Reconciling and comparing databases redux

Profiling, testing and deploying DDL changes in multiple databases

Norman Yamada
The Millburn Corporation
nyamada@millburncorp.com

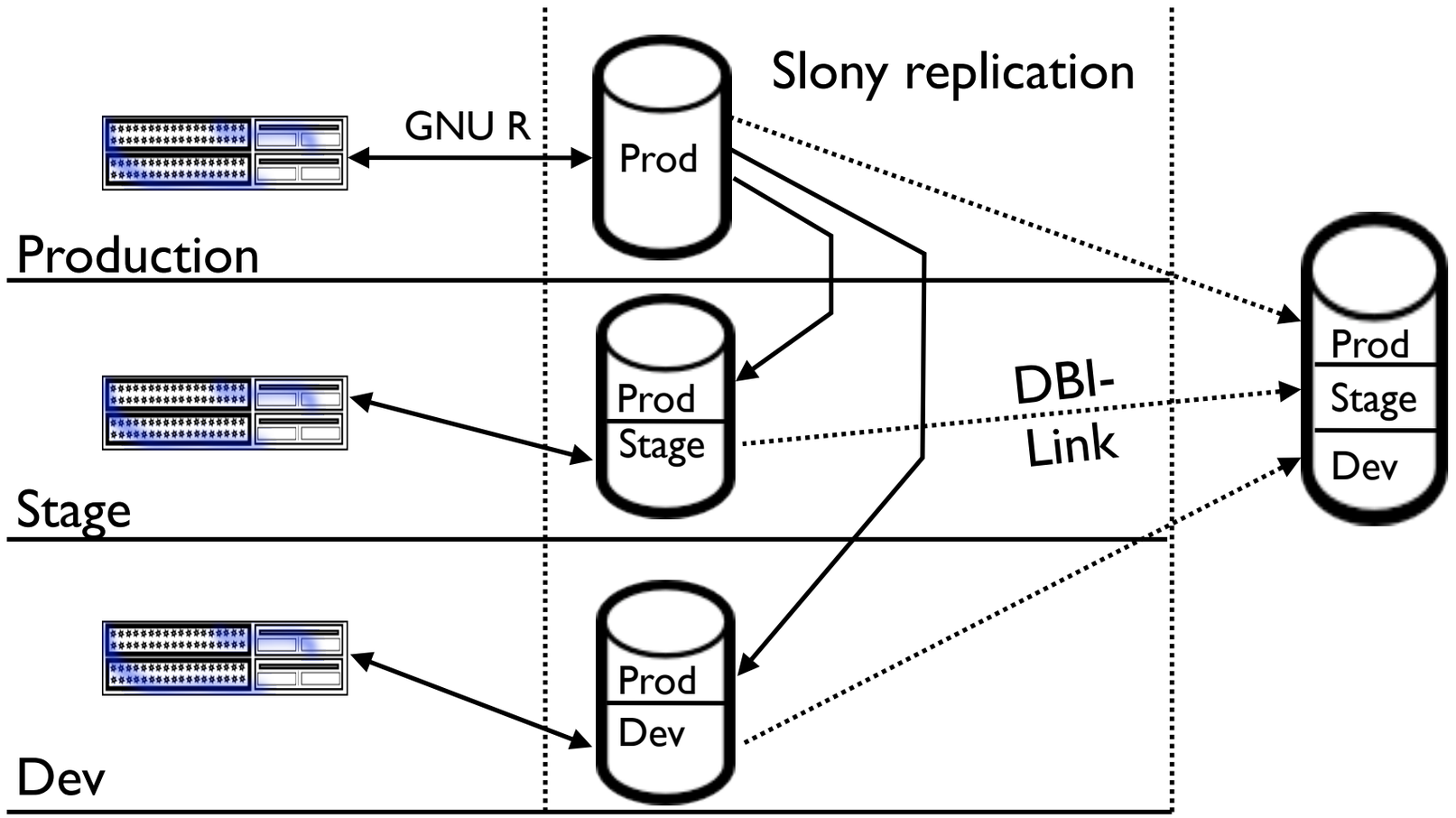
Millburn -- basic setup

- Hedge fund, specializing in commodity futures and currencies. Currently over $1.8 billion under management.
- Daily settlement price data from financial aggregators (Reuters/Bloomberg/CSI) stored in Postgresql; tickdata in OneTick (Proprietary database)
- Algorithmic models processed in Java/GNU R

