
Migration Overview

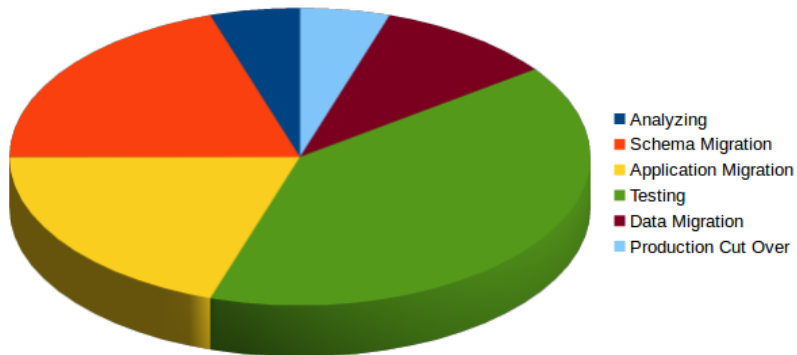
Why Migrate

- Financial
- Forced by vendor
- Technical Flexibility

Migration Phases

- Analyzing
- Schema Migration
- Application Migration
- Testing
- Data Migration
- Production Cut Over

Migration Breakdown



Migrations

- The migration project is really migrating an application, not a database
- A single migration project may actually migrate 5 or more databases
 - Development
 - QA
 - Staging
 - Production
 - Disaster Recovery

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	<u>character</u> varying(100)	
MNAME	<u>character</u> varying(100)	
LNAME	<u>character</u> 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;
```

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 timestampz  
);  
  
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

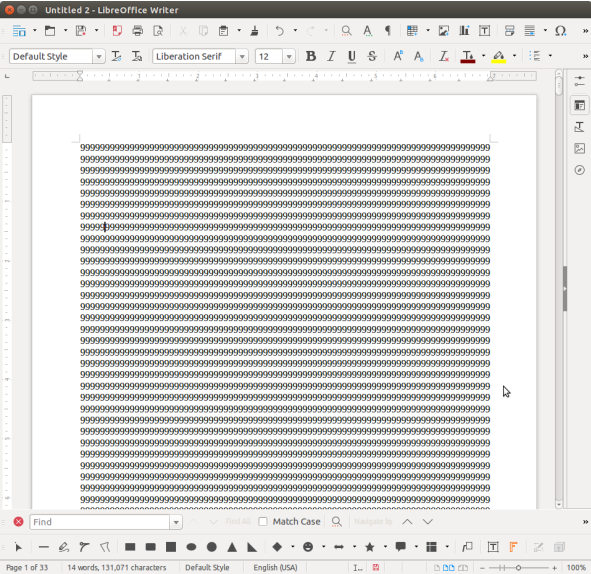
```
test=# INSERT INTO max_varchar SELECT repeat('x', 1073741800);  
INSERT 0 1
```

```
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 as NUMBER

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

Analyzing

Determining Candidates

- Look at the entire portfolios of applications
- Split the portfolio into 2 high level buckets
 - 3rd party applications
 - Home grown applications

3rd Party Applications

- Split the 3rd party applications into 2 sub groups
 - Applications that do support Postgres
 - These are prime candidates
 - Applications that do not support Postgres
 - These are potentially trapped
 - Note the alternative databases if any

Home Grown Applications

- Very varied set of challenges
 - Age of the application
 - Size of the data
 - Downtime window
 - Specialized features
 - Data access pattern
 - Application development language
 - Java
 - .NET
 - C/C++

Older Applications

- Usually very intense use of server side logic
 - In the client/server era, most business logic resided in stored procedures
- Brain drain
 - Talent that wrote the application have moved on
- These are usually the scariest applications to move, but have the largest upside

Large Databases

- Sheer data movement becomes a significant factor in the migration

Downtime Window

- Some applications are 24x7 with very small maintenance windows
 - Coordinate effort needed for a production cut over
- Applications with nightly downtime windows are ideal

Specialized Features

- Partitioning
- Spatial
- XML
- Flashback Query
- Full Text Search

Data Access Pattern

- Read mostly
- Append only
- Update intensive
- Insert and purge
- Nightly batch

Development Language

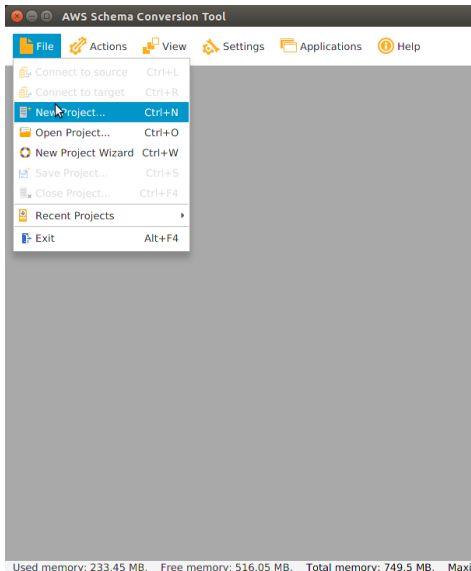
- Java
- .NET
- C/C++
- Perl/Python/PHP
- ORMs in use?

Scanning the Schema

- Many tools available for scanning an existing schema, but 2 mainly used
- AWS Schema Conversion Tool (SCT)
 - Creates an assessment report highlighting the areas of a database that will require manual effort
 - Free and closed source
 - GUI
- Ora2PG
 - Creates an assessment report for all schema objects
 - Free and open source
 - Command line

Using SCT

- Everything is project based
- A project has a source and target database



Creating an SCT Project

- Define the source and target database types
 - OLTP vs OLAP changes the available choices
- Targets are all listed as Amazon RDS endpoints, but can be a local PostgreSQL database

New Project

Enter the name, location and type of the new migration project.

