Common Migration Mistakes

Why?

- Project deadline
  - Looming Oracle renewal
- Lack of education
- Attitude
  - Only see the world through an Oracle lens
- Using migration tools or other short cuts

System Tuning

- When moving to PostgreSQL, many admins start with configuring values similar to the Oracle settings

- “My SGA was set to 16GB so shared_buffers is 16GB”
- “My redo logs are 2GB so max_wal_size is 2GB”

System Tuning

- In Oracle, it is possible to get better performance with a 32k block size

```
configure –with-blocksize=32
make
make install
```

Uppercase Folding

- In Oracle, all meta-data folds to uppercase

```
SQL> DESC USERS
 Name           Null?   Type
 ---------- ------- ------------------------
 FNAME                  VARCHAR2(100)
 MNAME                  VARCHAR2(100)
 LNAME                  VARCHAR2(100)
```

Uppercase Folding

- In PostgreSQL, all meta-data folds to lowercase

```
test=# \d users
            Table "public.users"
 Column |          Type          | Nullable
--------+------------------------+---------
 fname  | character varying(100) |
 mname  | character varying(100) |
 lname  | character varying(100) |
```

Uppercase Folding

- Many migration tools carry the uppercase from Oracle over to PostgreSQL

```
test=# \d "USERS"
            Table "public.USERS"
 Column |          Type          | Nullable
--------+------------------------+----------
 FNAME  | character varying(100) |
 MNAME  | character varying(100) |
 LNAME  | character varying(100) |
```

Uppercase Folding

- Becomes very tedious needing to double quote everything

```
test=# SELECT "FNAME", "MNAME", "LNAME" FROM "USERS";
  FNAME   | MNAME   |   LNAME
----------+---------+------------
 George   |         | Washington
 John     |         | Adams
 Thomas   |         | Jefferson
 James    |         | Madison
 James    |         | Monroe
 Andrew   |         | Jackson
 Martin   |         | Van Buren
 John     |         | Tyler
 John     | Quincy  | Adams
 William  | Henry   | Harrison
(10 rows)
```

Table Spaces

- In Oracle, table spaces are critical for storing data
- Generally many table spaces are used for indexes and tables

```
CREATE TABLESPACE ts_data1
  LOGGING
  DATAFILE '/data/ts_data1.dbf'
  SIZE 32m
  AUTOEXTEND ON
  NEXT 32m MAXSIZE 2048m
  EXTENT MANAGEMENT local;
```

Table Spaces

- In PostgreSQL, table spaces are just directory locations
- Provide no real benefit unless the database spans multiple mount points

```
CREATE TABLESPACE ts_data1
  LOCATION '/data/ts_data1';
```

Table Spaces

- Additional table spaces makes operations more cumbersome like
  - Backup and restore
  - Replication setup
  - Major version upgrades

Dual Table

```
SQL> SELECT SYSDATE FROM DUAL;

SYSDATE
---------
09-MAY-17
```

Dual Table

- In PostgreSQL, the FROM clause is optional and is unnecessary
- Don’t mock a DUAL table

```
test=# SELECT CURRENT_DATE;
 current_date
--------------
 2017-05-09
(1 row)
```

Exceptions

- Many Oracle procedures use exceptions as part of standard practice
  - Application developers are comfortable catching exceptions
- Some applications have exception handling in every procedure and function
- Most migration tools simply translate the code to pl/pgsql

Exceptions

```
CREATE FUNCTION get_first_name(p_lname varchar2)
  RETURN varchar2
IS
  l_fname varchar2(100);
BEGIN
    SELECT  fname
      INTO  l_fname
      FROM  users
     WHERE  lname = p_lname;

    RETURN  l_fname;
EXCEPTION
    WHEN no_data_found THEN
      l_fname := null;
    RETURN l_fname;

END get_first_name;
```

Exceptions

```
CREATE FUNCTION get_first_name(p_lname varchar) RETURNS varchar
  AS $$
DECLARE
  l_fname varchar;
BEGIN
    SELECT  fname
      INTO  l_fname
      FROM  users
     WHERE  lname = p_lname;

    RETURN  l_fname;
EXCEPTION
    WHEN no_data_found THEN
      l_fname := null;
    RETURN l_fname;

END$$ LANGUAGE plpgsql;
```

Exceptions

- PostgreSQL uses sub transactions to handle exceptions

```
CREATE OR REPLACE FUNCTION get_first_name(p_lname varchar)
  RETURNS varchar
AS $$
DECLARE
  l_fname varchar := null;
BEGIN
    SELECT  fname
      INTO  l_fname
      FROM  users
     WHERE  lname = p_lname;

    RETURN  l_fname;
END
$$ LANGUAGE plpgsql;
```

Exceptions

- Not all Oracle exceptions are considered PostgreSQL exceptions

```
CREATE FUNCTION get_first_name(p_lname varchar) RETURNS varchar
  AS $$
DECLARE
  l_fname varchar;
BEGIN
    SELECT  fname
      INTO  l_fname
      FROM  users
     WHERE  lname = p_lname;

    RETURN  l_fname;
EXCEPTION
    WHEN no_data_found THEN
      l_fname := 'NOT_FOUND';
    RETURN l_fname;
END$$ LANGUAGE plpgsql;
```