Millburn's setup

- Main database has production, stage and dev nodes -- moderate size (100+ g)
- Stage and dev nodes use Slony to receive replicated data from production
- Staging and dev servers use independent schemas to rehearse both DDL and DML changes

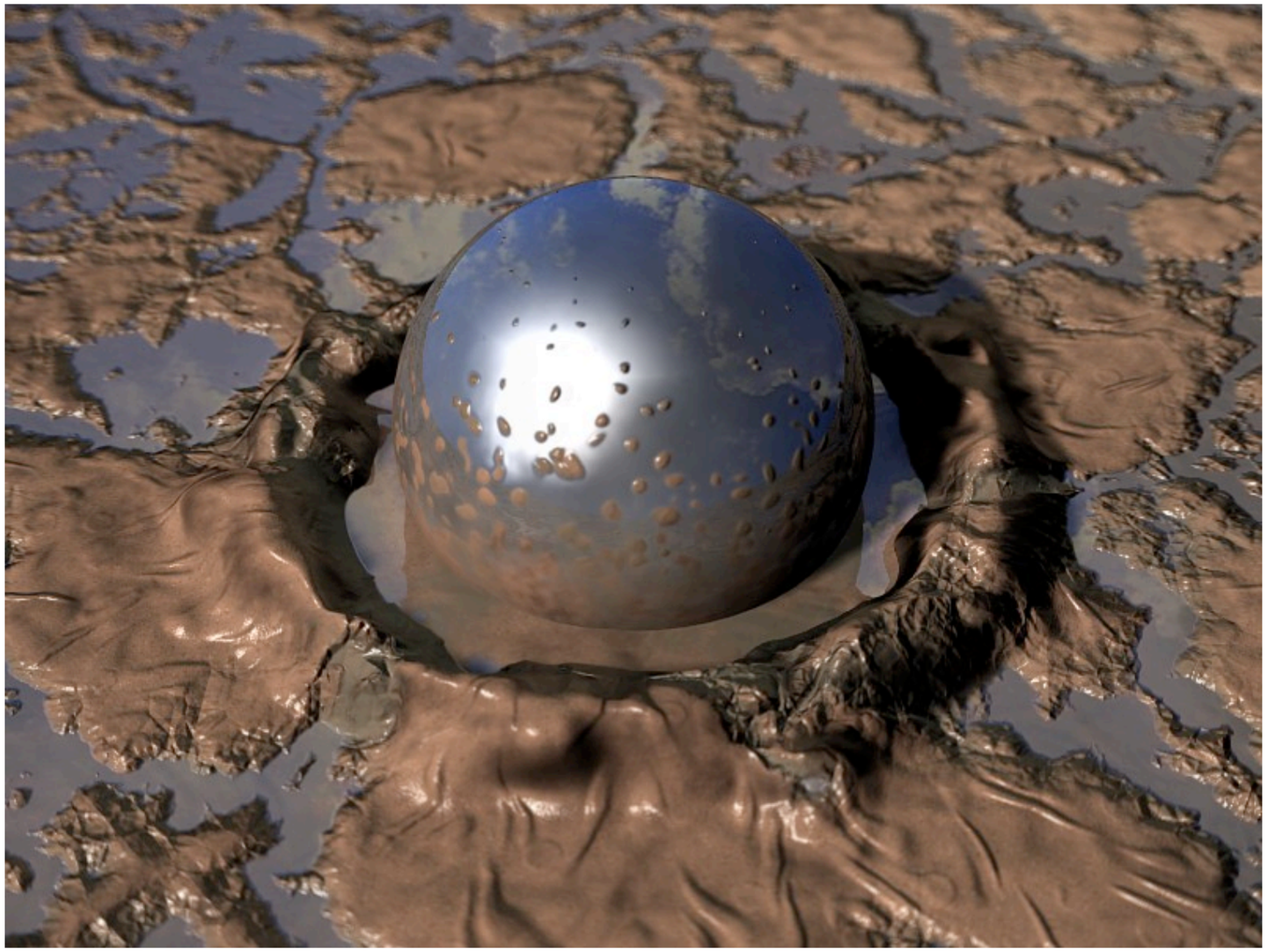
Millburn's setup (cont.)



