid, :aid, :delta, CURRENT_TIMESTAMP);');
```

```
scaling factor: 10
query mode: simple
number of clients: 8
number of threads: 1
duration: 20 s
number of transactions actually processed: 1559
latency average = 7.172 ms
tps = 1115.515613 (including connections establishing)
tps = 1117.799104 (excluding connections establishing)
```

Сравнение производительности

dblink+pgbouncer+постоянное соединение

Вставка в таблицу pgbench_history тестов pgbench:

```
DO $DL$
<<dbl>>
DECLARE isconn int; connstr text; qstr text;
BEGIN
        connstr = 'hostaddr=127.0.0.1 port=6432 dbname=test user=test password=testpass';
        qstr = 'INSERT INTO pgbench_history (tid, bid, aid, delta, mtime)
                VALUES (:tid, :bid, :aid, :delta, CURRENT_TIMESTAMP);';
        SELECT coalesce INTO isconn
        FROM COALESCE(array_position(dblink_get_connections(),'dblink_test'),0);
        IF isconn = 0 THEN PERFORM dblink_connect('dblink_test',dbl.connstr);
        END IF;
        PERFORM dblink_exec('dblink_test',dbl.qstr);
END $DL$;
```

```
scaling factor: 10
query mode: simple
number of clients: 8
number of threads: 1
duration: 20 s
number of transactions actually processed: 87830
latency average = 1.823 ms
tps = 4388.209720 (including connections establishing)
tps = 4388.807001 (excluding connections establishing)
```

Сравнение производительности

PostgresPro Enterprise: "чистый"SQL

```
scaling factor: 10
query mode: simple
number of clients: 8
number of threads: 1
duration: 20 s
number of transactions actually processed: 137569
latency average = 1.164 ms
tps = 6871.881169 (including connections establishing)
tps = 6872.825409 (excluding connections establishing)
```

Сравнение производительности

PostgresPro Enterprise: "чистый"SQL + ATX

Различия в скриптах по сравнению с "чистым" SQL - вставка в табличку `pgbench_history` выполняется в автономной транзакции:

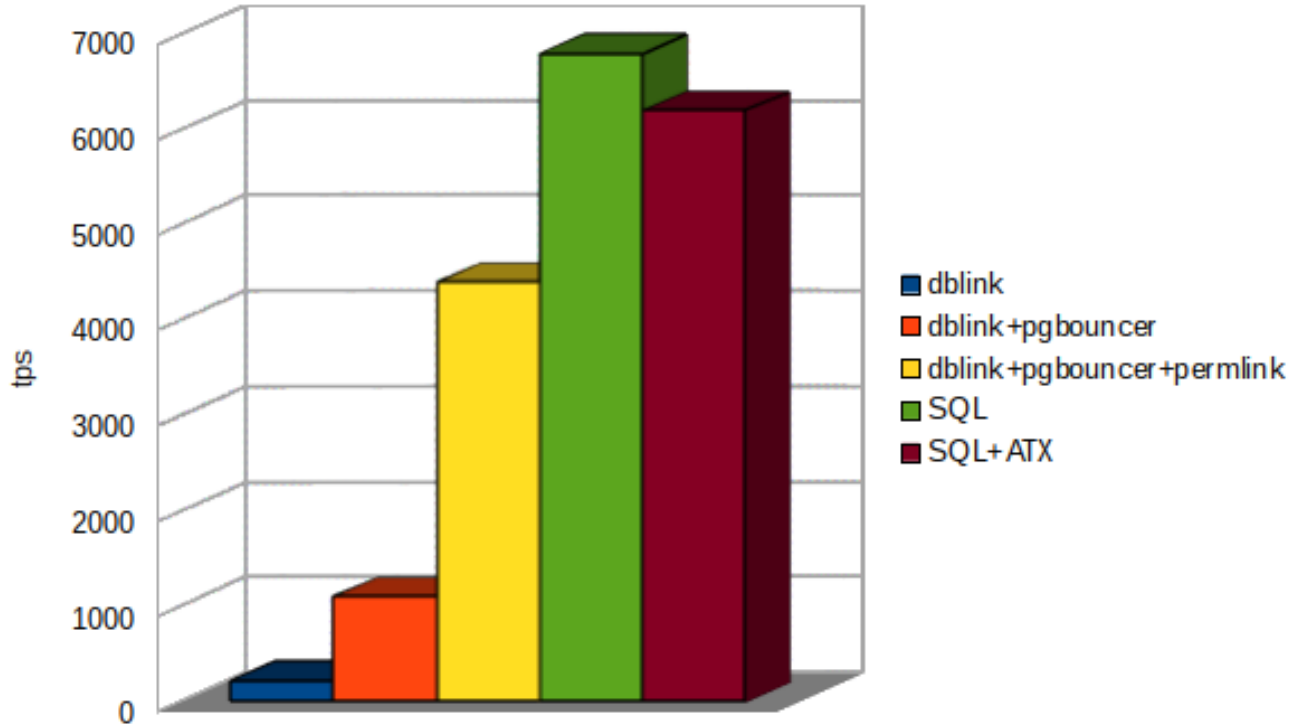
```
BEGIN AUTONOMOUS;
INSERT INTO pgbench_history (tid, bid, aid, delta, mtime)
        VALUES (:tid, :bid, :aid, :delta, CURRENT_TIMESTAMP);
END;
```

Скорость работы SQL с ATX:

```
scaling factor: 10
query mode: simple
number of clients: 8
number of threads: 1
duration: 20 s
number of transactions actually processed: 125044
latency average = 1.280 ms
tps = 6248.292318 (including connections establishing)
tps = 6249.267844 (excluding connections establishing)
```

Сравнение производительности

Результаты



dblink	220 tps
dblink+pgbouncer	1100 tps
dblink+pgbouncer+permanent link	4400 tps
SQL	6800 tps
SQL+ATX	6200 tps

Postgres PROFESSIONAL

Вопросы?

Список использованных источников

Oracle:
http://docs.oracle.com/database/122/LNPLS/static-sql.htm#LNPLS616
IBM DB2:
https://www.ibm.com/developerworks/data/library/techarticle/dm-0907autonomoustransactions/
SAP HANA:
http://help-legacy.sap.com/saphelp_hanaplatform/helpdata/en
/4a/d70daee8b64b90ab162565ed6f73ef/frameset.htm PostgresPro Enterprise:
https://postgrespro.ru/docs/postgresproee/9.6/atx.html
dblink:
https://www.postgresql.org/docs/9.6/static/dblink.html