Exceptions

- Not found and too many rows are not PL/pgSQL exceptions

```
(jim@[local]:5432) [gnb] > SELECT get_first_name('missing');
 get_first_name
----------------
 (null)
(1 row)
```

Exceptions

- The keyword STRICT needs to be used to get Oracle like behavior

```
CREATE FUNCTION get_first_name(p_lname varchar) RETURNS varchar
  AS $$
DECLARE
  l_fname varchar;
BEGIN
    SELECT  fname
      INTO  STRICT l_fname
      FROM  users
     WHERE  lname = p_lname;

    RETURN  l_fname;
EXCEPTION
    WHEN no_data_found THEN
      l_fname := 'NOT_FOUND';
    RETURN l_fname;
END$$ LANGUAGE plpgsql;
```

Fine Tuning Queries

"I added a hint to use an index but PostgreSQL does not use it"

- PostgreSQL does not have hints as part of the core database
  - It treats Oracle hints as comments
- PostgreSQL's optimizer is different than Oracle so queries are tuned differently

Fine Tuning Queries

“I didn’t index my column in Oracle, why would I in PostgreSQL?”

- PostgreSQL has more and different types of indexes than Oracle

- B-tree
- Hash
- GIN
- GiST
- SP-GiST
- BRIN

Fine Tuning Queries

- PostgreSQL can even use indexes on LIKE queries

```
CREATE INDEX idx_users_lname
  ON users USING gin (lname gin_trgm_ops);
EXPLAIN SELECT * FROM users WHERE lname LIKE '%ing%';
                    QUERY PLAN
-----------------------------------------------------
 Bitmap Heap Scan on users  (cost=8.00..12.02 rows=1 width=654)
   Recheck Cond: ((lname)::text ~~ '%ing%'::text)
   ->  Bitmap Index Scan on idx_users_lname
         (cost=0.00..8.00 rows=1 width=0)
         Index Cond: ((lname)::text ~~ '%ing%'::text)
```

Not Using Native Features

- PostgreSQL is more feature rich for developers than Oracle
  - Stored Procedure Languages
  - Foreign Data Wrappers
  - Data Types
  - Spatial

Not Using Native Features

```
CREATE OR REPLACE FUNCTION has_valid_keys(doc json)
    RETURNS boolean AS
$$
    if (!doc.hasOwnProperty('fname'))
        return false;

    if (!doc.hasOwnProperty('lname'))
        return false;

    return true;
$$ LANGUAGE plv8 IMMUTABLE;


ALTER TABLE user_collection
    ADD CONSTRAINT collection_key_chk
      CHECK (has_valid_keys(doc::json));
```

Not Using Native Features

```
CREATE TABLE login_history (
  user_id   bigint,
  host      inet,
  login_ts  timestamptz
);

SELECT user_id, count(*)
  FROM login_history
 WHERE host << '17.0.0.0/8'::inet
   AND login_ts > now() - '7 days'::interval
 GROUP BY 1;
```

Synonyms

“PostgreSQL doesn’t have synonyms so I can’t migrate my application”

```
CREATE PUBLIC SYNONYM emp
FOR SCOTT.emp;
```

- Synonyms are used to not fully qualify cross schema objects
- Mostly a convenience feature

Synonyms

- In PostgreSQL, search_path can accomplish many of the same things and is less tedious to setup

```
test=# show search_path;
   search_path
-----------------
 "$user", public
(1 row)
```

Synonyms

```
CREATE FUNCTION user1.get_int()
  RETURNS int AS
$$
  SELECT 1;
$$ LANGUAGE sql;

CREATE FUNCTION user2.get_int()
  RETURNS int AS
$$
  SELECT 2;
$$ LANGUAGE sql;

CREATE FUNCTION public.get_number()
  RETURNS float8 AS
$$
  SELECT 3.14::float8;
$$ LANGUAGE sql;
```

Synonyms

```
test=# SELECT get_int();
2017-05-08 17:38 EDT [28855] ERROR:  function get_int() does not ...
2017-05-08 17:38 EDT [28855] HINT:  No function matches the given...
2017-05-08 17:38 EDT [28855] STATEMENT:  SELECT get_int();
ERROR:  function get_int() does not exist
LINE 1: SELECT get_int();
               ^
HINT:  No function matches the given name and argument types. You...
```

Synonyms

```
test=# SET search_path = user1, user2, public;
SET

test=# SELECT get_int();
 get_int
---------
       1
(1 row)
```

Synonyms

```
test=# SET search_path = user2, user1, public;
SET

test=# SELECT get_int();
 get_int
---------
       2
(1 row)
```

Synonyms

```
test=# select get_number();
 get_number
------------
       3.14
(1 row)
```