Complex use case

- Stage and dev schemas not complete copies of prod schema; only have override tables and functions
- Trigger functions on prod schema may have to fire on stage and dev nodes and write to table in different location
- Continuous development cycle
- Bugs in prod can have immediate financial impact

Leads to



Big Ball of Mud

- Problems reconciling both data and DDL
- Metadata not smoothly promoted from test to stage to prod

- DDL problems
  - Indexes/Constraints/Functions missing in different schemas
  - Functions with different arguments in different schemas
  - Functions with same arguments, but different bodies in different schemas

Fixing Data Diffs

- Compare data between nodes (via DBI-Link):
  - Query to compare test.foo vs. stage.foo

```
--DDL for foo (bar_int int, baz_text text, primary key bar_int)
SELECT 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_test 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;
```

  - Custom app creates SQL changesets and rollback set. Sets can be applied in single transcation with rollback on error.

Sample changeset

Schema Comparison >> Comparing test to stage >> Differences for run_market

Description: sample

Save Changes | Cancel Edit | Delete Changeset

Diffs already in Changesets >> Editing Changeset # 1011

Filter: Check Uncheck Apply Filter Clear all

Sql	Delete
INSERT INTO run_market (market,sig_sysname,pos_sysname,status,exec_delay,signal_execution_delay,position_execution_delay,rmid) VALUES ('(2437) AO-SY-0 [1060]','(1327) ASPLRAF','(1268) VOLAT050HFA','A','00:00:00','1','1','13892')	☐
INSERT INTO run_market (market,sig_sysname,pos_sysname,status,exec_delay,signal_execution_delay,position_execution_delay,rmid) VALUES ('(2437) AO-SY-0 [1060]','(1328) ASPLRBF','(1268) VOLAT050HFA','A','00:00:00','1','1','13893')	☐
INSERT INTO run_market (market,sig_sysname,pos_sysname,status,exec_delay,signal_execution_delay,position_execution_delay,rmid) VALUES ('(2437) AO-SY-0 [1060]','(1329) ASPLRAR','(1268) VOLAT050HFA','A','00:00:00','1','1','13894')	☐
INSERT INTO run_market (market,sig_sysname,pos_sysname,status,exec_delay,signal_execution_delay,position_execution_delay,rmid) VALUES ('(2437) AO-SY-0 [1060]','(1330) ASPLRBR','(1268) VOLAT050HFA','A','00:00:00','1','1','13895')	☐

Uncommitted changes

Filter: Check Uncheck Apply Filter Clear all

Sql	Add
INSERT INTO run_market (market,sig_sysname,pos_sysname,status,exec_delay,signal_execution_delay,position_execution_delay,rmid) VALUES ('(2438) AUDB-SY-0 [1060]','(1327) ASPLRAF','(1268) VOLAT050HFA','A','00:00:00','1','1','13824')	☐
INSERT INTO run_market (market,sig_sysname,pos_sysname,status,exec_delay,signal_execution_delay,position_execution_delay,rmid) VALUES ('(2438) AUDB-SY-0 [1060]','(1328) ASPLRBF','(1268) VOLAT050HFA','A','00:00:00','1','1','13825')	☐

Cleaning up the Mud

- Profile DDL differences
- Tracking DDL changes in database daily
- Testing, testing, testing (pgTAP)
- Moving to automated deployment/rollback scripts
  - Difficulties:
  - Need to mix DML with DDL
  - Need to deploy in heterogenous environment (Slony on master node)

Profiling /deployment tools

- Another PostgreSQL Diff tool (http://apgdiff.sourceforge.net/)
- SQL Compare (http://www.red-gate.com/products/SQL_Compare) for SQL Server
- DB Solo (http://www.dbsolo.com)
- AquaFold (http://www.aquafold.com)