Project name: Migration Training

Location: /tmp/MigrationProjects

Transactional Database (OLTP)
 Data Warehouse (OLAP)

Source Database Engine: Oracle

Target Database Engine: Amazon RDS for PostgreSQL

Specify an Oracle source

- Ensure the machine running SCT can connect to the Oracle database
- Use the standard set of connection parameters for Oracle

Connect to Oracle

Specify parameters for new connections to the source

Connection: SSL

Type:

Server name:

Server port:

Oracle SID:

User name:

Password:

Use SSL

Specify a Postres Target

- Running the initial assessment on a local instance of PostgreSQL may simplify things in some environments
- Use the same user name in PostgreSQL that is used in Oracle

Connect to Amazon RDS for PostgreSQL

Specify parameters for new connections to the target

Connection SSL

Server name localhost

Server port 5432

Database compiere

User name reference

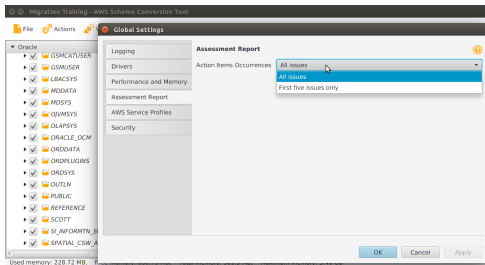
Password ●●●●●●●●●●

Use SSL

Test Connection Cancel OK

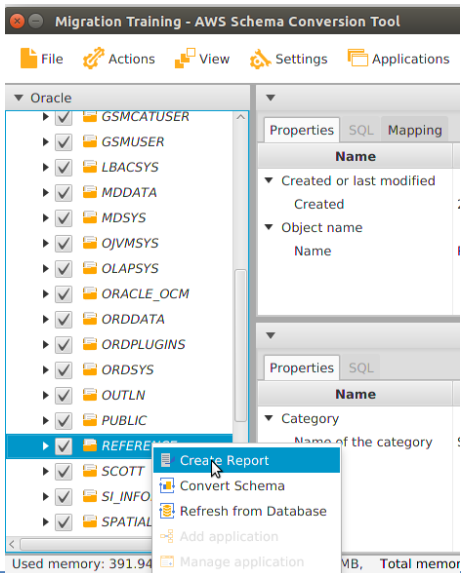
SCT Settings

- The default settings need to be adjusted to be really useful
- To accurately determine scope, all issues need to be shown



Creating an Assessment Report

- A report needs to be generated for each schema in Oracle
- Produces a simple PDF report



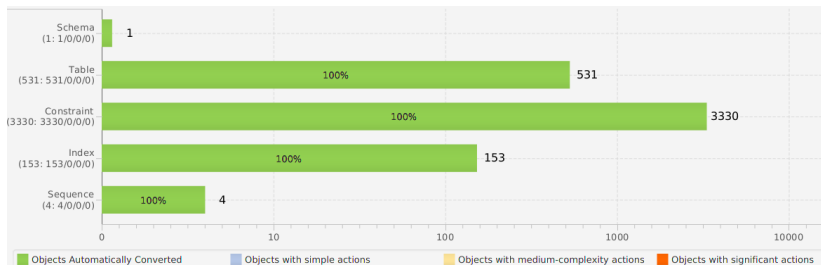
Analyzing the Report

We completed the analysis of your Oracle source database and estimate that 100% of the database storage objects and 87% of database code objects can be converted automatically or with minimal changes if you select Amazon RDS for PostgreSQL as your migration target.

- Conversion != Perform well
- The details matter

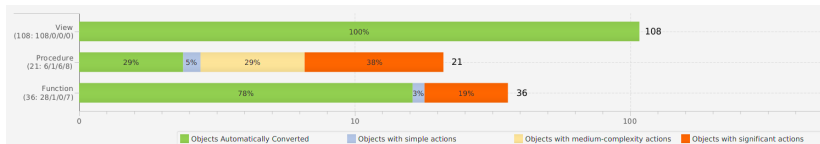
Analyzing Storage Objects

- Storage object definitions generally come over easily
 - Data types frequently need adjusting
- Partitioning or other more advanced constructs may come up here



Analyzing Database Code Objects

- Views frequently convert automatically
 - Focus on performance testing
- Other objects are usually the telling indicator of the complexity of a migration



Looking at the Details

- Issues typically follow patterns
- Can typically be categorized into 3 buckets
 - Ignore
 - Functional
 - Performance

- ▶ **Issue: 5028: Unable to convert unsupported datatypes**
Recommended action: The stub-code created. Please perform a manual conversion.
Number of occurrences: 5
- ▶ **Issue: 5030: Unable to convert complex usage of unsupported datatype objects**
Recommended action: The stub-code created. Please perform a manual conversion.
Number of occurrences: 8
- ▶ **Issue: 5118: Unable to convert an associative array declaration**
Recommended action: Use array or list-like types or temporary tables.
Number of occurrences: 1 | Documentation reference: <http://www.postgresql.org/docs/9.6/static/arrays.html>
- ▶ **Issue: 5198: PostgreSQL doesn't support GLOBAL TEMPORARY TABLE**
Recommended action: Try using a local temporary table.
Number of occurrences: 2 | Documentation reference: <http://www.postgresql.org/docs/9.6/static/sql-createtable.html>
- ▶ **Issue: 5320: PostgreSQL doesn't support the view with invalid status**
Recommended action: Perform a manual conversion.
Number of occurrences: 39 | Documentation reference: <http://www.postgresql.org/docs/9.6/static/sql-createtable.html>
- ▶ **Issue: 5334: Unable convert statements with dynamic SQL statement**
Recommended action: Please perform a manual conversion.
Number of occurrences: 5 | Documentation reference: <http://www.postgresql.org/docs/9.6/static/plpgsql-statements.html>
- ▶ **Issue: 5335: PostgreSQL doesn't support the GOTO operator**
Recommended action: Perform a manual conversion.
Number of occurrences: 5
- ▶ **Issue: 5340: Unable to convert functions**
Recommended action: Use suitable function or create user defined function.
Number of occurrences: 3

Ignorable Items

- Invalid objects typically make up the bulk of these
- These should be cleaned up regardless of a migration just as good practice

• **Issue: 5320: PostgreSQL doesn't support the view with invalid status**

Recommended action: Perform a manual conversion

Number of occurrences: 39 | Documentation reference: <http://www.postgresql.org/docs/9.6/stmts/sql-createview.html>

