ACE IT WITH ACID

Postgres transactions for fun and profit

Oleksii Kliukin

Postgres Conference 2019

New York, US

Oleksii Kliukin

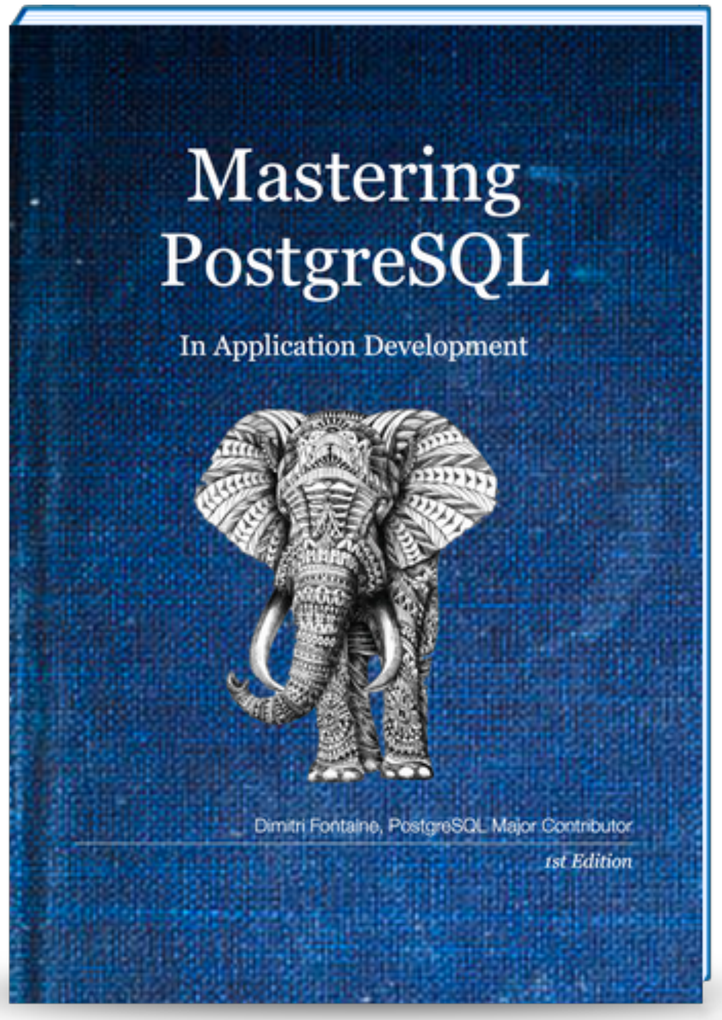
alex.kliukin@adjust.com

alexk@hintbits.com

@hintbits

*“We should talk about **transactional systems** **rather than storage** when we talk about RDBMS and other transaction technologies"*

https://masteringpostgresql.com



A

C

I

D

Atomicity

Consistency

Isolation

Durability

Every PostgreSQL statement is executed as a part of one or more (implicit) transactions

Explicit transaction blocks with **BEGIN…COMMIT**

https://www.postgresql.org/docs/current/static/dynamic-trace.html

```
postgresql$1:::transaction-start
{
	@start["Start"] = count();
	self->ts  = timestamp;
}

postgresql$1:::transaction-abort
{
	@abort["Abort"] = count();
}

postgresql$1:::transaction-commit
/self->ts/
{
	@commit["Commit"] = count();
	@time["Total time (ns)"] = sum(timestamp - self->ts);
	self->ts=0;
}
```

$1 - process pid

Usage: ./txn_count.d [pid]

Get pid with select pg_backend_pid();

```
$ ./txn_count.d 30580

Start                           1
Commit                          1
Total time (ns)            132552
```

```
select pg_backend_pid()
pg_backend_pid
----------------
          30580
(1 row)
```

```
$ ./txn_count.d 30580

Start                          1
Commit                         1
Total time (ns) 11640040116
```

```
begin;
BEGIN

insert into test values(1);
INSERT 0 1

insert into test values(2);
INSERT 0 1

commit;
COMMIT
```

```
$ ./txn_count.d 30580

Start                           1
Abort                           1
```

```
begin;
BEGIN

vacuum test;
ERROR:  VACUUM cannot run
inside a transaction block

rollback;
ROLLBACK
```

```
$ ./txn_count.d 30580

Start                           3
Commit                          3
Total time (ns)          20440868
```

```
vacuum test;
VACUUM
```

Find a Tile

```sql
create schema findatile;

set search_path to findatile, public;

create table player (
  p_id       bigserial primary key,
  p_name     text not null,
  p_email    text not null,
  p_password text not null
);

create table quest (
  q_id                 bigserial primary key,
  q_player_id          bigserial references player (p_id),
  q_tile_id            bigserial references tile (t_id),
  q_is_solved          boolean default false,
  q_proposed_location  point,
  q_proposed_photo_path text,
);

create table tile (
  t_id           serial primary key,
  t_name         text unique not null,
  t_description  text        not null,
  t_location     point       not null,
  t_photo_path   text        not null,
  t_author       bigint references player (p_id),
  t_is_published bool default false
);
```

Non-atomic changes

psql –qh localhost –U owner –d demo –f findatile.sql

```
psql:/findatile.sql:19: ERROR:  relation "tile" does not exist
```

psql –qh localhost –U owner –d demo –c "\dt findatile.*”

```
          List of relations
  Schema   |  Name  | Type  | Owner
-----------+--------+-------+-------
 findatile | tile   | table | owner
 findatile | player | table | owner
(2 rows)
```