Reconciling Schemas

- How do we compare schemas?
  - If two databases and same schema, use Greg Sabino Mullane's check_postgres.pl script (http://bucardo.org/wiki/Check_postgres)
  - *check_postgres.pl --same_schema*
  - Significant patches for index comparison this week

Comparing dbmaster1 to dbstage1

Tables, views and functions are identical

Trigger comparison

Table	Trigger	Status	Action
contract	contract_before_ins_upd	disabled	
	trig_contract_after_update	**active**	**dbmaster1:** EXECUTE PROCEDURE contract_after_update(); **dbstage1:** EXECUTE PROCEDURE stage.contract_after_update()
equity_dividend	trigger_equity_dividend_after_iud	**active**	
equity_split	trigger_equity_split_after_iud	**active**	

But with functions, need more information

- Had to create function profile sql using pg_proc (not enough information in information_schema.routines)

Function profiling

```
SELECT x.nspname,x.proname,
x.proargnames,x.proretset,x.prorettype,
x.proargtypes,x.proallargtypes,
x.proargmodes,x.proisstrict,
x.provolatile,x.prosrc,x.proisagg,
x.md5_src,x.function_language,
coalesce(trg.tgname,'') as trigger_name,
coalesce(n.nspname ||'.','') ||coalesce(tbl.relname,'')
       as table_name
FROM
(
   SELECT quote_ident(nspname) as nspname,
          quote_ident(proname) as proname,
          proargnames,proretset,prorettype,
          proargtypes, proallargtypes,
          proargmodes,proisstrict,
          provolatile, prosrc,proisagg,
          md5(prosrc) as md5_src,
          pg_proc.oid as function_oid,
          l.lanname as function_language,
    FROM pg_proc JOIN pg_namespace n ON (n.oid = pronamespace)
                 JOIN pg_language l on (l.oid = prolang)
    WHERE nspname = [SCHEMA_NAME]
) as x left outer join pg_trigger trg
       join pg_class tbl
         join pg_namespace n
         on tbl.relnamespace = n.oid
       on trg.tgrelid = tbl.oid
     on x.function_oid = trg.tgfoid
```

Function profiling (cont'd)

- Application (mod_perl) then compares all functions between schemas to see if they're matching. Uses md5 sum of function body for quick comparison between functions with same name and same arguments

Check schema functions >> Compare functions between stage and public

206 functions examined in stage. Function names followed by * are called by triggers; Functions preceded by √ have been reviewed

Different	Missing	Identical
auto_scheduler()	aggregate_updates_from_run_market()	actual_write(wk_row trade_signal_type,wk_dte timestamp,wk_is_spread bool,wk_is_high_freq bool)
contract_after_update()*	bround(wk_number numeric,bprecision int4)	add_signal_mapping(isymbol varchar,iexchange varchar,istid int4,irollid int4,isymbol_out varchar,iexchange_out varchar,imonth varchar,iyear int4,ifund varchar,isystem varchar)
fund_market_system_effective_date_max(wk_fundid int4,wk_effective_date date)	clean_data(currency bpchar)	aggregate_updates()
generate_sf(int4 bpchar bool varchar)	clean_data(year bpchar,in_start timestamptz,in_end timestamptz)	aggregate_updates(isearchpath varchar)
generate_sf(int4 bpchar)	cleanse_data(in_curr bpchar)	currency_convert(int4 int4)
√ generate_sf_values(varchar date date)	cleanse_data(in_curr bpchar,in_cid int4,in_stid int4,in_source int4)	
get_hft_markets_to_run()		

Function profiling (cont'd)

- Web page allows drill down into function code; uses Text::Diff to see differences interactively

Check schema functions >> Compare stage to public >> handle_price(int4 bpchar)