- ▶ View: **M_INOUT_LINE_VT** (Number of occurrences: 1)
- ▶ View: **RV_ASSET_SUMMONTH** (Number of occurrences: 1)
- ▶ View: **RV_OPENITEM** (Number of occurrences: 1)
- ▶ View: **RV_ORDERDETAIL** (Number of occurrences: 1)
- ▶ View: **RV_INOUTDETAILS** (Number of occurrences: 1)
- ▶ View: **C_RFRESPONSELINE_V** (Number of occurrences: 1)
- ▶ View: **RV_BPARTNEROPEN** (Number of occurrences: 1)
- ▶ View: **RV_CLICK_MONTH** (Number of occurrences: 1)
- ▶ View: **RV_CASH_DETAIL** (Number of occurrences: 1)
- ▶ View: **RV_C_INVOICE_WEEK** (Number of occurrences: 1)
- ▶ View: **M_INOUT_LINE_V** (Number of occurrences: 1)
- ▶ View: **C_INVOICELINE_V** (Number of occurrences: 1)
- ▶ View: **RV_C_INVOICE_PRODMONTH** (Number of occurrences: 1)
- ▶ View: **RV_C_INVOICE_VENDORMONTH** (Number of occurrences: 1)

Ignorable Items

- Some are less obvious, but can be determined by quickly scanning the object
- DBA maintenance routines will be different in PostgreSQL and many times not needed

```
▼ Issue: 5118: Unable to convert an associative array declaration
Recommended action: Use array or history types or temporary tables.
Number of occurrences: 1 | Documentation reference: http://www.postgresql.org/docs/9.6/static/arrays.html

▼ Procedure: DBA_RECOMPILE (Number of occurrences: 1)
Unable to convert the REFERENCE DBA_RECOMPILE.INVALID_TAB datatype declaration - PostgreSQL doesn't support associative arrays

▼ Oracle procedure: DBA_RECOMPILE

Properties SQL Mapping
023 v_Line VARCHAR(100);
024 v_PrintInfo CHAR(1) := 'N'; -- Diagnostic
025 --
026 CURSOR Cur_Invalids IS
027 SELECT object_id, object_name, object_type
028 FROM user_objects
029 WHERE status <> 'VALID'
030 AND object_type IN ('VIEW', 'PACKAGE', 'PACKAGE BODY', 'FUNCTION',
031 'PROCEDURE', 'TRIGGER', 'JAVA CLASS');
032 ORDER BY object_type, object_name;
033
034 CURSOR Cur_Valids (p_id NUMBER) IS
035 SELECT 'FOUND'
036 FROM user_objects
037 WHERE status = 'VALID'
038 AND object_id = p_id;
039
040 -- failed compile
041 TYPE invalid_tab IS TABLE OF Cur_Invalids%ROWTYPE INDEX BY BINARY_INTEGER;
042 invalid_tab_rec invalid_tab;
043
044
```

Functional Items

- There is usually a simple work around for these items
 - Requires manual intervention to know the correct one of many paths to take
- Usually a pattern that can be followed for other similar items

▼ **Issue: 5335: PostgreSQL doesn't support the GOTO operator**
Recommended action: Perform a manual conversion.
Number of occurrences: 5

▶ Procedure: **AD_COLUMN_SYNC** (Number of occurrences: 1)

▶ Procedure: **T_INVENTORYVALUE_CREATE** (Number of occurrences: 1)

PostgreSQL doesn't support the GOTO operator

▶ Procedure: **M_PRODUCTION_RUN** (Number of occurrences: 2)

▶ Procedure: **M_PRODUCT_BOM_CHECK** (Number of occurrences: 1)

▼ Oracle procedure: T_INVENTORYVALUE_CREATE

Properties	SQL	Mapping
083	--	
084	IF (SQL%ROWCOUNT = 0) THEN	
085	v_Message := '@Created = 0';	
086	GOTO FINISH_PROCESS;	
087	END IF;	
088		

Functional Items

- The solution of transaction control inside of a procedure can differ by procedure
 - Can be ignored
 - A foreign data wrapper (Database Link) can be used
 - Procedure can be redesigned

▼ Issue: 5350: Unable automatically convert statements that explicitly apply or cancel a transaction
Recommended action: Revise your code to try move transaction control on side of application.
Number of occurrences: 44 | Documentation reference: <http://www.postgresql.org/docs/9.6/interactives/pgap-structure.html>

▼ Procedure: C_BP_GROUP_ACCT_COPY (Number of occurrences: 3)

Unable automatically convert statements that explicitly apply or cancel a transaction

Unable automatically convert statements that explicitly apply or cancel a transaction

Unable automatically convert statements that explicitly apply or cancel a transaction

▼ Oracle procedure: C_BP_GROUP_ACCT_COPY

```
Properties SQL | Running  
045 update_rulac NUMBER := 0;  
046 Created_Total NUMBER := 0;  
047  
048 BEGIN  
049 -- Update AD_PInstance  
050 DBMS_OUTPUT.PUT_LINE('Updating PInstance - Processing ' || PInstance_ID);  
051 ResultStr := 'InstanceNotFound';  
052 UPDATE AD_PInstance  
053 SET Created = SysDate,  
054 IsProcessing = 'Y'  
055 WHERE AD_PInstance_ID=PInstance_ID;  
056 COMMIT;  
057
```


Performance Items

- Usually the more time consuming items to fix
 - Frequently very specific to Oracle
 - Exceptions fall into this category
 - (Should be functional, but its not)

```
▼ Issue: 5570: PostgreSQL doesn't support exception declaration
Recommended action: Review the exception used, and if possible convert it to another condition.
Number of occurrences: 1 | Documentation reference: http://www.postgresql.org/docs/9.4/static/elog/err-and-messages.html
► Procedure: M_PRICELIST_CREATE (Number of occurrences: 1)
▼ Oracle procedure: M_PRICELIST_CREATE
Properties | SQL | Message
-----|-----|-----
020 ...../
020 -- Logistics
021 ResultStr VARCHAR2(2000)
022 Message VARCHAR2(2000) := '';
023 Note EXCEPTION;
024 -- Parameter
025 CURSOR Cur_Parameter (PInstance NUMBER) IS
026 SELECT i.Record_ID, p.ParameterName, p.P_String, p.P_Number, p.P_Date
```

Using Ora2PG

- Everything is run via the command line with scripts and config files
 - ora2pg.conf is the main config file
- Allows for flexibility in scanning many schemas across many servers
- Learning curve is steeper than the GUI tools

Oracle Connection Information

- Set the Oracle home and standard connection details

```
# Set the Oracle home directory
```

```
ORACLE_HOME      /home/user1/development/oracle/instantclient_12_1
```

```
# Set Oracle database connection (datasource, user, password)
```

```
ORACLE_DSN       dbi:Oracle:host=192.168.122.215;sid=orcl
```

```
ORACLE_USER      reference
```

```
ORACLE_PWD       password1
```