Nulls

- PostgreSQL and Oracle handle nulls a bit differently
  - Need to account for them appropriately
  - Most often seen with string concatenation

Nulls

```
CREATE TABLE users (
    fname  VARCHAR2(100),
    mname  VARCHAR2(100),
    lname  VARCHAR2(100)
);

SELECT
 fname || ' ' || mname || ' ' || lname
FROM users;
```

Nulls

```
SQL> SELECT fname || ' ' || mname || ' ' || lname FROM users;

FNAME||''||MNAME||''||LNAME
---------------------------------------------------------------
George  Washington
John  Adams
Thomas  Jefferson
James  Madison
James  Monroe
Andrew  Jackson
Martin  Van Buren
John  Tyler
John Quincy Adams
William Henry Harrison

10 rows selected.
```

Nulls

```
test=# SELECT fname || ' ' || mname || ' ' || lname FROM users;
        ?column?
------------------------




 John Quincy Adams
 William Henry Harrison
(10 rows)
```

Nulls

```
test=# SELECT COALESCE(fname, '') || ' ' ||
             COALESCE(mname, '') || ' ' ||
             COALESCE(lname, '') FROM users;
        ?column?
------------------------
 George  Washington
 John  Adams
 Thomas  Jefferson
 James  Madison
 James  Monroe
 Andrew  Jackson
 Martin  Van Buren
 John  Tyler
 John Quincy Adams
 William Henry Harrison
(10 rows)
```

Data Types

- Oracle has a few main data types that are typically used
  - VARCHAR2
  - DATE
  - NUMBER
- And a couple Large Object types
  - CLOB
  - BLOB

Data Types

- PostgreSQL comes with 64 base types and can be extended for more

abstime	int2	pg_lsn	smgr
aclitem	int2vector	pg_node_tree	text
bit	int4	point	tid
bool	int8	polygon	time
box	interval	refcursor	timestamp
bpchar	json	regclass	timestamptz
bytea	jsonb	regconfig	timetz
char	line	regdictionary	tinterval
cid	lseg	regnamespace	tsquery
cidr	macaddr	regoper	tsvector
circle	money	regoperator	txid_snapshot
date	name	regproc	uuid
float4	numeric	regprocedure	varbit
float8	oid	regrole	varchar
gtsvector	oidvector	regtype	xid
inet	path	reltime	xml

Data Types

- Don’t assume that the perceived equivalent in PostgreSQL behaves the same as Oracle
- For example, managing CLOBS
  - Length
  - Substrings

```
DBMS_LOB.GETLENGTH(x)
```

Data Types

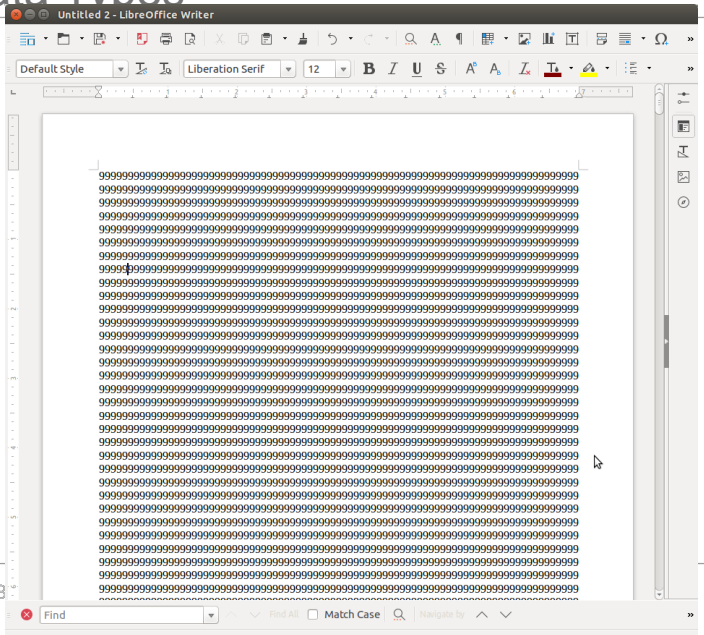
- In PostgreSQL, VARCHAR and TEXT are equivalent and behave the same

```
CREATE TABLE max_varchar (
    a  varchar(4001)
);

CREATE TABLE max_varchar (
    a  varchar(10485760)
);
```

Data Types

```
test=# INSERT INTO max_varchar SELECT repeat('x', 1073741800);
INSERT 0 1

test=# SELECT length(a) from max_varchar ;
   length
------------
 1073741800
(1 row)
```

Data Types



Data Types

- Most migration tools translate an Oracle NUMBER to a PostgreSQL NUMERIC
- A PostgreSQL NUMERIC can hold
  - 131072 before the decimal point
  - 16383 after the decimal point

- It is not the same are NUMBER

```
SELECT to_number(n, n)
FROM repeat('9', 131071) n;
```

Summary

- System Tuning
- Case Folding
- Table Spaces
- Dual Table
- Exceptions

- Fine Tuning
- Native Features
- Synonyms
- Nulls
- Data Types