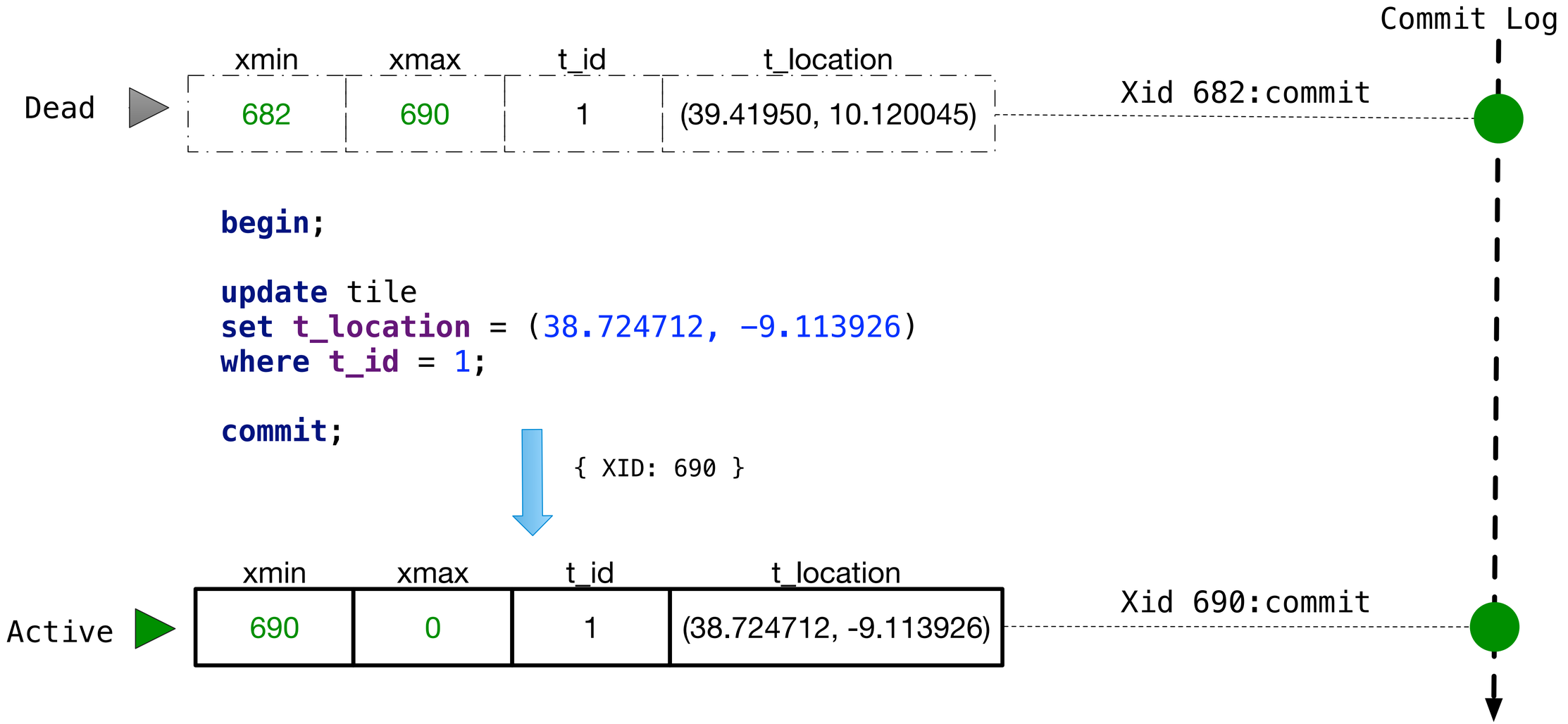
Atomic changes and rollbacks

```
psql -qh localhost -U owner -d demo -1f findatile.sql

psql:findatile.sql:19: ERROR:  relation "tile" does not exist

psql:findatile.sql:29: ERROR:  current transaction is aborted, commands
ignored until end of transaction block

psql -qh localhost -U owner -d demo  -c "\dt findatile.*”
       List of relations
 Schema | Name | Type | Owner
--------+------+------+-------
(0 rows)
```