Oracle User Details

- Turn off some functionality if the Oracle user does not have enough permissions

```
# Set this to 1 if you connect as simple user and can not extract things  
# from the DBA_... tables. It will use tables ALL_... This will not work  
# with GRANT export, you should use an Oracle DBA username at ORACLE_USER  
USER_GRANTS      0
```

Filtering Out Invalids

- Only try to migrate valid code

```
# Enable this directive to force Oracle to compile schema before exporting code.  
# This will ask to Oracle to validate the PL/SQL that could have been invalidate  
# after a export/import for example. If you set the value to 1 will exec:  
# DBMS_UTILITY.compile_schema(schema => sys_context('USERENV', 'SESSION_USER'));  
# but if you provide the name of a particular schema it will use the following  
# command: DBMS_UTILITY.compile_schema(schema => 'schemaname');  
COMPILE_SCHEMA 0
```

```
# If the above configuration directive is not enough to validate your PL/SQL code  
# enable this configuration directive to allow export of all PL/SQL code even if  
# it is marked as invalid. The 'VALID' or 'INVALID' status applies to functions,  
# procedures, packages and user defined types.  
EXPORT_INVALID 0
```

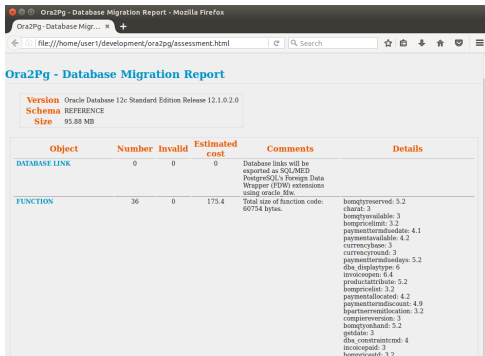
Running a Report

```
ora2pg -c ora2pg.conf -n REFERENCE --estimate_cost \  
      --cost_unit_value 20 --human_days_limit 30 \  
      --dump_as_html -t SHOW_REPORT > assessment.html
```

- “n” is the schema name
- “estimate_cost” generate a effort estimate
- “cost_unit_value” time duration to manually modify something (20 minutes)
- “human_days_limit” threshold that identifies a complex migration

Interpreting a Report

- Produces a simple and self contained HTML file
- Broken into several sections
- Does not identify problem areas



Ora2Pg - Database Migration Report

Version Oracle Database 12c Standard Edition Release 12.1.0.2.0
Schema REFERENCE
Size 95.88 MB

Object	Number	Invalid	Estimated cost	Comments	Details
DATABASE LINK	0	0	0	Database links will be exported as SQL/MED PostgreSQL's Foreign Data Wrapper (FDW) extensions using oracle fdw.	
FUNCTION	36	0	175.4	Total size of function code: 60754 bytes.	bonetyreserved: 5.2 charat: 3 bonetyavailable: 3 bonetpricelist: 3.2 paymenttermsoadate: 4.1 paymentavailable: 4.2 currencybase: 3 currencyround: 3 paymenttermsoadtype: 5.2 dba_displaytype: 6 invocopen: 6.4 productattribute: 5.2 bonetpricelist: 3.2 paymentallocated: 4.2 paymenttermsoadcount: 4.9 hparterrentallocation: 3.2 compierversion: 3 bonetyband: 5.2 getdate: 3 dba_constraintm: 4 incoespad: 3 bonetpricestd: 3.2

Looking at the Details

- Shows a break down by object type and relative complexity of each function

Object	Number	Invalid	Estimated cost	Comments	Details
FUNCTION	36	0	175.4	Total size of function code: 60754 bytes.	bomqtyreserved: 5.2 charat: 3 bomqtyavailable: 3 bompricelimit: 3.2 paymenttermduedate: 4.1 paymentavailable: 4.2 currencybase: 3 currencyround: 3 paymenttermduedays: 5.2 dba_displaytype: 6 invoiceopen: 6.4 productattribute: 5.2 bompricelist: 3.2 paymentallocated: 4.2 paymenttermdiscount: 4.9 bpartnerremitlocation: 3.2 compiereversion: 3 bomqtyonhand: 5.2 getdate: 3 dba_constraintcmd: 4

Looking at the Details

- The table analysis shows the row count of tables
- Also finds BLOBs
- Gives an indicator of the relative complexity of a data migration

Object	Number	Invalid	Estimated cost	Comments	Details
TABLE	528	0	173.2	965 check constraint(s).	9 binary columns 2 reserved words in column name Total number of rows: 38768 Top 10 of tables sorted by number of rows: ad_column has 11839 rows ad_field has 10683 rows ad_element has 2155 rows c_periodcontrol has 1800 rows ad_printformatitem has 1283 rows ad_sequence has 810 rows ad_message has 762 rows ad_ref_list has 709 rows ad_tab has 624 rows ad_table has 591 rows

Looking at the Summary

- The total is for all database objects
- The estimated effort uses the cost unit value
 - Assumes a 7 hour work day
- The migration level attempts to determine the complexity of a migration

Total	1378	39	908.60	908.60 cost migration units means approximately 44 man-day(s). The migration unit was set to 20 minute(s)
--------------	------	----	--------	---

Migration level: B-5

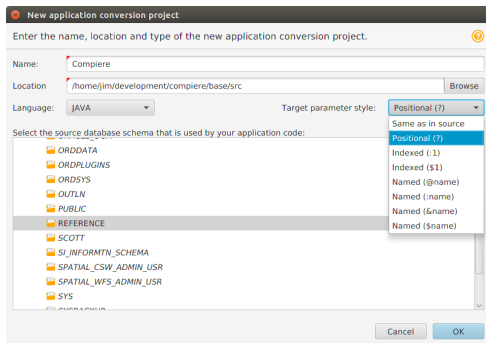
- Migration levels:
 - A - Migration that might be run automatically
 - B - Migration with code rewrite and a human-days cost up to 30 days
 - C - Migration with code rewrite and a human-days cost above 30 days
- Technical levels:
 - 1 = trivial: no stored functions and no triggers
 - 2 = easy: no stored functions but with triggers, no manual rewriting
 - 3 = simple: stored functions and/or triggers, no manual rewriting
 - 4 = manual: no stored functions but with triggers or views with code rewriting
 - 5 = difficult: stored functions and/or triggers with code rewriting

Scanning the Application

- AWS Schema Conversion Tool (SCT)
 - Java
 - C/C++
 - C#
- Ora2PG
 - Does not attempt to scan the application code

