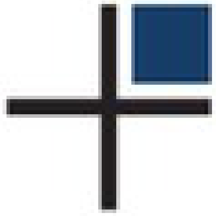
2ndQuadrant
Professional PostgreSQL

Logical Decoding for Auditing

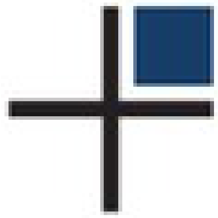
Gianni Ciolli

PostgreSQL Conference Europe 2014
Madrid, 21-24 October



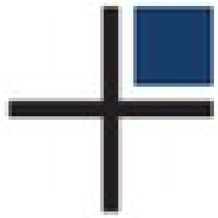
Feature History 1/2

- 7.0 and older
  - Changes written to 1+ files at once
  - Random writes
  - Changes are not collected
- 7.1 (2001): **W**rite **A**head **L**og
  - All changes “serialized” into *one* sequence
  - Sequential writes to *WAL files*
  - Changes are collected in binary format
- 8.0 (2005): **P**oint **I**n **T**ime **R**ecovery
  - WAL files copied to the *archive*
  - Replay changes on another database server
  - The whole database server is cloned



Feature History 2/2

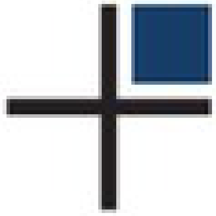
- 8.2 (2006): Warm Standby
  - While replaying changes, waits for next WAL file
  - The clone is continuously updated. . .
  - . . . a.k.a. Replication
- 9.0 (2010): Hot Standby, Streaming Replication
  - While replaying changes, read-only access
  - Changes are streamed using a client connection
- 9.4 (2014): Logical Decoding
  - Changes are streamed in *logical* format



Binary changes: WAL

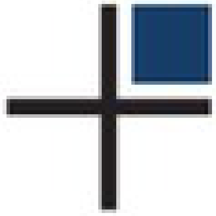
- Example

  «write bytes X,Y,. . . ,Z into file A at offset B»
- Very fast
  - Only which bytes have changed, and how
  - No SQL, very little logical information
- Each change depends on the previous one
  - Cannot "understand" the change (*opaque*)
  - Cannot modify a change safely
  - Cannot merge changes from different systems
- Binary Replication is *unidirectional*



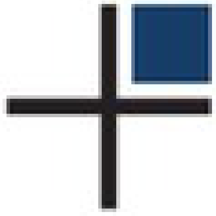
Logical changes

- Example:
  «Insert string ‘Hello’ into table T»
- Logical changes can be *understood*
  - Resolve Conflicts
    - Bi-Directional Replication
  - Filter Changes
    - Operate data workflows
    - Share a subset of the database
  - Monitor Changes
    - Audit selected portions



How Logical Works

- Decoding
  - WAL includes binary changes to *files*
  - WAL is *decoded* to changes to *tables*
- Tables $\leftrightarrow$ Files
  - Mapping required for decoding
  - Defined in the *catalog*
  - The catalog was not transactional…
- Output
  - Changes are streamed by `walsender`
  - Logical decoding **transforms** data before sending
- Lots of work to make it work…



Binary changes

App

Backend

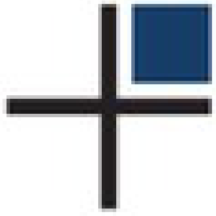
walsender

(remote)

pg_xlog

Binary

Logical



Logical decoding

Binary

Logical

App

walsender

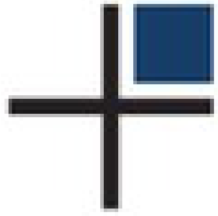
Backend

output plugin

(remote)

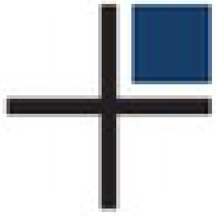
pg_xlog

catalog



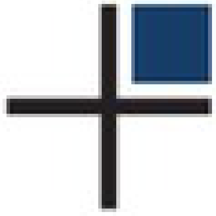
Logical v Binary

- Logical provides flexibility
- Binary provides performance
- Good to have *both*!
- Different use cases
- Mature use cases
  - Experience comes from existing “external” solutions:
  - Slony, Londiste, Bucardo, pgpool-II
- **Michael Paquier** talked about Logical Decoding and its applications on Thursday morning