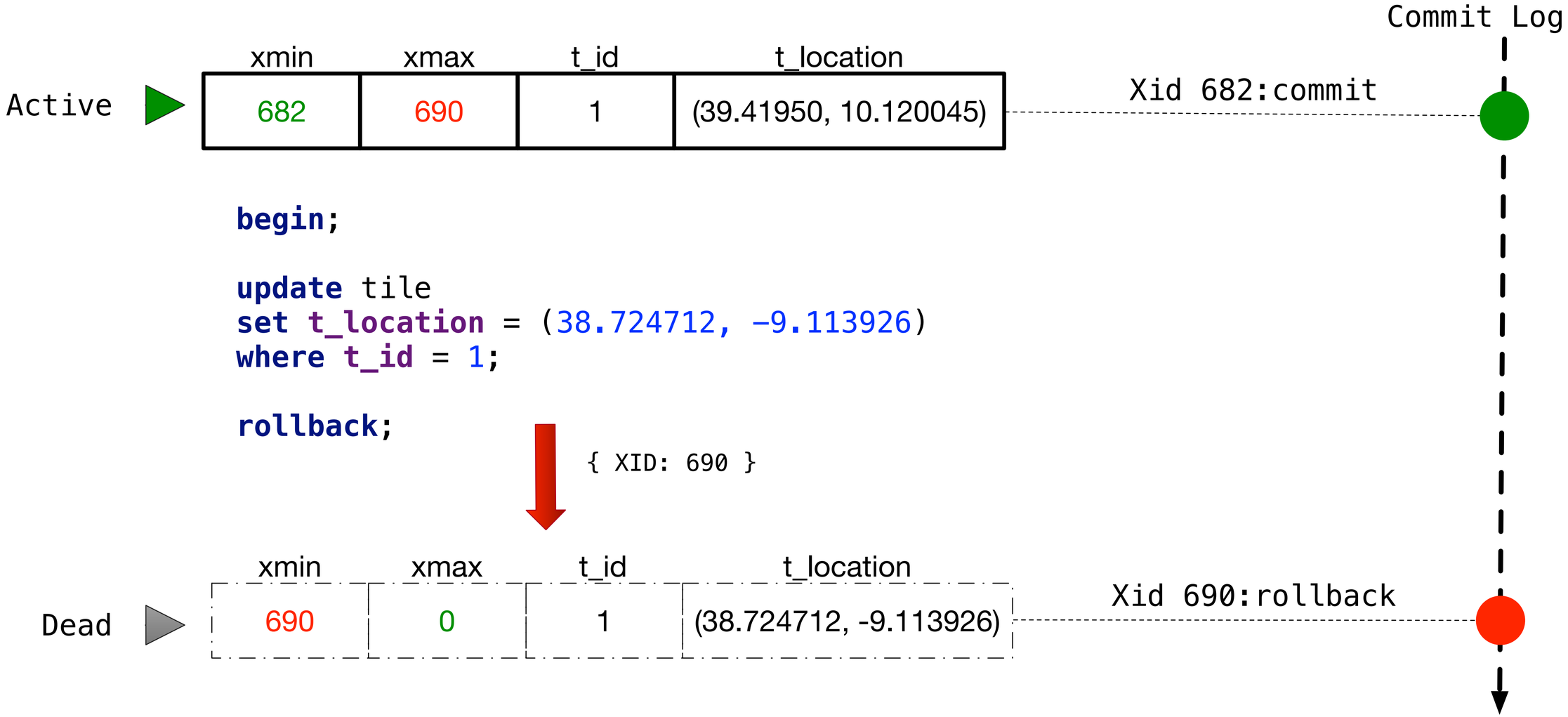
GDPR delete

```
begin;

delete
from quest using player
where q_player_id = p_id
  and p_email = 'anonymous@example.com';

update tile
set t_author_id = -1
from player
where p_id = t_author_id
  and p_email = 'anonymous@example.com';

delete
from player
where p_email = ‘anonymous@example.com';

commit;
```

MVCC and commit



MVCC and rollback



Transactional DDL: testing indexes

```
begin;

create extension pg_trgm;

create index tile_t_description_trgm_idx
on tile using gin(t_description gin_trgm_ops);

explain analyze select * from tile
where t_description like '%amsterdam%';

rollback;
```

Transactional DDL: tracking changes

```
begin;

alter table player add column is_visible(boolean);

create table scores(
  player_id integer references player(p_id),
  score integer not null
);

insert into versioning.changes(name, description, last_modified, modified_by)
values
('findmytiles-1', 'add user visibility and scores table', now(), current_user);

commit; -- or rollback if we made a mistake
```

Database dump: single transaction

```
pg_dump  -h localhost -U robot_backup -d demo -f dump.sql
```

```
select a.pid, a.query, a.xact_start, l.virtualtransaction
from pg_stat_activity a
       join pg_locks l using (pid)
where a.usename = 'robot_backup'
  and locktype = ‘virtualxid';
```

```
-[ RECORD 1 ]-----+-----------------------------------------------------------------------------
pid               | 6060
query             | COPY findatile.player (p_id, p_name, p_email, p_password) TO stdout;
xact_start        | 2018-10-20 23:52:43.81651+02
virtualtransaction | 3/3620
```

```
select a.pid, a.query, a.xact_start, l.virtualtransaction from pg_stat_activity …
```

```
-[ RECORD 1 ]---+-------------------------------------------------------------------------------
pid               | 6060
query             | COPY findatile.quest (q_player_id, q_tile_id, q_is_solved, q_proposed_location,
q_proposed_photo_path) TO stdout;
xact_start        | 2018-10-20 23:52:43.81651+02
virtualtransaction | 3/3620
```

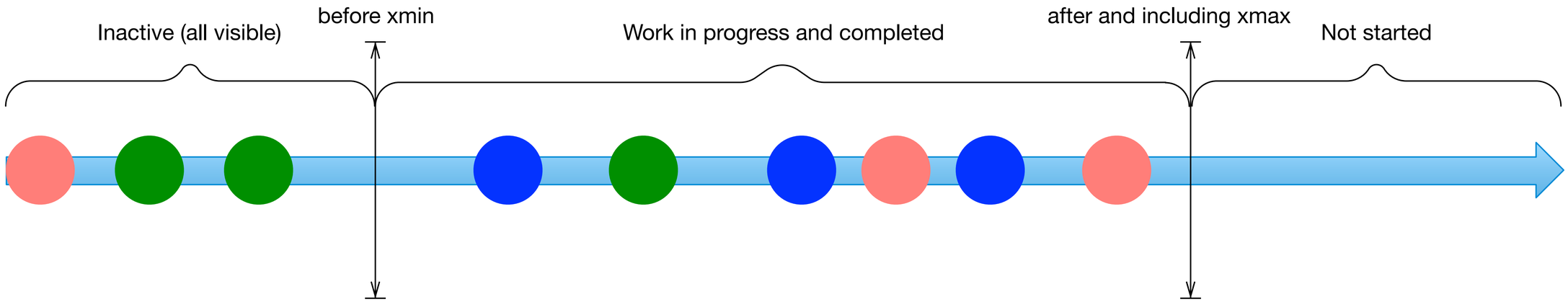
Transaction isolation levels

pg_dump creates a transaction with **REPEATABLE READ** isolation level

Isolation levels:

