Stored Procedures

What Are They Good For

Peter Eisentraut

2ndQuadrant®
PostgreSQL

peter.eisentraut@2ndquadrant.com
@petereisentraut

```
CREATE PROCEDURE new_customer(name text, address text)
LANGUAGE plpgsql
AS $$
BEGIN
    INSERT INTO customers VALUES (name, address);
END
$$;

CALL new_customer('somename', 'someaddress');
```

```
CREATE PROCEDURE convert_to_upper(INOUT string text)
LANGUAGE plpgsql
AS $$
BEGIN
    string := upper(string);
END
$$;

CALL convert_to_upper('abc');
 string
--------
 ABC
(1 row)
```

```
CREATE PROCEDURE new_customer(name text, address text)
LANGUAGE plperl
AS $$
$plan = spi_prepare('INSERT INTO customers VALUES ($1, $2)');
spi_exec_prepare($plan, $_[0], $_[1]);
$$;

CALL new_customer('somename', 'someaddress');
```

```
CREATE PROCEDURE transaction_test1(x int, y text)
LANGUAGE plpgsql
AS $$
BEGIN
    FOR i IN 0..x LOOP
        INSERT INTO test1 (a, b) VALUES (i, y);
        IF i % 2 = 0 THEN
            COMMIT;
        ELSE
            ROLLBACK;
        END IF;
    END LOOP;
END
$$;

CALL transaction_test1(9, 37);
```

```
CREATE PROCEDURE transaction_test1()
LANGUAGE plperl
AS $$
foreach my $i (0..9) {
    spi_exec_query("INSERT INTO test1 (a) VALUES ($i)");
    if ($i % 2 == 0) {
        spi_commit();
    } else {
        spi_rollback();
    }
}
$$;
```

```
CREATE PROCEDURE transaction_test1()
LANGUAGE plr
AS $$
for(i in 0:9){
  pg.spi.exec(paste("INSERT INTO test1 (a) VALUES (", i, ")"))
  if (i %% 2 == 0) {
    pg.spi.commit()
  } else {
    pg.spi.rollback()
  }
}
$$;
```

https://github.com/petere/plr/tree/procedure-transaction

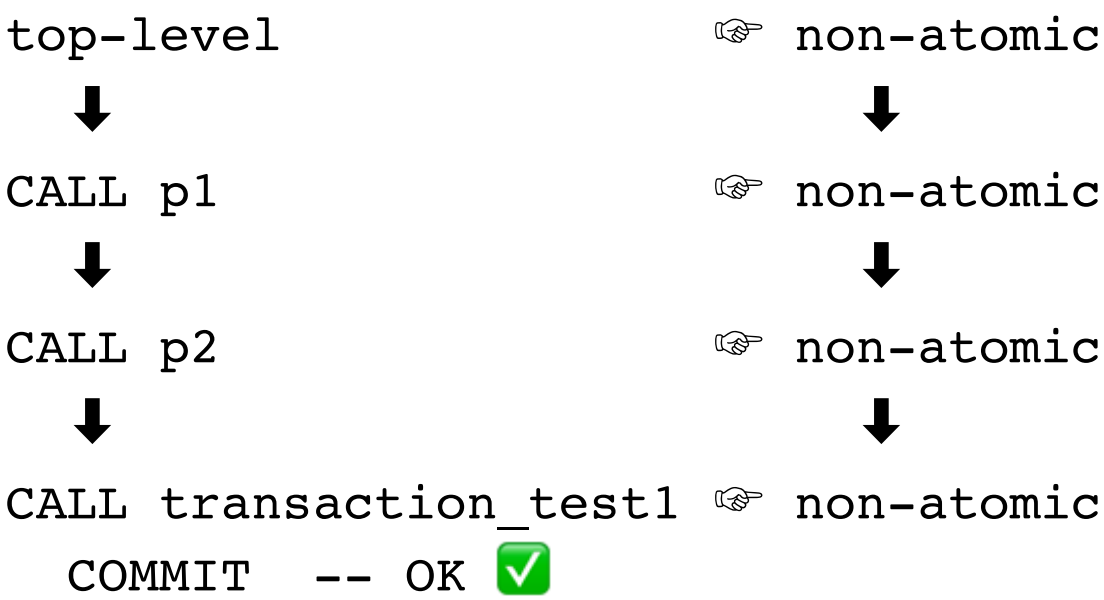
```
DO
LANGUAGE plpgsql
$$
BEGIN
    FOR i IN 0..9 LOOP
        INSERT INTO test1 (a) VALUES (i);
        IF i % 2 = 0 THEN
            COMMIT;
        ELSE
            ROLLBACK;
        END IF;
    END LOOP;
END
$$;
```