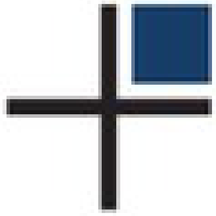
Logical Decoding

- Other Applications
  - Selective replication
  - Bi-Directional replication
  - Diagnostics
  - Data workflows



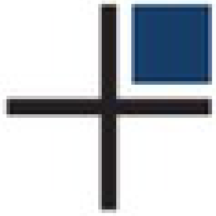
Quickstart

- Ensure PostgreSQL is configured
- Choose an *output plugin*
- Create a *logical replication slot*
- Logical decoding is running. . .
- Drop the replication slot when finished



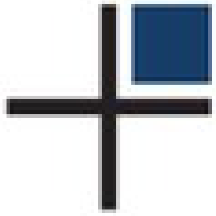
Output Plugins

- `test_decoding`
  - Part of *contrib*
  - Provided as an example
  - Decodes changes to text
- Write your own plugin
  - Clearly documented procedure
  - An output plugin encodes:
    - Filter rules
    - Output format



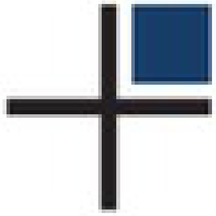
Other Logical Benefits

- Binary format is “forwards”
  - No “old” available, only “new”
  - Cannot go backwards
- Logical format is also “backwards”
  - “old” and “new” are both available
    - `REPLICA IDENTITY`
  - Could implement “time travel”. . .



WARNING

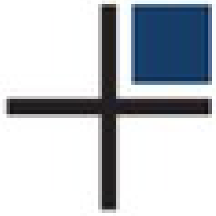
- PostgreSQL **had** time travel!
- Removed **16 years** ago
  - Performance reasons
- Before coding:
  - Evaluate costs v benefits
  - Check history…



REPLICA IDENTITY

```
ALTER TABLE myTable
  REPLICA IDENTITY ...;
```

- Controls what columns are replicated for old rows
  - NOTHING
    - No column
  - FULL
    - All columns
  - USING INDEX myIndex
    - Columns covered by the index
  - DEFAULT
    - Primary key columns, if any

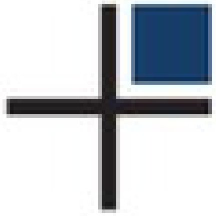


Minimal Example: 1/2 (config)

```
ALTER SYSTEM
  SET wal_level = logical;

ALTER SYSTEM
  SET max_replication_slots = 10;

-- Then restart...
```

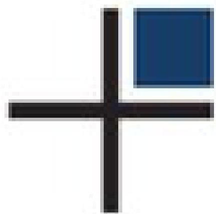


Minimal Example: 2/2 (SQL)

- Step 1: create a logical replication slot

```
SELECT * FROM
  pg_create_logical_replication_slot
    ('slot1','test_decoding');


 slot_name | xlog_position
-----------+-----------------
 slot1     | 0/1AE67D4
(1 row)
```

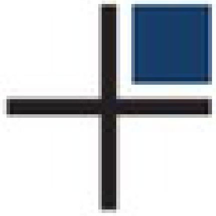


Minimal Example: 2/2 (SQL)

- Step 2: peek changes (same db)

```
SELECT * FROM
  pg_logical_slot_peek_changes
    ('slot1', NULL, NULL);

 location  | xid  |            data
-----------+------+------------------------
 0/1B137EC | 9980 | BEGIN 9980
 0/1B18FCC | 9980 | COMMIT 9980
 0/1B18FCC | 9981 | BEGIN 9981
 0/1B18FCC | 9981 | table public.don_juan:
           |      |   INSERT:
           |      |   country[text]:'Spain'
           |      |   count[integer]:1003
 0/1B1904C | 9981 | COMMIT 9981
(5 rows)
```



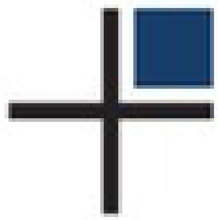
Minimal Example: 2/2 (SQL)