Stage	Public

CREATE OR REPLACE FUNCTION handle_price(int4 bpchar)
RETURNS void AS
$$
 	
DECLARE
l_func VARCHAR;
l_ipfrecord RECORD;
l_pricerecord RECORD;
BEGIN
FOR l_ipfrecord IN
SELECT ('SELECT * FROM ' || ipf.func || '(' || p.pid || E',''' || $2 || E''')') as runfunc FROM instrument_price_function ipf, contract c, price p
WHERE
p.pid=$1
AND c.cid=p.cid
AND ipf.seid=c.seid
AND ipf.effective<=p.dte
AND ipf.expire>p.dte
ORDER BY priority ASC
LOOP
BEGIN
EXECUTE l_ipfrecord.runfunc;
END;
END LOOP;
END;
$$ LANGUAGE plpgsql VOLATILE
	CREATE OR REPLACE FUNCTION handle_price(int4 bpchar)
	RETURNS void AS
	$$
	
	DECLARE
	l_func VARCHAR;
	l_ipfrecord RECORD;
	l_pricerecord RECORD;
	BEGIN
	perform generate_updates($1,$2);
	FOR l_ipfrecord IN
	SELECT ('SELECT * FROM ' || ipf.func || '(' || p.pid || E',''' || $2 || E''')') as runfunc FROM instrument_price_function ipf, contract c, price p
	WHERE
	p.pid=$1
	AND c.cid=p.cid
	AND ipf.seid=c.seid
	AND ipf.effective<=p.dte
	AND ipf.expire>p.dte
	ORDER BY priority ASC
	LOOP
	BEGIN
	EXECUTE l_ipfrecord.runfunc;
	END;
	END LOOP;
	END;
	$$ LANGUAGE plpgsql VOLATILE

1 Record

Diffs

```
+--+-----------------------------------------------------------------------------+---+-----------------------------------------------------------------------------+
| 4|  l_ipfrecord RECORD;                                                        |  4|  l_ipfrecord RECORD;                                                        |
| 5|  l_pricerecord RECORD;                                                      |  5|  l_pricerecord RECORD;                                                      |
| 6|BEGIN                                                                        |  6|BEGIN                                                                        |
|  |                                                                             |* 7|  perform generate_updates($1,$2);                                           *
| 7|  FOR l_ipfrecord IN                                                         |  8|  FOR l_ipfrecord IN                                                         |
| 8|      SELECT ('SELECT * FROM ' || ipf.func || '(' || p.pid || E',''' || $2 |||  9|      SELECT ('SELECT * FROM ' || ipf.func || '(' || p.pid || E',''' || $2 |||
| 9|E''')') as runfunc FROM instrument_price_function ipf, contract c, price p  | 10|E''')') as runfunc FROM instrument_price_function ipf, contract c, price p  |
+--+-----------------------------------------------------------------------------+---+-----------------------------------------------------------------------------+
```

Tracking DDL changes

- DDL files for each schema is stored in subversion

```
                +----sequences
                |
                +----tables (also indexes, triggers, constraints, indexes, sequence ownership, primary key)
                |
                +----views
                |
[schema]--+----operators
                |
                +----functions (includes aggregates)
                |
                +----ddl_changes
```

- Subversion repository refreshed nightly by automated perl job
  - pg_dump -sx -n schema
  - split output into individual files
  - commit changes

Tracking DDL Changes (cont'd)

- Application uses perl and regular expression to parse pg_dump
  - Must adjust parsing script as pg_dump version changes
- This captures how the table changed over time; but you can't use subversion versions for pushing changes.
- Other possibilities: www.post-facto.org

Testing Functions

- How do we test functions (especially ones that write to tables) without side effects?
- Using David Wheeler's test framework: pgTAP (http://pgtap.projects.postgresql.org) -- create mock tables and use canned data from fixture files
- Fixture files to test both using actual production data and edge cases (which may never happen in production)

Sample test script setup

```
BEGIN;

\i fixtures/pg_prove_preamble.sql

\i fixtures/test_create_tickets/test_create_tickets_setup.sql
```

Pg_prove preamble

```
\set ON_ERROR_ROLLBACK 1
\set ON_ERROR_STOP true
-- suppress NOTICES and WARNINGS...
SET client_min_messages TO ERROR;

 -- set search_path to proper search_path for applications
drop schema if exists tap_test;
create schema tap_test;

--swallow output
\o /dev/null
SELECT public.set_tmc_search_path('tap_test','tap');
\o

-- load the function file if testing new build
--  \i ../foo_function.sql

-- start tests.
SELECT * FROM no_plan() ;
```