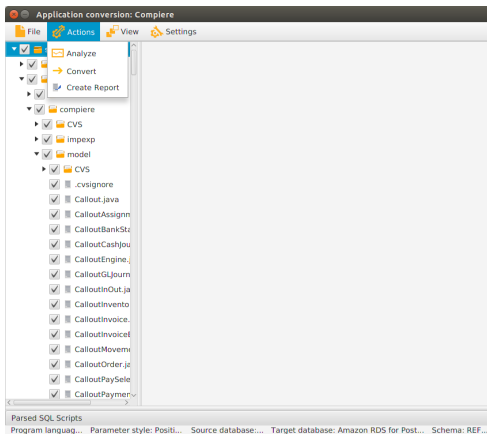
Using SCT for Applications

- Add an application to an existing project
- Needs a database connection to the corresponding schema



Creating an Application Report

- A report can be generated for all or part of the source code tree



Looking at the Summary

- Identifies the relative scope of changes required
- Tries to classify the required changes by difficulty

Executive Summary

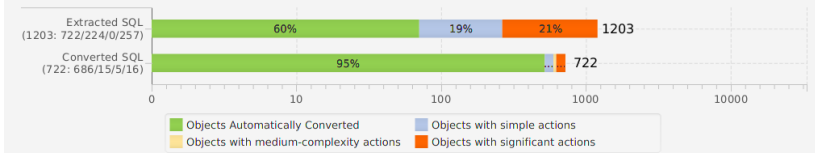
We completed the analysis of your application "Compiere". We were able to extract 1,203 SQL statements from your application code, which need to be converted to Amazon RDS for PostgreSQL.

Of the total 1,203 SQL Statements in the source code, we were able to identify 946 (79%) SQL statements that can be extracted automatically. 257 (21%) SQL statements require 21 medium and 0 significant user action(s) to complete the extraction.

Based on our analysis of the SQL syntax in your code, we estimate that 97% of your statements can be converted to Amazon RDS for PostgreSQL automatically. To complete the migration, we recommend 670 conversion action(s) ranging from simple tasks to medium-complexity actions to significant conversion actions.

Of the total 722 SQL Statements in the source code, we were able to identify 701 (97%) SQL statements that can be converted to Amazon RDS for PostgreSQL automatically or with minimal changes.

21 (3%) SQL statements require 10 medium and 36 significant user action(s) to complete the conversion.



Common Actions

- SCT has difficulty determining the completeness of SQL that is constructed with string concatenation
- Use it as an indicator of where to look in the code

The screenshot shows an IDE window with a 'Summary' tab. It displays an error message: 'Issue: 100002: Statement has been truncated'. Below the error, a list of statements is shown, each with a file path and line numbers. The first statement is highlighted in blue. Below the error list, the source code for 'FactAcct.java' is visible. The code shows a SQL statement being constructed using string concatenation, which is the cause of the truncation error.

```
Summary | SQL Extraction Actions | SQL Conversion Actions
* Issue: 100002: Statement has been truncated
Recommended action: Perform a manual extraction of SQL code and try to convert it from inside SCT studio
Number of occurrences: 71
▶ Statement: /home/jim/development/compiere/base/src/org/compiere/model/MFactAcct.java Line 40 13:50 (Number of occurrences: 1)
▶ Statement: /home/jim/development/compiere/base/src/org/compiere/model/MinOut.java Line 751 16:140 (Number of occurrences: 1)
▶ Statement: /home/jim/development/compiere/base/src/org/compiere/model/MinOut.java Line 1400 15:34 (Number of occurrences: 1)
▶ Statement: /home/jim/development/compiere/base/src/org/compiere/model/Minventory.java Line 535 15:34 (Number of occurrences: 1)
▶ Statement: /home/jim/development/compiere/base/src/org/compiere/model/Minvoice.java Line 1399 16:90 (Number of occurrences: 1)
▶ Statement: /home/jim/development/compiere/base/src/org/compiere/model/MMovement.java Line 532 15:34 (Number of occurrences: 1)
▶ Statement: /home/jim/development/compiere/base/src/org/compiere/model/MOrder.java Line 1316 16:86 (Number of occurrences: 1)
▶ Statement: /home/jim/development/compiere/base/src/org/compiere/model/MProjectType.java Line 151 38:206 (Number of occurrences: 1)
▶ Statement: /home/jim/development/compiere/base/src/org/compiere/model/MStorage.java Line 42 16:94 (Number of occurrences: 1)

Source file: /home/jim/development/compiere/base/src/org/compiere/model/MFactAcct.java
040 sb.append("DELETE Fact_Acct WHERE AD_Table_ID=").append(AD_Table_ID)
041 .append(" AND Record_ID=").append(Record_ID);
042 int no = DB.executeUpdate(sb.toString(), trxName);
043 if (no == -1)
044     s_log.log(Level.SEVERE, "failed: AD_Table_ID=" + AD_Table_ID + ", Record_ID" +
045         Record_ID);
046 else
047     s_log.fine("delete - AD_Table_ID=" + AD_Table_ID
048         + ", Record_ID=" + Record_ID + " - #" + no);
049 return no;
050 // delete
```

Common Actions

- The more dynamic the SQL construction is coded, the less likely SCT will evaluate it correctly

▼  **Issue: 101001: General syntax error. We failed to parse the statement**

Recommended action: Recommended Action: Perform a manual extraction of SQL code and convert it from inside using SCT studio.
Number of occurrences: 251


- ▶ Statement: `/home/jim/development/compiere/base/src/org/compiere/impexp/ImpFormat.java Line 490 40:63` (Number of occurrences)
- ▶ Statement: `/home/jim/development/compiere/base/src/org/compiere/impexp/ImpFormat.java Line 559 27:41` (Number of occurrences)
- ▶ Statement: `/home/jim/development/compiere/base/src/org/compiere/impexp/ImpFormat.java Line 570 27:75` (Number of occurrences)

Source file: /ho...mpFormat.java

```
488
489
490 //      Check if the record is already there -----
491 StringBuffer sql = new StringBuffer ("SELECT COUNT(*), MAX(")
492 .append(m_tablePK).append(") FROM ").append(m_tableName)
      .append(" WHERE AD_Client_ID=").append(AD_Client_ID).append(" AND (");
```


False Positives

- SCT searches for keywords in the code so frequently it picks up things that are not SQL
- Use the output as a guide and planning tool

