Reconciling Databases
(using schemas, Slony, DBI-Link, pgTAP and other tools)

Norman Yamada
The Millburn Corporation
nyamada@millburncorp.com
Basic Problem

- Data-driven application
- Inputs (e.g., prices, market hours, inventory levels) processed by algorithmic models using GNU R to produce output (e.g. signals)
- Both inputs and models can change at any time
  - Bad data
  - Corrections/revisions to models
- How do we test/compare our changes?
Traditional solution for code development

- Development/Staging/Production servers
- Code developed only on dev server
- When developer ready, change pushed to staging server.
- Staging server = Production server + 1 change
- Regression tests on all servers/automated builds from source control/scripted deployment
But data is not static

• Price data continually changes--corrections even to historical data
  – Different sources for prices may have different data

• For new markets, often real-time data best way to test

• Metadata for algorithmic models may change: sampling rate, number of inputs, etc…
Our use case

- Input tables
  - Instrument – Corn
  - Contract – Corn January 2010
  - Price – Corn Jan 2010 on 5/22/2009
  - Models – Buy commodity if price yesterday > price 3 months ago

- Parameter tables
  - Roll schedule – when do we switch from one contract to the next?
  - Market – Corn (always using nearest contract)
  - Market models – apply models A, B & C to Corn

- Output tables
  - Price percent – continuous stream of price changes for given market
  - Signals (buy / sell) – results of applying models to market on given date
  - Simulated profits – what should have happened...
How do we test and stage our data-driven application?
Development Environment

Model Processors

Production

Stage

Dev

Database Servers

Prod

Stage

Prod

Dev

Changeset Server

Prod

Stage

Dev

GNU R

Slony replication

DBI-Link
Schemas and Slony

- Replication via Slony from production server to dev and stage
  - Slony provides asynchronous table replication; blocks writes on slave nodes
- Duplicate tables for metadata in another schema: stage has stage schema, dev has dev schema
  - Change search_path to see different results
Search Path gotchas

- SET search_path / SHOW search_path
- Foreign key constraints explicitly remember search path from time of creation
- Remember default search_path is $user,public (don’t create $user schema unless you really want it!)
Search Path gotchas (cont.)

• Functions cache query plan
  – If run same function in same connection twice with different search_path settings, connection will use first search_path setting for function
    • Can explicitly pass in set search_path to function, but may not be what you want
  – Simpler to reconnect

• If you create tables like this:

```sql
CREATE TABLE foo (LIKE prod.foo INCLUDING INDEXES INCLUDING CONSTRAINTS INCLUDING DEFAULTS)
```

remember to drop default on any synthetic keys and explicitly re-create sequence in new schema; and to check path of foreign key constraints
Replication possibilities

- Synchronous replication
  - WAL shipping (but only warm standby)
  - Hot standby in 8.5?
- Asynchronous replication
  - Slony
  - Command Prompt Replicator (Mammoth)
  - Londiste
  - Golconde
  - Bucardo
Slony basics

- Adds triggers to replicated tables
  - System catalog hackery if Postgres version < 8.3
- Slave node tables are read-only
- Writes event log that tracks data changes on replicated tables into its own schema
- Replication sets can be made of one or many tables
- Different nodes can replicate different sets
- Requires one slon daemon per node to push/pull changes
- Nodes can be cascaded
Slony gotchas

- All schema changes on replicated tables must be added through EXECUTE DDL scripts.
- Asynchronous replication means slaves can fall behind, especially on large deletes/copies. \copy on master node = inserts on slave node.
- Never truncate replicated tables!
Slony gotchas -- continued

- Foreign keys on non-replicated tables that references replicated tables must be dropped through Slonik EXECUTE DDL scripts
- All triggers on replicated tables are automatically disabled on slave nodes. To enable triggers on slave nodes, use Slonik STORE TRIGGER command (not true in Slony 2.x).
- Functions must be kept in sync manually!
Reconciling Databases

• Three different problems
  • Reconciling data
  • Reconciling table structures
  • Reconciling functions, indexes, constraints, triggers and views
Slony replication and different schemas give us prod/stage/dev database environments – but ...
Issue 1:

How do we reconcile data between the different databases and schemas?
Easy stuff

- Between stage and public and test and public, since both live in the same database

- But usually we need to compare test to stage, since we're pushing database changes this way...