Search_path difficulties

```
CREATE OR REPLACE FUNCTION set_tmc_search_path(wk_prefix_path text,
wk_append_path text) RETURNS TEXT AS $$
DECLARE
  l_searchpath text;
BEGIN
  select into l_searchpath current_setting('search_path');
  PERFORM set_config('search_path',
             CASE WHEN wk_prefix_path IS NOT NULL THEN wk_prefix_path ||
','  ELSE '' END ||
             default_search_path ||
             CASE WHEN wk_append_path IS NOT NULL THEN ',' ||
wk_append_path ELSE '' END,
             'false')
  FROM tmc_hosts WHERE ip_addr=inet_server_addr();
  RETURN l_searchpath;
END;
$$ LANGUAGE plpgsql;
```

Fixtures file sample

```
-- clone table structure from other schema
 CREATE TABLE trade_signals
     (like trade_signals
     including defaults
     including constraints
     including indexes);
-- fix synthetic key
ALTER TABLE trade_signals ALTER trade_signals_id DROP DEFAULT;
CREATE SEQUENCE trade_signals_trade_signals_id_seq OWNED BY
trade_signals.trade_signals_id;
ALTER TABLE trade_signals ALTER trade_signals_id SET DEFAULT
    NEXTVAL('trade_signals_trade_signals_id_seq');
-- load data...
\copy trade_signals from fixtures/test_create_tickets/trade_signals.csv CSV
HEADER
\o /dev/null
select setval('trade_signals_trade_signals_id_seq',(Select max(trade_signals_id)
from trade_signals));
\o
```

Sample test

```
SELECT diag('confirm functions existence and signatures');
select has_function('get_broker_trade_profile');
select has_function('get_broker_trade_profile',
    ARRAY['integer','timestamp without time zone']);

-- test broker trades for normal tickets
-- data for CORN
-- cid 110038 = C CB N 2010
-- some simple variables
\set NO_BROKER 827
\set CORN_OUT 110038

select diag('testing get_broker_trade_profile...');

PREPARE expected_broker_assign as
select fundid,brokerid, brokerid as xbid,:NO_BROKER as pbid,cid
FROM (select cid,fundid,brokerid,sum(amount) as pos
      FROM ticket_view
      WHERE cid = :CORN_OUT
      group by 1,2,3
      having sum(amount) <> 0) as x
ORDER BY 1,2,3,4,5;

PREPARE actual_broker_assign as
SELECT * from get_broker_trade_profile(:CORN_OUT,localtimestamp)
ORDER BY fundid,brokerid,xbid,pbid,cid;

select results_eq('actual_broker_assign','expected_broker_assign',
'New ticket should have same brokers as previous ticket');
```

Sample test results

```
norman@apptest1:~/sql/tests$ pg_prove -h dbtest1 -d tmc --no-color -tv
test_create_tickets.sql

[14:18:49] test_create_tickets.sql ..
# confirm functions existence and signatures
ok 1 - Function get_broker_trade_profile() should exist
ok 2 - Function get_broker_trade_profile(integer, timestamp without time zone) should exist
# testing get_broker_trade_profile... (skip 20 tests)
ok 22 - New ticket should have same brokers as previous ticket
ok 23 - Ticket for new contract month should have same brokers as previous contract
# testing get_broker_trade_profile for HFT
ok 24 - Ticket for new contract month for hft should have same brokers as outright contract
in same month (by default) or as previous month in HFT
ok 50 - Expect default brokers for new trade for HF going in opposite direction from
outright to be same as outright broker choices
ok 51 - Expect that you can't change default brokers for new trade for HF going in opposite
direction from outright
1..51
ok    52144 ms
[14:18:49]
All tests successful.
Files=1, Tests=51, 53 wallclock secs ( 0.06 usr  0.00 sys + 11.05 cusr  0.50 csys = 11.61
```