- READ COMMITTED (default)
- REPEATABLE READ
- SERIALIZABLE

Snapshots and visibility



committed

rolled back

in-progress

```
xmin:24295
xmax:24299
…
xcnt:3
xip:24295
xip:24296
xip:24297
```

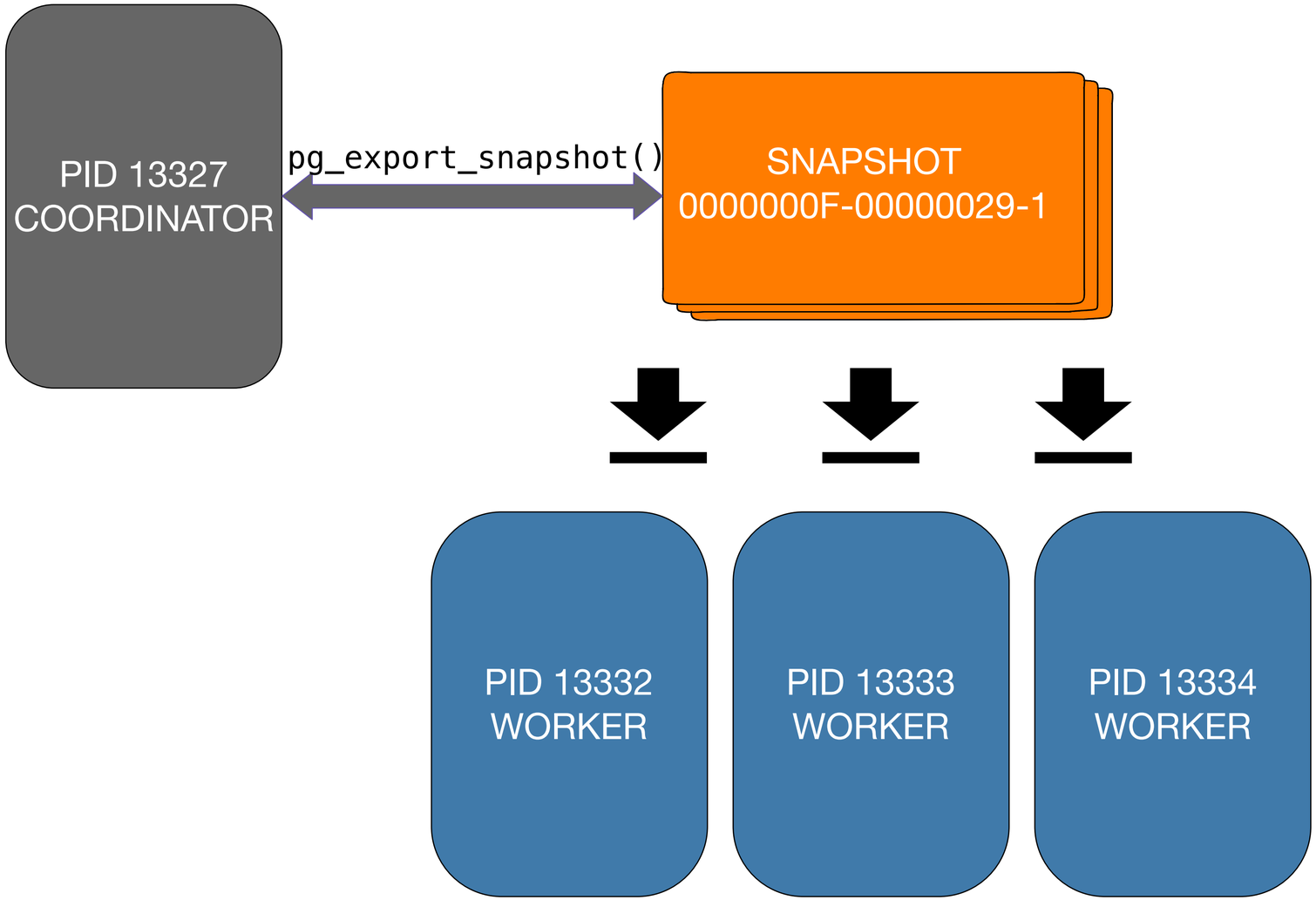
Database dump: multi-process

```
pg_dump -h localhost -d demo -U robot_backup -Fd -j3 -f /backup/
```

```
select pid, usename,
       case when state = 'active' then query else state end as action
from pg_stat_activity
where usename = ‘robot_backup';
```

```
-[ RECORD1 ]-------------------------------------------------------------------------------------------------------------------------------------------------
pid      | 13327
usename  | robot_backup
action   | idle in transaction
-[ RECORD2 ]-------------------------------------------------------------------------------------------------------------------------------------------------
pid      | 13332
usename  | robot_backup
action   | COPY findatile.tile (t_id, t_name, t_description, t_location, t_photo_path, t_author, t_is_published) TO stdout;
-[ RECORD 3 ]------------------------------------------------------------------------------------------------------------------------------------------------
pid      | 13333
usename  | robot_backup
action   | COPY findatile.player (p_id, p_name, p_email, p_password) TO stdout;
-[ RECORD 4 ]
-------------------------------------------------------------------------------------------------------------------------------------------------
pid      | 13334
usename  | robot_backup
action   | COPY findatile.quest (q_player_id, q_tile_id, q_is_solved, q_proposed_location, q_proposed_photo_path) TO stdout;
```

Synchronized visibility via snapshots



SET TRANSACTION SNAPSHOT ‘0000000F-00000029-1

```
$ cat pg_snapshots/
0000000F-00000029-1

vxid:15/41
pid:13327
dbid:24593
iso:2
ro:1
xmin:24295
xmax:24299
xcnt:3
xip:24295
xip:24296
xip:24297
sof:0
sxcnt:0
rec:0
```

