A TARDIS for your ORM

PGDay'15 Russia
St Petersburg, Russia

Magnus Hagander
magnus@hagander.net
Magnus Hagander

- PostgreSQL
  - Core Team member
  - Committer
- PostgreSQL Europe
- Redpill Linpro
  - Infrastructure services
  - Principal database consultant
A TARDIS for your ORM
A TARDIS for your ORM

(Photo by zir.com)
A TARDIS for your ORM

• Application level time-travel
A step back

• Requirements
Requirements

- Existing data model
  - Minimize changes
- Detailed and statistical data
  - Highly sensitive personal information
Requirements

• Data is loaded in batches
  • (most of it)
• Manual corrections incrementally
  • No high concurrency
Requirements

• Large aggregate reports
• Smaller detailed reports
  • Including personal information
Requirements

• All pretty standard?
The challenges

• Reproduce incorrect reports
• If data was corrected between runs
The challenges

- Identify which reports contained a person
- Far in the past
- (luckily, not performance sensitive)
The challenges

- Maintain application flexibility
- Including manual query interface
  - Simple UI to build queries
  - Not direct SQL, but close
The challenges

• Preferably zero changes to application
• At least minimize them
The toolbox

• JBoss/Hibernate
  • Existing application
• PostgreSQL
  • phew...
Schema

- Fairly simple schema
  - ORM generated after all
- Many tables
- No "unusual" constructs
Schema restrictions

- All tables in **public** schema
- All tables have **id** column
  - Courtesy of Hibernate
- Very few schema changes
Step 1

- Keep the old data
- And keep track of when it's for
History tables

• Everybody knows a history table!
  • (right?)
• And everybody knows range types?
  • Each row gets a validity period
CREATE TABLE history.table1 (LIKE public.table1, _validrange tstzrange)
• Everybody used it?

```csharp
_validrange
[["2014-02-17 14:49:52.482618+01","2014-02-17 14:50:06.722589+01"),
["2014-02-17 14:50:06.722589+01",infinity])
```
ALTER TABLE history.table1
ADD CONSTRAINT table1_exclusion
EXCLUDE USING gist
  (id WITH =, _validrange WITH &&)
Update trigger

CREATE TRIGGER table1_history
  BEFORE INSERT OR UPDATE OR DELETE
  ON public.table1
  FOR EACH ROW
  EXECUTE PROCEDURE history.logtable_trigger()
Update trigger

• public contains current data
• history contains all historic data
• So we need to track all operations
IF TG_OP = 'INSERT' THEN
  EXECUTE 'INSERT INTO history.' || TG_RELNAME ||
    ' SELECT $1.*, tstzrange(
      NOW(),
      $$infinity$$,
      $$[]$$
    )' USING NEW;

RETURN NEW;
ELSIF TG_OP = 'UPDATE' THEN
    OPEN c FOR EXECUTE 'SELECT _validrange FROM history.' ||
    TG_RELNAME || ' WHERE id=$1 ORDER BY _validrange DESC
    LIMIT 1 FOR UPDATE' USING NEW.id;
    FETCH FROM c INTO tt;

    IF isempty(tstzrange(lower(tt), now(), $$[]$$)) THEN
        IF NOT lastxid = txid_current() THEN
            RAISE EXCEPTION 'UPDATEd would have empty validity: %d!', OLD;
        END IF;
    -- Row already updated! Delete the update for reinsert
    EXECUTE 'DELETE FROM history.' || TG_RELNAME ||
    ' WHERE CURRENT OF ' || quote_ident(c::text);
ELSE

EXECUTE 'UPDATE history.' || TG_RELNAME || ' SET _validrange=
tstzrange($1, now(), $$[)$$)
WHERE CURRENT OF ' || quote_ident(c::text) USING lower(tt);
END IF

EXECUTE 'INSERT INTO history.' || TG_RELNAME || ' SELECT $1.*,
tstzrange(NOW(), $$infinity$$, $$[)$$) ' USING NEW;

RETURN NEW;
ELSIF TG_OP = 'DELETE' THEN
OPEN c FOR EXECUTE 'SELECT _validrange FROM history.' || TG_RELNAME || ' WHERE id=$1 ORDER BY _validrange DESC LIMIT 1 FOR UPDATE' USING NEW.id;

FETCH FROM c INTO tt;

IF isempty(tstzrange(lower(tt), now(), $$[]$$)) THEN
    -- Row already updated, but now deleted
    EXECUTE 'DELETE FROM history.' || TG_RELNAME || ' WHERE CURRENT OF ' || quote_ident(c::text);
    RETURN OLD;
END IF;
DELETE trigger (contd)

EXECUTE 'UPDATE history.' || TG_RELNAME || ' SET _validrange=tstzrange($1, now(), $$[])$$)
WHERE CURRENT OF ' || quote_ident(c::text) USING lower(tt);
RETURN OLD;
END IF;
Accessing the history data

• Accessing history rows is easy
• Just specify validity time

```
SELECT id, a, b, c FROM history.table1
WHERE id = 42
AND _validrange @> '2015-03-07 14:32'::timestamptz
```

• Will use gist index
Almost there?

- Not very "minimum modifications"
- Especially when considering joins
  - Works fine
  - But _validrange check has to be on all tables!
Another shadow schema

```
CREATE SCHEMA timetravel;
```
CREATE VIEW timetravel.table1 AS
    SELECT id, a, b, c
    FROM history.table1
    WHERE _validrange @>
        current_setting('history.timestamp'::text)::timestamptz
Time-travel setting

• One setting controls "current time"
• Schema search order decides views
Time-travel

test=# SET search_path='timetravel';

SET

test=# SET history.history.timestamp='2015-03-07 14:32';

SET

test=# SELECT * FROM table1;

<table>
<thead>
<tr>
<th>id</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>42</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>
Time-travel

test=# SELECT * FROM table1;

<table>
<thead>
<tr>
<th>id</th>
<th>a</th>
<th>b</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>42</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

test=# SET history.timestamp='2015-03-07 14:29'

SET

test=# SELECT * FROM table1;

<table>
<thead>
<tr>
<th>id</th>
<th>a</th>
<th>b</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>42</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Application injection

- Time-travel is now automatic
- Once variables are injected
  - search_path
  - history.timestamp
Application injection

• Depends on framework
• Driver level
• Query wrapper
• Just a function call?
package redacted.postgresql.driver;

public class Driver extends org.postgresql.Driver {
  public Connection connect(String url, Properties info) throws SQLException {
    Connection con = super.connect(url, info);
    if (con != null) {
      InjectTimetravel();
    }
    return con;
  }
}
Considerations

• Don't forget to reset
  • Connection pooling!
• Query **public** schema for current data
  • Better performance!
The last requirement

• "Identify which reports contained a person"
The last requirement

• Full reporting query-logging
• Re-run reports to identify
  • With time-travel
  • Heuristics for known reports
• Yes, it's slow...
A word of warning

• ORM level cache
• Query or entity
• Needs to be aware
Conclusions

• Rangetypes are awesome :)  
• ORMs can be tricked  
  • And their simpleness can help  
• Use the flexibility of PostgreSQL!
Thank you!

Magnus Hagander
magnus@hagander.net
@magnushagander
http://www.hagander.net/talks/

This material is licensed CC BY-NC 4.0.