Common DDL Changes

- CREATE OR REPLACE FUNCTION (AGGREGATE)
  - Flush cached connections if replacing function
- CREATE INDEX
  - Locks table writes (but not reads) during creation
- CREATE UNIQUE INDEX
  - Locks table writes (but not reads); can fail if data not unique
- CREATE (UNIQUE) INDEX CONCURRENTLY
  - Allows other writes, but can't be done in transaction
- CREATE TABLE
  - Relatively safe, unless create table includes foreign key constraints; in which case referenced table must be locked
- CREATE SEQUENCE / VIEW / DOMAIN
  - Should be safe
- CREATE RULE
  - Require exclusive lock on table
- CREATE TRIGGER
  - Locks table writes (but not reads);
- ALTER DOMAIN
- ALTER SEQUENCE
  - Should be relatively safe
- ALTER TABLE ADD/DROP (CONSTRAINT | COLUMN)
  - Require exclusive lock on table
- DROP FUNCTION
  - Flush cached connections
  - Function dependencies are not caught
- DROP TYPE (cascades to functions);
- DROP DOMAIN
- DROP RULE / SEQUENCE / TABLE
  - Requires exclusive lock on tables affected

Applying DDL Changes vs. Data Changes

- Usually require locking out writes; may involve recycling database connections
- Can be difficult to rollback; harder to validate and test
- May require updating or adding role permissions

Promoting DDL

- Create DDL scripts with rollback scripts
- Test promotion + rollback in single transaction
- Set permissions so that script can't write except into appropriate schema
- Dependent on table locking -- when can you promote DDL? Do you have a clear maintenance window?

Setting DDL permission

- Use roles that are updated automatically daily (schema_write / prod_read)
- Create temp user (DDL_applier_YYYYMMDD) with appropriate permissions
- Have superuser assign ownership to DDL applier based on objects in DDL script
- Drop temp user at end of DDL application run

Example DDL interactions

```
DDL script (with mixed DML):

CREATE TABLE my_lookup (foo int, primary key foo);
\copy my_lookup from [original_data_file]
ALTER TABLE my_table add column foo int;
update my_table set foo = (complex SQL);
ALTER TABLE my_table alter foo set not null;
ALTER TABLE my_table add foreign key (foo) references
my_lookup (foo);

Rollback:
ALTER TABLE my_table drop column foo;
DROP TABLE my_lookup;

Ownership script:
CREATE user tmp_ddl_applier PASSWD ‘secret’ ENCRYPTED
   IN ROLE dev_writer, prod_reader;
ALTER TABLE dev.my_table set owner to ddl_applier;

At end of DDL script:
REASSIGN OWNED BY tmp_ddl_applier TO [generic owner];
```

Simple promotion

```
norman@appdev1:~$ ./apply_ddl_changes.pl --db_schema=test --svn_ddl_root_dir=/home/norman/test_ddl_dir
2010-05-18 16:27:03,089 INFO - Settings:
db_host=dbtest1
db_schema=test
database=tmc
svn_ddl_root_dir=/home/norman/test_ddl_dir
svn_schema_dir=/home/norman/test_ddl_dir/test
ddl_last_applied_file=20100518_001_upsert_raw_prices.sql
ddl_stop_date=20100519
log_level=DEBUG
write_db=0
rollback=0
2010-05-18 16:27:03,090 DEBUG -

Creating temp user in database:
CREATE USER ddl_test_20100518 ENCRYPTED PASSWORD 'zcN2kC2{u' IN ROLE test_writer,public_reader
2010-05-18 16:27:03,196 DEBUG - Parsing 20100518_002_fix_ts_summary_view.sql
2010-05-18 16:27:03,198 DEBUG - Parsing 20100518_002_fix_ts_summary_view_rollback.sql
2010-05-18 16:27:03,200 DEBUG -

Applying ownership change script:
ALTER TABLE ts_summary OWNER to ddl_test_20100518;

2010-05-18 16:27:03,202 DEBUG -
Applying action script:
BEGIN;
SET SEARCH_PATH TO test, public;
create or replace view ts_summary [lots cut out];
# Rollback script played
create or replace view ts_summary [old stuff];
ROLLBACK;
2010-05-18 16:27:03,202 DEBUG - Applying /tmp/PBwpFtEL2L.sql
2010-05-18 16:27:03,203 DEBUG - export ON_ERROR_STOP=1 && export PGPASSFILE=/tmp/pg_pass_20100518 && psql
-q -X -l -d tmc -h dbtest1 -U ddl_test_20100518 -f /tmp/PBwpFtEL2L.sql 2>&1
2010-05-18 16:27:03,455 INFO - All changes simulated: applied and rolled back successfully
```

Promotion complexities

- Not every DDL changeset is applied at the same time
- How to push some changesets up to stage and omit others?