Explain analyze data-modifying queries

```
begin;

explain analyze with
 players_to_delete as (
 delete from player
 where p_email = 'player42@example.com'
 returning p_id
),
 tiles_to_update as (
 update tile
 set t_author_id = -1
 from players_to_delete
 where p_id = t_author_id)

delete from quest
using players_to_delete
where q_player_id = p_id;

rollback;
```

exclusive	node
0.016	→ Delete on quest (cost=319,316.66..1,494,052.57 rows=10,002 width=38) (actual time=725,652.154..725,652.154 rows=0 loops=1)
	CTE players_to_delete
0.408	⭐ → Delete on player (cost=0.56..8.58 rows=1 width=6) (actual time=0.439..0.473 rows=1 loops=1)
0.065	→ Index Scan using player_p_email_key on player (cost=0.56..8.58 rows=1 width=6) (actual time=0.054..0.065 rows=1 loops=1) Index Cond: (p_email = 'player42@example.com'::text)
	CTE tiles_to_update
0.015	→ Update on tile (cost=0.03..319,308.05 rows=2 width=143) (actual time=134,597.969..134,597.969 rows=0 loops=1)
65,450.791	→ Hash Join (cost=0.03..319,308.05 rows=2 width=143) (actual time=134,597.954..134,597.954 rows=0 loops=1) Hash Cond: tile.t_author_id = players_to_delete_1.p_id)
69,147.117	→ Seq Scan on tile (cost=0.00..281,808.00 rows=10,000,000 width=111) (actual time=0.042..69,147.117 rows=10,000,000 loops=1)
0.021	→ Hash (cost=0.02..0.02 rows=1 width=40) (actual time=0.046..0.046 rows=1 loops=1) Buckets: 1024 Batches: 1 Memory Usage: 9kB
0.025	→ CTE Scan on players_to_delete players_to_delete_1 (cost=0.00..0.02 rows=1 width=40) (actual time=0.017..0.025 rows=1 loops=1)
352,541.428	→ Hash Join (cost=0.03..1,174,735.94 rows=10,002 width=38) (actual time=725,652.138..725,652.138 rows=0 loops=1) Hash Cond: (quest.q_player_id = players_to_delete.p_id)
373,110.152	→ Seq Scan on quest (cost=0.00..987,135.92 rows=49,999,992 width=14) (actual time=0.030..373,110.152 rows=49,999,991 loops=1)
0.038	→ Hash (cost=0.02..0.02 rows=1 width=40) (actual time=0.558..0.558 rows=1 loops=1) Buckets: 1024 Batches: 1 Memory Usage: 9kB
0.520	→ CTE Scan on players_to_delete (cost=0.00..0.02 rows=1 width=40) (actual time=0.473..0.520 rows=1 loops=1)

explain.depesz.com

```
Delete on quest (actual time=725652.154..725652.154 rows=0 loops=1)
  CTE players_to_delete
    ->  Delete on player (actual time=0.439..0.473 rows=1 loops=1)
          ->  Index Scan using player_p_email_key on player (actual time=0.054..0.065 rows=1 loops=1)
                Index Cond: (p_email = 'player42@example.com'::text)
  CTE tiles_to_update
    ->  Update on tile (actual time=134597.969..134597.969 rows=0 loops=1)
          ->  Hash Join (actual time=134597.954..134597.954 rows=0 loops=1)
                Hash Cond: (tile.t_author_id = players_to_delete_1.p_id)
                ->  Seq Scan on tile (actual time=0.042..69147.117 rows=10000000 loops=1)
                ->  Hash (actual time=0.046..0.046 rows=1 loops=1)
                      Buckets: 1024  Batches: 1  Memory Usage: 9kB
                      ->  CTE Scan on players_to_delete players_to_delete_1 (actual time=0.017..0.025 rows=1 loops=1)
  ->  Hash Join (actual time=725652.138..725652.138 rows=0 loops=1)
        Hash Cond: (quest.q_player_id = players_to_delete.p_id)
        ->  Seq Scan on quest (actual time=0.030..373110.152 rows=49999991 loops=1)
        ->  Hash (actual time=0.558..0.558 rows=1 loops=1)
              Buckets: 1024  Batches: 1  Memory Usage: 9kB
              ->  CTE Scan on players_to_delete (actual time=0.473..0.520 rows=1 loops=1)
Planning Time: 1.002 ms
Trigger for constraint tile_t_author_id_fkey on player: time=1775.411 calls=1
Trigger for constraint quest_q_player_id_fkey on player: time=17393.655 calls=1
Execution Time: 879419.872 ms
(23 rows)
```

```
create table quest (
  q_id                  bigserial primary key,
  q_player_id           bigserial references player (p_id),
  q_tile_id             bigserial references tile (t_id),
  q_is_solved           boolean default false,
  q_proposed_location   point,
  q_proposed_photo_path text
);

create table tile (
  t_id           serial primary key,
  t_name         text unique not null,
  t_description  text        not null,
  t_location     point       not null,
  t_photo_path   text        not null,
  t_author_id    bigint references player (p_id),
  t_is_published bool default false
);

-- new indexes to speed up the foreign key check
-- note: create index concurrently cannot be part of a transaction block
create index concurrently quest_q_tile_id_idx on quest(q_tile_id);
create index concurrently quest_q_player_id_idx on quest(q_player_id);

create index concurrently tile_t_author_id_idx on tile(t_author_id);
```

```
Delete on quest (actual time=0.870..0.870 rows=0 loops=1)
  CTE players_to_delete
    ->  Delete on player (actual time=0.565..0.709 rows=1 loops=1)
          ->  Index Scan using player_p_email_key on player (actual time=0.085..0.097 rows=1 loops=1)
                Index Cond: (p_email = 'player42@example.com'::text)
  CTE tiles_to_update
    ->  Update on tile (actual time=0.180..0.180 rows=0 loops=1)
          ->  Nested Loop (actual time=0.161..0.161 rows=0 loops=1)
                ->  CTE Scan on players_to_delete players_to_delete_1 (actual time=0.014..0.025 rows=1 loops=1)
                ->  Index Scan using tile_t_author_id_idx on tile (actual time=0.095..0.095 rows=0 loops=1)
                      Index Cond: (t_author_id = players_to_delete_1.p_id)
  ->  Nested Loop (actual time=0.850..0.850 rows=0 loops=1)
        ->  CTE Scan on players_to_delete (actual time=0.598..0.752 rows=1 loops=1)
        ->  Bitmap Heap Scan on quest (actual time=0.056..0.056 rows=0 loops=1)
              Recheck Cond: (q_player_id = players_to_delete.p_id)
              ->  Bitmap Index Scan on quest_q_player_id_idx (actual time=0.032..0.032 rows=0 loops=1)
                    Index Cond: (q_player_id = players_to_delete.p_id)
Planning Time: 0.859 ms
Trigger for constraint tile_t_author_id_fkey on player: time=0.531 calls=1
Trigger for constraint quest_q_player_id_fkey on player: time=0.362 calls=1
Execution Time: 2.241 ms
(21 rows)
```