▼  **Issue: 100003: DDL statement is not supported. Statement has been skipped**

Recommended action: Rewrite the DDL statement

Number of occurrences: 7

▶ Statement: `/home/jjm/development/compiere/base/src/org/compiere/model/MInOutConfirm.java Line 49 17:48` (Number of occurrence)

▶ Statement: `/home/jjm/development/compiere/base/src/org/compiere/model/PaymentProcessor.java Line 61 14:33` (Number of occurrence)

▶ Statement: `/home/jjm/development/compiere/base/src/org/compiere/model/ProductCost.java Line 329 30:36` (Number of occurrence)

Source file: /ho...tConfirm.java

```
041         if (checkExisting)
042         {
043             MInOutConfirm[] confirmations = ship.getConfirmations(false);
044             for (int i = 0; i < confirmations.length; i++)
045             {
046                 MInOutConfirm confirm = confirmations[i];
047                 if (confirm.getConfirmType().equals(confirmType))
048                 {
049                     s_log.info("create - existing: " + confirm);
050                     return confirm;
051                 }
052             }
053         }
054     }
```

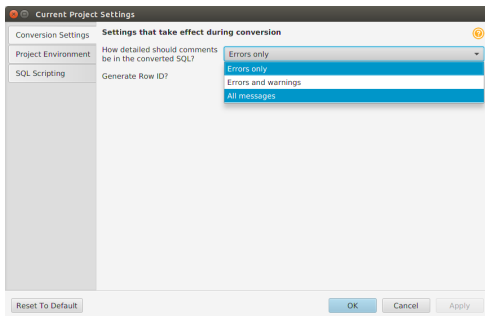
Consolidating the Input

- Use the available tools to determine the relative scope
- SCT can be used to evaluate the complexity of the required changes and the amount of application changes
- Ora2pg can be used to evaluate the amount of database changes required
- Good old fashion gut instinct can be the fastest analysis tool

Conversion Using SCT

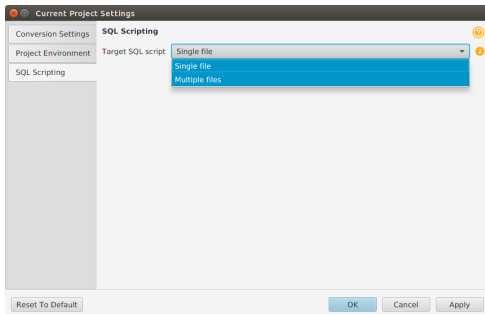
Configuring SCT

- Before a conversion, some changes to the default settings will improve the migration project
- Adding all comments to the generated PLpgSQL will help the developers interpreting the generated output



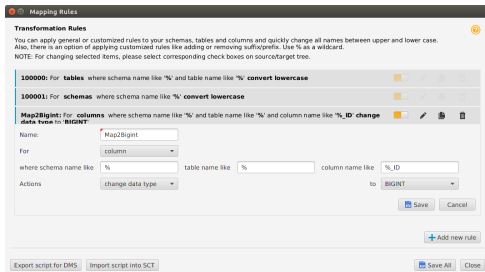
Configuring SCT

- Creating multiple SQL files will make the files easier to manage for large schemas
- Splitting the SQL speeds up the data loads
 - Create indexes and constraints after the data load



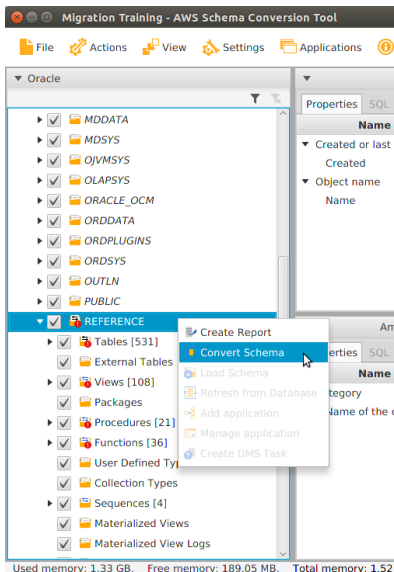
Mapping Rules

- The default rules convert the object names to lower case
- If possible, a mapping rule for changing data types will save a lot of time
 - This is functionally not necessary but makes a huge performance difference on the converted application



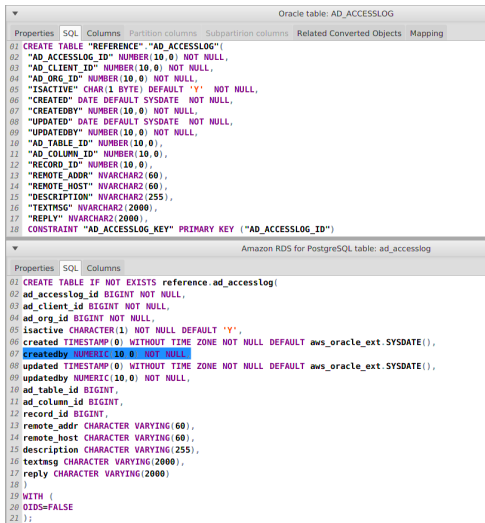
Converting a Schema

- Once configured, converting a schema will bring all database objects into the SCT project
- This step may take a while depending on the size and complexity of the schema



Check the Results

- Visually inspect the resulting tables
- Look for patterns that can be fixed using a Mapping Rule
- Continue to iterate until the obvious data types can be automatically mapped