Oh oh... no cross-database queries...
How to do cross-database queries?

- Outside database:
  - pg_comparator (http://www.coelho.net/pg_comparator/)
  - Home-grown

- Inside database:
  - Dblink
  - DBI-Link
DBI-Link

- David Fetter’s project (http://pgfoundry.org/projects/dbi-link/) to treat heterogenous data sources as tables within Postgres
- Uses plperl + DBI to connect to other datasources; uses empty tables and rules to make other datasources seem like different schema in host database
DBI-Link setup

- CREATELANG plperlu linkdb
- Install DBD::Pg on database server
- `psql -d linkdb -f dbi_link.sql`

```sql
SELECT make_accessor_functions(
    'dbi:Pg:devdb:dev_server',
    'username','password",
    '__'
AutoCommit:1
RaiseError:1
'':dbi_link.yaml, --YAML for DBI connection attributes
NULL::dbi_link.yaml, --YAML for DBI connection environment
    'dev', --Remote schema name
    NULL::text, --Remote catalog name
    'dev') --Local schema name
```
DBI-Link setup (continued)

- Should now be able to run simple queries on remote tables as if in link database
- If remote schema changes, must do the following:
  - DROP SCHEMA dev CASCADE;
  - SELECT
dbi_link.refresh_schema(data_source_id)
FROM
dbi_link.dbi_connection
WHERE local_schema = 'dev';
DBI-Link gotchas

- Can’t use indexes; must load entire contents of table into memory
- Currently, no real support for arrays
How do we figure out what’s changed?

- EXCEPT queries between same tables in different schemas?
  - Gets inserts or deletes -- but how do we figure out updates?
  - Dynamic query that compares contents of fields will break when one schema’s field has a null and the other doesn’t
    - If master.foo is null and slave.foo = 3, then (master.foo = slave.foo) = NULL!
Use ROW(*) comparison

- SELECT sf.ROW(*), pf.ROW(*)
  from stage.foo sf
  full outer join public.foo pf
  on sf.pkey = pf.pkey
  WHERE sf.ROW(*) IS DISTINCT
  FROM pf.ROW(*)

  - Handles nulls for you: (null compared to
    non-null) = false, not null!
Create changeset from `ROW(*)` query

- Cast `ROW` back up to recordset
- If right side is null, need to insert row
- If left side is null, need to delete row
- If both sides have content, need to update row
  - Easy to make reversal set at same time
-- reconciling test.foo v. stage.foo
(bar_int int, baz_text text, primary key bar_int)

SELECT
    -- rename fields so easier to distinguish master vs. slave field
    master_row.bar_int AS bar_int_m,
    master_row.baz_text AS baz_text_m,
    slave_row.bar_int AS bar_int_s,
    slave_row.baz_text AS baz_text_s,
    CASE WHEN slave_row IS NULL
        THEN 'I'::varchar
        WHEN master_row IS NULL
        THEN 'D'::varchar
        ELSE 'U'::varchar END AS ddl_action
FROM
    (SELECT ROW(m.*)::foo as master_row,
        ROW(s.*)::foo as slave_row
    FROM test.foo m
        FULL OUTER JOIN stage.foo s
            ON m.bar_int = s.bar_int
    WHERE row(m.*)::foo IS DISTINCT FROM row(s.*)::foo
    ) as x;
Problem: Must verify changesets

- Changes to metadata tables not all promoted at same time
- Changesets can get out of sync--need to verify content in changeset is still in database
Use digest to vet changeset / reverse changeset

- Before applying changeset, regenerate all differences. Take MD5() of DDL statements.
  - Strict commit: if any line in changeset is missing md5 entry in current set of differences, throw out whole set
  - OR
  - Relaxed commit: if any line in changeset is missing md5 entry, throw out line
Edge cases