Setting ON_ERROR_ROLLBACK = off

```
psql -h localhost -U owner -d demo

# begin

BEGIN

# insert into player(p_name, p_email, p_password) values('tom', 'tom@example.com', '');

INSERT 0 1

# insert into player(p_name, p_email, p_password) values('dave' 'dave@example.com', '');

ERROR:  syntax error at or near "'@example.com'"

LINE 1: ...ayer(p_name, p_email, p_password) values('dave' 'dave@ex...
^

# insert into player(p_name, p_email, p_password) values('dave' ‘dave@example.com', '');

ERROR:  current transaction is aborted, commands ignored until end of transaction block

# rollback;

ROLLBACK
```

Setting ON_ERROR_ROLLBACK = interactive

```
psql -h localhost -U owner -d demo -v ON_ERROR_ROLLBACK=interactive


# begin
BEGIN
# insert into player(p_name, p_email, p_password) values('tom', 'tom@example.com', '');
INSERT 0 1
# insert into player(p_name, p_email, p_password) values('dave' 'dave@example.com', '');
ERROR:  syntax error at or near "'dave@example.com'"
LINE 1: ...ayer(p_name, p_email, p_password) values('dave' 'dave@ex...
^
# insert into player(p_name, p_email, p_password) values('dave' ‘dave@example.com', ‘');
INSERT 0 1
# commit;
COMMIT
```

Catching errors with subtransactions

```
psql -h localhost -U owner -d demo
# begin;
BEGIN
# savepoint one;
SAVEPOINT
# insert into player(p_name, p_email, p_password) values('tom', 'tom@example.com', '');
INSERT 0 1
# release one;
RELEASE
# savepoint two;
SAVEPOINT
# insert into player(p_name, p_email, p_password) values('dave' 'dave@example.com', '');
ERROR:  syntax error at or near "'dave@example.com'"
LINE 1: ...ayer(p_name, p_email, p_password) values('dave' 'dave@ex...
^
# rollback to two;
ROLLBACK
# insert into player(p_name, p_email, p_password) values(‘dave’, ‘dave@example.com', '');
INSERT 0 1
# commit;
COMMIT
```

Batch updates

```
-- session 1
update tile set t_photo_path = format('/photos/%s', t_photo_path);

-- session 2, a bit later
update tile set description = format('new and shiny! %s', description) where t_id = 1000;

select pid, query, wait_event_type
from pg_stat_activity
where state = 'active'
  and pid != pg_backend_pid()
  and backend_type = 'client backend';

-[ RECORD 1 ]---+-----------------------------------------------------------------------------------------
pid             | 25611
query           | update tile set t_description = format('new! %s', t_description) where t_id = 1000
wait_event_type | Lock
-[ RECORD 2 ]---+-----------------------------------------------------------------------------------------
pid             | 25610
query           | update tile set t_photo_path = format('/photos/%s', t_photo_path);
wait_event_type |
```

Batch updates (via a function)

```
CREATE OR REPLACE FUNCTION batch_change_path(p_new_prefix TEXT, p_batch_size INTEGER)
  RETURNS VOID AS
$$
BEGIN
  WHILE EXISTS(SELECT 1
               FROM tile
               WHERE t_photo_path NOT LIKE p_new_prefix || '%') LOOP
    WITH keys_to_update AS (
        SELECT t_id
        FROM tile
        WHERE t_photo_path NOT LIKE p_new_prefix || '%'
        LIMIT p_batch_size
    ) UPDATE tile b
    SET t_photo_path = p_new_prefix || t_photo_path FROM keys_to_update ktu
    WHERE a.t_id = ktu.t_id;
  END LOOP;
END;
$$
LANGUAGE plpgsql;
```

Batch updates (via a function)

```
WHILE EXISTS(SELECT 1
             FROM tile
             WHERE t_photo_path NOT LIKE p_new_prefix || '%') LOOP
   WITH keys_to_update AS (
       SELECT t_id
       FROM tile
       WHERE t_photo_path NOT LIKE p_new_prefix || '%'
       LIMIT p_batch_size
   ) UPDATE tile b
   SET t_photo_path = p_new_prefix || t_photo_path FROM keys_to_update ktu
   WHERE t.a_id = ktu.a_id;
 END LOOP;
```