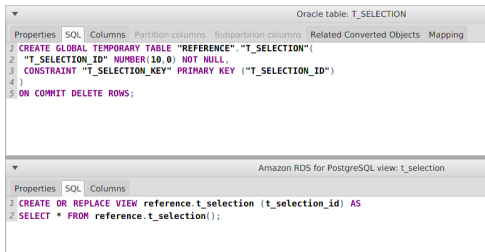
The image shows two screenshots of SQL queries. The top screenshot is for an Oracle table named 'AD_ACCESSLOG'. The bottom screenshot is for an Amazon RDS PostgreSQL table named 'ad_accesslog'. Both queries define columns with various data types and constraints.

```
Oracle table: AD_ACCESSLOG
Properties | SQL | Columns | Partition columns | Subpartition columns | Related Converted Objects | Mapping
01 CREATE TABLE "REFERENCE"."AD_ACCESSLOG" (
02 "AD_ACCESSLOG_ID" NUMBER(10,0) NOT NULL,
03 "AD_CLIENT_ID" NUMBER(10,0) NOT NULL,
04 "AD_ORG_ID" NUMBER(10,0) NOT NULL,
05 "ISACTIVE" CHAR(1 BYTE) DEFAULT 'Y' NOT NULL,
06 "CREATED" DATE DEFAULT SYSDATE NOT NULL,
07 "CREATEDBY" NUMBER(10,0) NOT NULL,
08 "UPDATED" DATE DEFAULT SYSDATE NOT NULL,
09 "UPDATEDBY" NUMBER(10,0) NOT NULL,
10 "AD_TABLE_ID" NUMBER(10,0),
11 "AD_COLUMN_ID" NUMBER(10,0),
12 "RECORD_ID" NUMBER(10,0),
13 "REMOTE_ADDR" NVARCHAR2(60),
14 "REMOTE_HOST" NVARCHAR2(60),
15 "DESCRIPTION" NVARCHAR2(255),
16 "TEXTMSG" NVARCHAR2(2000),
17 "REPLY" NVARCHAR2(2000),
18 CONSTRAINT "AD_ACCESSLOG_KEY" PRIMARY KEY ("AD_ACCESSLOG_ID")

Amazon RDS for PostgreSQL table: ad_accesslog
Properties | SQL | Columns
01 CREATE TABLE IF NOT EXISTS reference.ad_accesslog(
02 ad_accesslog_id BIGINT NOT NULL,
03 ad_client_id BIGINT NOT NULL,
04 ad_org_id BIGINT NOT NULL,
05 isactive CHARACTER(1) NOT NULL DEFAULT 'Y',
06 created TIMESTAMPTZ(0) WITHOUT TIME ZONE NOT NULL DEFAULT aws_oracle_ext.SYSDATE(),
07 createdby NUMERIC(10,0) NOT NULL
08 updated TIMESTAMPTZ(0) WITHOUT TIME ZONE NOT NULL DEFAULT aws_oracle_ext.SYSDATE(),
09 updatedby NUMERIC(10,0) NOT NULL,
10 ad_table_id BIGINT,
11 ad_column_id BIGINT,
12 record_id BIGINT,
13 remote_addr CHARACTER VARYING(60),
14 remote_host CHARACTER VARYING(60),
15 description CHARACTER VARYING(255),
16 textmsg CHARACTER VARYING(2000),
17 reply CHARACTER VARYING(2000)
18 )
19 WITH (
20 OIDS=FALSE
21 );
```


Check the Results

- Visually inspect the resulting views
- Note any conversions of Global Temporary Tables
- These will need to be manually converted to unlogged tables



The screenshot displays two SQL query windows. The top window, titled "Oracle table: T_SELECTION", shows the following SQL code:

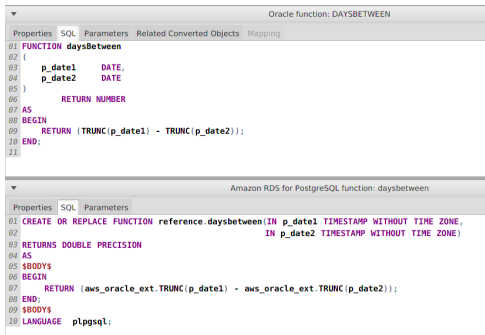
```
1 CREATE GLOBAL TEMPORARY TABLE "REFERENCE"."T_SELECTION"(  
2 "T_SELECTION_ID" NUMBER(10,0) NOT NULL,  
3 CONSTRAINT "T_SELECTION_KEY" PRIMARY KEY ("T_SELECTION_ID")  
4 )  
5 ON COMMIT DELETE ROWS;
```

The bottom window, titled "Amazon RDS for PostgreSQL view: t_selection", shows the following SQL code:

```
1 CREATE OR REPLACE VIEW reference.t_selection (t_selection_id) AS  
2 SELECT * FROM reference.t_selection();
```

Check the Results

- Visually inspect the resulting functions
- Initially inspect the parameters that the match the data types of the tables
- Most functions will need at least minor manual modification



The screenshot displays two SQL function definitions side-by-side. The top window, titled "Oracle function: DAYS BETWEEN", shows the following code:

```
01 FUNCTION daysBetween
02 (
03   p_date1   DATE,
04   p_date2   DATE
05 )
06   RETURN NUMBER
07 AS
08 BEGIN
09   RETURN (TRUNC(p_date1) - TRUNC(p_date2));
10 END;
11
```

The bottom window, titled "Amazon RDS for PostgreSQL function: daysbetween", shows the following code:

```
01 CREATE OR REPLACE FUNCTION reference.daysbetween(IN p_date1 TIMESTAMP WITHOUT TIME ZONE,
02                                                    IN p_date2 TIMESTAMP WITHOUT TIME ZONE)
03 RETURNS DOUBLE PRECISION
04 AS
05 $BODY$
06 BEGIN
07   RETURN (aws_oracle_ext.TRUNC(p_date1) - aws_oracle_ext.TRUNC(p_date2));
08 END;
09 $BODY$
10 LANGUAGE plpgsql;
```

Check the Results

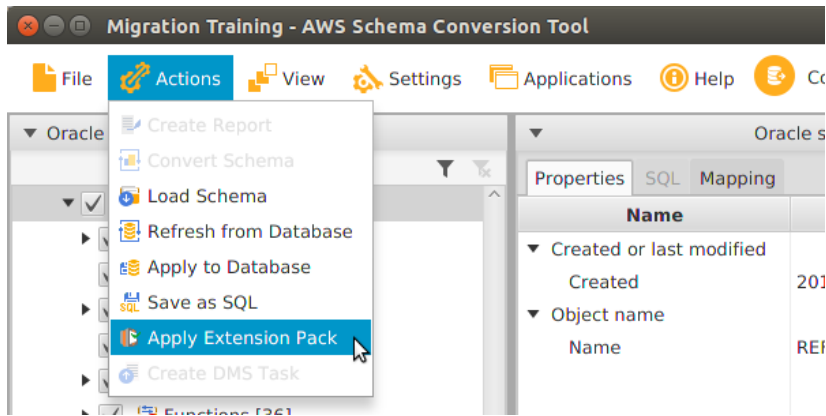
- Visually inspect the resulting packages
- Initially inspect the package content names
- SCT prefixes `PACKAGE_NAME$` before each function name
 - This results in many manual changes in the application code

```
Oracle function: IS_DATE
Properties SQL Parameters Related Converted Objects Mapping
01 function is_date(
02   p_str in varchar2,
03   p_date_format in varchar2)
04   return boolean
05   deterministic
06 as
07   l_date date;
08 begin
09   l_date := to_date(p_str, p_date_format);
10   return true;
11 exception
12   when others then -- Using a when others since date format could also be invalid
13     return false;
14 end is_date

Amazon RDS for PostgreSQL function: oos_util_validation$is_date
Properties SQL Parameters
01 CREATE OR REPLACE FUNCTION reference_oos_util_validation$is_date(IN p_str TEXT, IN p_date_format TEXT)
02 RETURNS BOOLEAN
03 AS
04 $BODY$
05 DECLARE
06   l_date TIMESTAMPTZ(0) WITHOUT TIME ZONE;
07 BEGIN
08   l_date := aws_oracle_ext.TO_DATE(p_str, p_date_format);
09   RETURN TRUE;
10 EXCEPTION
11   WHEN others THEN
12     RETURN FALSE;
13 END;
14 $BODY$
15 LANGUAGE plpgsql;
```