• Sometimes metadata tables need to be combined or need to overlap:
  • Edge case 1: discrete additions added to large table
  • Edge case 2: overlapping/conflicting data to replace records in large table
Edge Case I -- Discrete additions to large table

- Want to test 100K new prices for new market
  - No overlap with current data
  - Don’t want to add to public schema if not useful
- Use inheritance and schemas
  - Set search_path to dev, public;
  - ALTER TABLE public.price INHERITS dev.price
  - Hides dev.price if search_path=public; shows dev.price + public.price if search_price=dev,public
- Simple select * from price will worrk
Edge Case II -- Replacing prices

• Want to test replacing 100K prices for already existing market -- some dates are accurate in production
• Since prices are different between schema, normal query will show both series of prices
• Have to create custom accessor function or view
Example of accessor function

-- Price table:
-- (price_id serial primary key not null,
--  contract_id int not null, sampletime_id int not null,
--  dte timestamp, open numeric, high numeric,
--  low numeric, settle numeric, volume numeric, open_interest numeric)
--  with candidate key (contract_id, sampletime_id, dte);
CREATE OR REPLACE FUNCTION get_prices(wk_contract_id int, wk_sampletime_id int) AS $$
SELECT price_id, dte, open, high, low, settle, volume, open_interest
FROM ONLY test.price t_pub
WHERE contract_id = $1
AND sampletime_id = $2
UNION
SELECT price_id, dte, open, high, low, settle, volume, open_interest
FROM only public.price p_pub
WHERE contract_id = $1
AND sampletime_id = $2
AND NOT EXISTS (SELECT NULL
                FROM ONLY test.price
                WHERE contract_id = $1
                AND sampletime_id = $2
                AND dte = p_pub.dte)
$$ LANGUAGE 'SQL';
Problems with these approaches

- If using synthetic primary keys, sequences may have to adjusted to avoid overlap
- Changing inheritance of public table will break identity of schema across nodes
- Accessor functions hurt performance
Limitations

- Changesets can’t be used to promote changes in column definitions, defaults, constraints or triggers for tables.
- Changesets can’t be used to promote functions or views.
- Changesets can’t validate trigger function equality.
Limitations (continued)

- Inserts and updates in public schema may need to trigger events in other schemas

- Same trigger function wants to write to different table based on search_path of connection. Because of caching of function plans, may have to do the following:

```sql
DECLARE
    l_savesearchpath text;
    l_newsearchpath text;
BEGIN
    raise notice 'aggregate_updates (... with schema)';
    select into l_savesearchpath current_setting('search_path');
    raise notice 'Current search path is %', l_savesearchpath;
    select into l_newsearchpath set_config('search_path','' || $1 || ''','f');
    raise notice 'Using search path %', l_newsearchpath;
    perform aggregate_updates();
    select into l_newsearchpath set_config('search_path','' || l_savesearchpath || ''','f');
    raise notice 'Restored search path is %', l_newsearchpath;
```
Triggers vs. Messages

- Could use messages
  - Simplifies event handling
  - Alternatives:
    - LISTEN/NOTIFY (but can only take a NAME (no other payload)
    - PgQ
    - External message queue
      - ActiveMQ
      - RabbitMQ
      - WebsphereMQ
Issues 2 and 3:

How do we 2) reconcile DDL and 3) functions, triggers and views between databases and schemas?
Possible ideas

- **Issue 2:**
  - Script promotion of DDL changes, function changes via external application (e.g., Robert Brewer's post facto – [http://post-facto.org/](http://post-facto.org/))

- **Issue 3:**
  - Functions in source control; automated checkout and deployment
  - Test table definition, constraints, and function behavior and expected signatures via David Wheeler's pgTAP ([http://pgtap.projects.postgresql.org/](http://pgtap.projects.postgresql.org/))
    - PL/pgSQL version of TAP (Test Anything Protocol)
    - has_column(), col_not_null(), col_has_default() for column definitions; has_fk() for column constraints
    - has_trigger(), trigger_is() to test trigger unity
    - can(), can_ok() to test function behavior and signatures
Questions?