A function is executed as one transaction

```
-- session 1
select batch_change_path('/photos', 100)

-- session 2, a bit later
update tile set description = format('new and shiny! %s', description) where t_id = 1000;

select pid, query, wait_event_type
from pg_stat_activity
where state = 'active'
  and pid != pg_backend_pid()
  and backend_type = 'client backend';

-[ RECORD 1 ]---+-----------------------------------------------------------
pid             | 25961
query           | select batch_change_path('/photos', 100)
wait_event_type | Lock
-[ RECORD 2 ]---+-----------------------------------------------------------
pid             | 25610
query           | update tile set t_photo_path = format('/photos/%s', t_photo_path)
wait_event_type |
```

Batch updates (Postgres 11 procedures)

```
CREATE PROCEDURE batch_change_path(p_new_prefix TEXT, p_batch_size INTEGER)
  AS
$$
BEGIN
  WHILE EXISTS(SELECT 1
               FROM tile
               WHERE t_photo_path NOT LIKE p_new_prefix || '%') LOOP
    WITH keys_to_update AS (
        SELECT t_id
        FROM artifact
        WHERE t_photo_path NOT LIKE p_new_prefix || '%'
        LIMIT p_batch_size
    ) UPDATE artifact b
    SET t_photo_path = p_new_prefix || t_photo_path FROM keys_to_update ktu
    WHERE b.t_id = ktu.a_id;
    COMMIT;
  END LOOP;
END;
$$
LANGUAGE plpgsql;
```

Batch updates (Postgres 11 procedures)

```
WHILE EXISTS(SELECT 1
             FROM tile
             WHERE t_photo_path NOT LIKE p_new_prefix || '%') LOOP
   WITH keys_to_update AS (
       SELECT t_id
       FROM tile
       WHERE t_photo_path NOT LIKE p_new_prefix || '%'
       LIMIT p_batch_size
   ) UPDATE tile b
   SET t_photo_path = p_new_prefix || t_photo_path FROM keys_to_update ktu
   WHERE t.a_id = ktu.a_id;
   COMMIT;
 END LOOP;
```

A procedure can produce multiple transactions

```
select pid, query, backend_xid
from pg_stat_activity
where state = 'active'
  and pid != pg_backend_pid()
  and backend_type = 'client backend';

-[ RECORD 1 ]---+-----------------------------------------------
pid             | 26015
query           | call batch_change_path('/photos', 100)
backend_xid     | 783
-[ RECORD 2 ]---+-----------------------------------------------
pid             | 26015
query           | call batch_change_path('/photos', 100)
backend_xid     | 799
```

Transactions with foreign data wrappers

- Query data external to the database of origin
- PostgreSQL and other data sources (MySQL, SQLLite, MongoDB, Cassandra, Kafka, text files, Oracle, SQL Server, …)
- postgres_fdw supports reading and writing other postgresql clusters (federation)
- Some other FDWs (i.e. SQLite) supports writing as well
- How do they implement transactional semantics?

Transactions with foreign data wrappers

```
CREATE EXTENSION cstore_fdw;
CREATE SERVER cstore_server FOREIGN DATA WRAPPER cstore_fdw;


CREATE FOREIGN TABLE tile_reviews (
  ta_id          integer,
  tr_tile_id     integer,
  tr_reviewer_id integer,
  tr_date        date,
  tr_content     text,
  tr_rating      integer
)
SERVER cstore_server OPTIONS(compression 'pglz');

begin;
BEGIN
insert into tile_reviews select 1, 1, 1, now(), 'I found this title via a postgres talk', 10;
INSERT 0 1
rollback;
ROLLBACK

select * from tile_reviews ;
 tr_id |   tr_tile_id   | tr_reviewer_id |  tr_date   |                tr_content                | tr_rating
-------+----------------+----------------+------------+------------------------------------------+-----------
     1 |              1 |              1 | 2018-10-23 | I found this title via the postgres talk |        10
(1 row)
```

Federation with postgres foreign data wrapper

```
create extension postgres_fdw;

create server one foreign data wrapper postgres_fdw options (host
'localhost', port ‘5433');
create user mapping for owner server one options (user ‘owner');

create server two foreign data wrapper postgres_fdw options (host
'localhost', port '5434');
create user mapping for owner server two options (user 'owner');

create schema tiles_one;
create schema tiles_two;

import foreign schema findatile from server one into tiles_one;
import foreign schema findatile from server two into tiles_two;
```