- What was the SQL?

```
CREATE TABLE don_juan (
country text NOT NULL,
count   int  NOT NULL );

INSERT INTO don_juan
  VALUES ('Spain', 1003);
```

- DDL is not replicated :-(



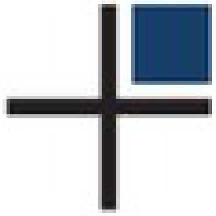
Minimal Example: 2/2 (SQL)

- Step 3: when finished, drop the slot

```
SELECT pg_drop_replication_slot('slot1');

 pg_drop_replication_slot
--------------------------

(1 row)
```



Minimal Example: 2/2 (cmd line)

- Step 1: create a logical replication slot

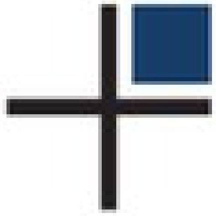
```
$ pg_recvlogical --dbname=postgres \
    --slot=slot1 --plugin=test_decoding \
    --create
```

- Step 2: peek changes (same db)

```
$ pg_recvlogical --dbname=postgres \
    --slot=slot1 \
    --start --file=-
```

- Step 3: when finished, drop the slot

```
$ pg_recvlogical --slot=slot1 --drop
```

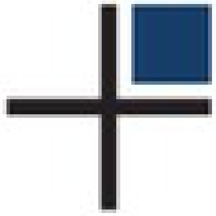


SQL v command line

- Not really the same
- Access via command line uses streaming replication
- Must set up a connection:

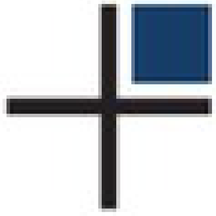
```
ALTER SYSTEM
  SET max_wal_senders = 10;
```

- Then restart. . .



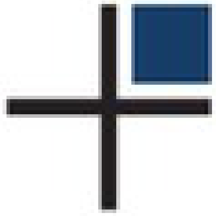
Auditing Mode 1

- Via SQL interface
  - Log to log tables
    - Do not log the logs!
    - From the same DB
    - But: superuser can retrospectively alter logs
    - Is it really a downside???
  - No separate service
    - Always up
    - Cheaper to manage
    - Very easy to query and monitor in **real time**



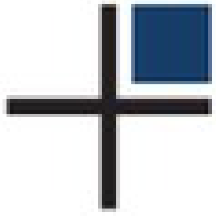
Auditing Mode 2

- Via external plugin
  - Log outside
    - Logs cannot be retrospectively altered
    - “eventually superuser-safe”
  - Separate service
    - Could be down
    - Must be managed
  - Custom plugin, to avoid parsing text
    - Can be done in YourSetup v2.0



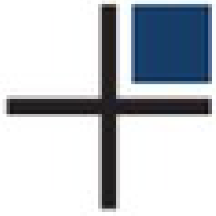
Auditing Pros and Cons

- Pros and Cons
  - Can only audit DML, not DDL
    - BDR uses Event Triggers to bridge the gap…
  - Cannot audit SELECT
    - This is difficult to audit anyway…
  - Single-database audit
    - Not a limitation actually!
  - Performance
    - Very efficient
    - Related benchmarks from Petr Jelinek and Andres Freund



And now. . .

Questions?

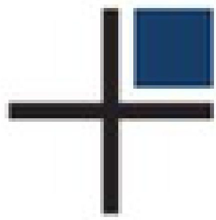


And then. . .

Thank you!

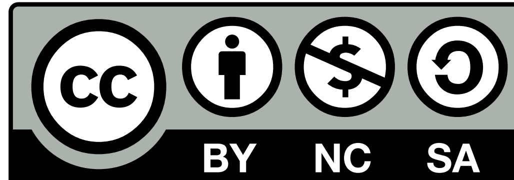
`http://2014.pgconf.eu/feedback`

(you need a PostgreSQL community account. . .)



Licence

This document is distributed under the **Creative Commons Attribution-Non commercial-ShareAlike 3.0 Unported** licence



A copy of the licence is available at the URL
`http://creativecommons.org/licenses/by-nc-sa/3.0/`
or you can write to
Creative Commons, 171 Second Street, Suite 300, San Francisco, California, 94105, USA.