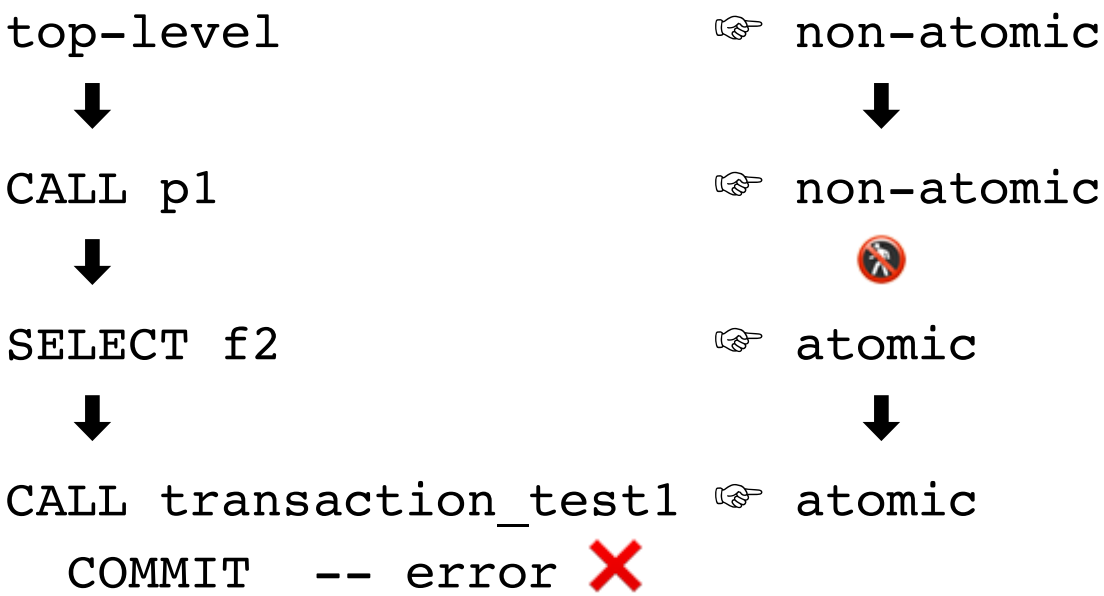
```
CREATE PROCEDURE transaction_test2()
LANGUAGE plpgsql
AS $$
DECLARE
    r RECORD;
BEGIN
    FOR r IN SELECT * FROM test2 ORDER BY x LOOP
        INSERT INTO test1 (a) VALUES (r.x);
        COMMIT;
    END LOOP;
END;
$$;
```

```
CREATE PROCEDURE p1() LANGUAGE plpgsql
    AS $$ BEGIN CALL p2(); END $$;

CREATE PROCEDURE p2() LANGUAGE plpgsql
    AS $$ BEGIN CALL transaction_test1(9, 37); END $$;

CALL p1();
```

```
CREATE PROCEDURE p1() LANGUAGE plpgsql
    AS $$ BEGIN SELECT f2(); END $$;

CREATE FUNCTION f2() LANGUAGE plpgsql
    AS $$ BEGIN CALL transaction_test1(9, 37); END $$;

CALL p1();
```

(non)atomic execution context


top-level
☞ non-atomic
CALL p1
☞ non-atomic
CALL p2
☞ non-atomic
CALL transaction_test1
☞ non-atomic
COMMIT -- OK ✅


top-level
non-atomic
CALL p1
non-atomic
SELECT f2
atomic
CALL transaction_test1
atomic
COMMIT -- error

other stuff that disallows transaction control

- transaction block
- GUC settings attached to procedure
- security definer
- cursor loop not read-only
- inside subtransaction (block with exception handling)

other stuff that doesn't work (yet)

- procedures with OUT parameters
- VACUUM etc. inside procedures
- support in other PLs

use cases

- porting (Oracle, DB2)
- batch stuff
- audit logging

```
CREATE PROCEDURE batch_geocode()
LANGUAGE plpgsql
AS $$
BEGIN
  WHILE EXISTS (SELECT 1 FROM addr_to_geocode WHERE pt IS NULL) LOOP
    WITH a AS (SELECT addid, address FROM addr_to_geocode WHERE pt IS NULL
                ORDER BY addid LIMIT 5 FOR UPDATE SKIP LOCKED)
    UPDATE addr_to_geocode SET pt = ST_SetSRID(g.geomout, 4326)::geography
      FROM (SELECT addid, (gc).rating, (gc).addy, (gc).geomout
            FROM a LEFT JOIN LATERAL geocode(address, 1) AS gc ON (true)
            ) AS g
      WHERE g.addid = addr_to_geocode.addid;

      COMMIT;
  END LOOP;
END;
$$;
```

http://www.postgresonline.com/journal/index.php?/archives/390-Using-procedures-for-batch-geocoding-and-other-batch-processing.html

```
CREATE PROCEDURE new_customer(name text, address text)
LANGUAGE plpgsql
AS $$
BEGIN
    INSERT INTO audit_log (entry)
      VALUES ('someone tried to create a new customer');
    COMMIT;
    INSERT INTO customers VALUES (name, address);
END;
$$;

CALL new_customer('somename', 'someaddress');
```

```
CREATE PROCEDURE waste_xid(cnt int)
LANGUAGE plpgsql
AS $$
DECLARE
    i int;
BEGIN
    FOR i IN 1..cnt LOOP
        PERFORM txid_current();
        COMMIT;
    END LOOP;
END;
$$;
```

implementation details

SPI functions

- `SPI_connect_ext(SPI_OPT_NONATOMIC)`
- `SPI_start_transaction()`
- `SPI_commit()`
- `SPI_rollback()`

tips for PL authors

- write a giant test suite
- use `SPI_connect_ext()`
- implement commit/rollback calls
- use portal pinning (`PinPortal()`, `HoldPinnedPortals()`)
- use `PortalContext` instead of `TopTransactionContext`
- consider lifetime of objects carefully

dynamic result sets

(not implemented)

```
CREATE PROCEDURE pdrstest1()
LANGUAGE SQL
AS $$
DECLARE c1 CURSOR WITH RETURN FOR SELECT * FROM cp_test2;
DECLARE c2 CURSOR WITH RETURN FOR SELECT * FROM cp_test3;
$$;

CALL pdrstest1();
```

the end