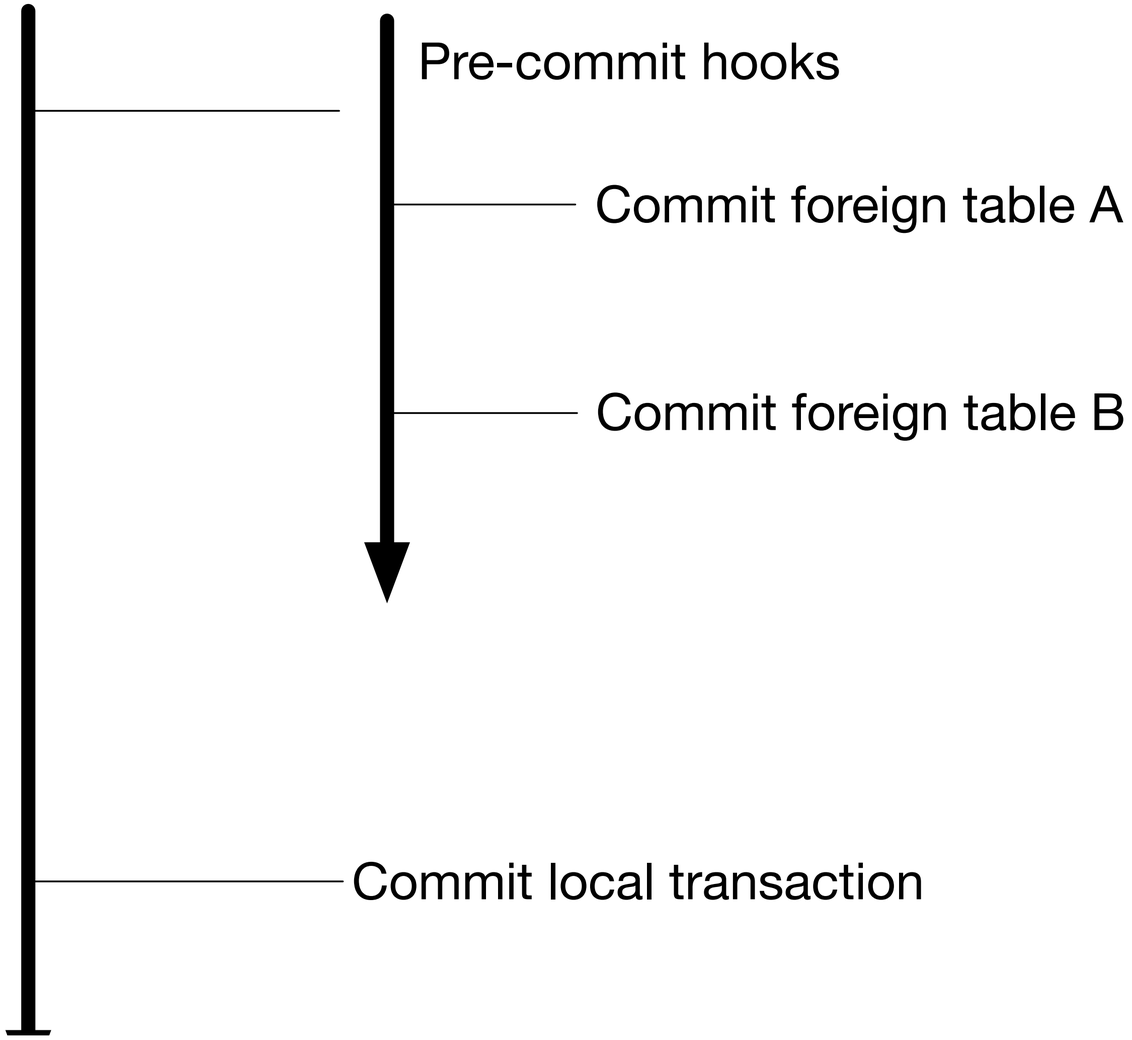
Rollback with postgres_fdw

```
select (select count(1) from findatile.artifact   where t_is_published = false)  as not_published_local,
       (select count(1) from tiles_one.artifact where t_is_published = false) as not_published_one,
       (select count(1) from tiles_two.artifact where t_is_published = false) as not_published_two;

 not_published_local | not_published_one   | not_published_two
---------------------+---------------------+--------------------
                   1 |                   8 |                  5
(1 row)


begin;

delete from findatile.tile where   t_is_published = false;
delete from tiles_one.tile where t_is_published = false;
delete from tiles_two.tile where t_is_published = false;

rollback;

select (select count(1) from findatile.tile   where t_is_published = false)   as not_published_local,
       (select count(1) from tiles_one.tile where t_is_published = false) as not_published_one,
       (select count(1) from tiles_two.tile where t_is_published = false) as not_published_two;

 not_published_local | not_published_one   | not_published_two
---------------------+---------------------+--------------------
                   1 |                   8 |                  5
(1 row)
```

Do we have real distributed transactions?

```
begin;

with deletes_local as (delete from findatile.tile where t_is_published = false) select count(1) from deletes_local;
   1
with deletes_one as (delete from tiles_one.artifact where t_is_published = false) select count(1) from deletes_one;
   8
with deletes_two as (delete from tiles_two.artifact where t_is_published = false) select count(1) from deletes_two;
   5

-- shutdown server one


commit;
ERROR:  server closed the connection unexpectedly
  This probably means the server terminated abnormally
  before or while processing the request.

-- start the server one back


select (select count(1) from findatile.tile where t_is_published = false)       as not_published_local,
       (select count(1) from tiles_one.artifact where t_is_published = false) as not_published_one,
       (select count(1) from tiles_two.artifact where t_is_published = false) as not_published_two;

 not_published_local | not_published_one   | not_published_two
---------------------+---------------------+-------------------
                   1 |                   0 |                 5
(1 row)
```

Distributed commit with postgres_fdw



Prepared transaction and 2PC

On every shard (prepare transaction)

```
begin;
delete from findatile.tile where t_is_published = false;
prepare transaction 'delete_unpublished';
```

Later, on every shard (commit transaction)

```
commit prepared ‘delete_unpublished';
```

If some shards are down at the commit stage, find all unfinished prepared transactions when they are up and issue commit/rollback on them.

Currently not supported by Postgres FDW (see https://wiki.postgresql.org/wiki/2PC_on_FDW)

“we believe it is better to have application programmers deal with performance problems due to overuse of transactions as bottlenecks arise, rather than always coding around the lack of transactions”

–Spanner: Google’s Globally-Distributed Database paper

Thank you!

Contact me via:
Email: alexk@hintbits.com
Twitter: @hintbits