Rough notes

- Changeset named YYYYMMDD_[desc].sql / YYYYMMDD_[desc]_rollback.sql
- Changeset not applied unless rollback script exists
- Changeset automatically applied nightly in dev
- Application updates file in directory that stores last_file_applied
- Developer responsible for validating changeset (using simulate mode) and using SVN mv or SVN rm to delay or block changes
- Changes on stage and production run manually

Problems

- Some sets have dependencies on other sets
  - If assume unitary pipeline for deployment, then force sets to follow in sequential order
- Some DDL sets are too costly to simulate
  - Perhaps can be done in empty schema with no contention
- Replication...

DDL Replication

- Slony 1.2x in production requires global lock on every table in replication set before public DDL changes.
- Same applies for adding or subtracting table in different schema with foreign key constraints that reference tables in slony cluster
- DDL changes must be applied on all nodes via EXECUTE SCRIPT on master node and requires global lock on every table
- Slony 2.x should be better...

Possible directions

- Enforcement of test suite success before promotion
- Integration of SVN snapshot with rollback scripts for DDL Changes
- DDL Triggers in Postgres 9.1?
- More easily available tools for PostgreSQL?

Package Management?

Database development sucks, and I would like to fix it. I don't mean database system software development; that's awesome. ;-) I mean database development as in writing the tables, views, functions, and other code that make up your database. We have come further in recent years in PostgreSQL land. We have had the PL/pgSQL debugger, there is Piggly for PL/pgSQL code coverage, we have pgTAP for unit testing support, we have in Post Facto a version control system running inside the database, although I'll try to explain below why I think that that is not the right solution. My problem is that getting database code from the editor to the database server in a safe manner is pretty difficult. This already starts with deploying simple database code to a single server for the first time (as shown in this entry), and gradually gets more complicated when you need to update existing installations, manage multiple servers, or even multiple versions of that code.

My answer to that problem is an old friend: package management. Package managers such as dpkg and rpm are pretty well-established solutions and have shown over the years that managing software deployments can be easy and occasionally even fun.

http://petereisentraut.blogspot.com/2010/05/postgresql-package-management.html

Or?

Schema Compare™ for Oracle

New! Compares and synchronizes the schemas of Oracle databases

- Automatically compare and synchronize the schemas of Oracle databases
- Eliminate mistakes deploying database changes from dev, to test, to production
- Troubleshoot and fix errors caused by differences in database schemas
- Supports Oracle 10g, 11g

from $295

($369 including support & upgrades)

Download 14-day free trial

Get a quote

Purchase this now

Product information

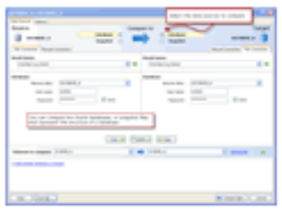
- Product overview
- Walk-through
- Requirements
- Forum
- Support and documentation
- Provide feedback
- License

"One word.... WOW!!!! Exactly what I'm looking for. With multiple developers possibly posting changes into a dev database, this allows me to generate a script to sync up and compare the two schemas. Works like a charm."

Al Drozd, Senior Application Developer, Hewlett Packard

Screenshots

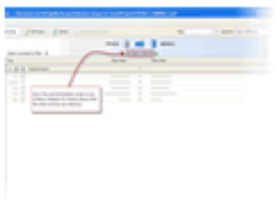
Schema Compare for Oracle allows you to compare and synchronize the schemas of Oracle databases in three simple steps:



1) Select data sources

2) Select objects to synchronize

3) Run the Synchronization Wizard



The data sources are identical

Questions?

- Comments?
- Criticisms?
- We’re hiring !