Extension Packs

- Provides an extension pack to ease a conversion
- It contains many Oracle functions and procedures commonly used in applications



Extension Packs

- dbms_job
 - dbms_random
 - utl_smtp
 - get_package_variable
 - add_months
 - instr
 - sysdate
 - to_date
 - to_char
- These functions can help during the initial development phase to quickly create interdependent functions
 - All references to these functions should be removed for the final production release
 - Many of these functions are slow and troublesome

Wrapper Functions

- The intent is good for using the extension pack
- Allows for easier unit testing especially with sysdate
 - Overloading allows for setting the value of a date

```
CREATE OR REPLACE FUNCTION aws_oracle_ext."sysdate" ()
  RETURNS timestamp without time zone AS
$BODY$
DECLARE
  l_var1 interval ;
BEGIN
  l_var1 := '0 hour'; /* Please type your value instead of 0 */
  return (clock_timestamp()::TIMESTAMP(0) WITHOUT TIME ZONE) + l_var1;
END;
$BODY$
LANGUAGE plpgsql VOLATILE;
```

Wrapper Functions

- It is much more efficient to create sysdate as a STABLE SQL function

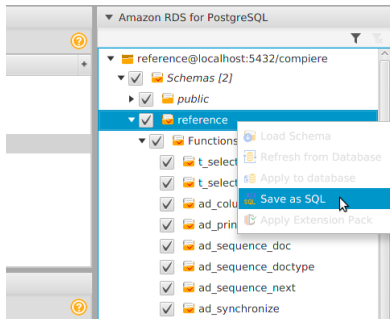
```
CREATE OR REPLACE FUNCTION training.sysdate()  
  RETURNS timestamp AS  
$$  
  SELECT CURRENT_TIMESTAMP::timestamp(0);  
$$ LANGUAGE sql STABLE;
```

```
SELECT count(aws_oracle_ext.sysdate())  
  FROM generate_series(1, 1000000);  
  count  
-----  
1000000  
(1 row)  
  
Time: 2787.506 ms (00:02.788)
```

```
SELECT count(training.sysdate())  
  FROM generate_series(1, 1000000);  
  count  
-----  
1000000  
(1 row)  
  
Time: 334.603 ms
```


















Export the Schema

- For quick tests, applying the schema directly to the database is fine
- For the actual conversion, exporting the schema as files allows for
 - Modification of functions and views noted during manual inspection
 - Delaying creating indexes and constraints until the data is loaded



Export the Schema

- SCT will create a file for each type of object
- For many environments, it is more convenient to break each object into its own file and tracked in source control

 0. drop-function.sql	6.0 kB
 1. drop-foreign-key-constraint.sql	47.6 kB
 2. drop-constraint.sql	106.0 kB
 3. drop-index.sql	8.3 kB
 4. drop-view.sql	5.8 kB
 5. drop-table.sql	26.2 kB
 6. drop-sequence.sql	270 bytes
 7. drop-database.sql	64 bytes
 8. create-database.sql	107 bytes
 9. create-sequence.sql	496 bytes
 10. create-table.sql	420.7 kB
 11. create-view.sql	202.2 kB
 12. create-index.sql	16.9 kB
 13. create-constraint.sql	154.5 kB
 14. create-foreign-key-constraint.sql	325.2 kB
 15. create-function.sql	274.7 kB
 16. create-trigger.sql	381 bytes

Fixing Issues

- Several of the files will require manual intervention
- Look for the word Severity

```
IF (v_Result = 1) THEN
  /*
  [5334 - Severity CRITICAL - Unable convert statements ...
  EXECUTE IMMEDIATE v_Cmd
  */
  v_Message := CONCAT_WS(' ', '@Created@ - ', v_Cmd);
END IF;
EXCEPTION
  WHEN others THEN
```

Fixing Issues

- Convert data types in functions to match the pattern in the tables

```
CREATE OR REPLACE FUNCTION
  reference.ad_column_sync(IN p_pinstance_id DOUBLE PRECISION)
RETURNS void
AS
$BODY$
/* Logistice */
DECLARE
  v_ResultStr CHARACTER VARYING(2000);
  v_Message CHARACTER VARYING(2000);
  v_Result DOUBLE PRECISION := 1
  /* 0=failure */;
  v_Record_ID DOUBLE PRECISION;
  v_AD_User_ID DOUBLE PRECISION
```

Loading the Schema

- Once the objects have been manually adjusted, load the following object types into PostgreSQL
 - Databases
 - Schemas
 - Tables
 - Views
 - Functions
 - Sequences

- Other object types will be loaded after the data load

Conversion Using Ora2PG

Initializing Ora2PG

```
ora2pg -b training --init_project training
```

```
Creating project training.
```

```
./training/
```

```
  schema/
```

```
    dblinks/
```

```
    directories/
```

```
    functions/
```

```
    grants/
```

```
    mviews/
```

```
    packages/
```

```
    partitions/
```

```
    procedures/
```

```
    sequences/
```

```
    synonyms/
```

```
    tables/
```

```
    tablespaces/
```

```
    triggers/
```

```
    types/
```

```
    views/
```

```
sources/
```

```
  functions/
```

```
  mviews/
```

```
  packages/
```

```
  partitions/
```

```
  procedures/
```

```
  triggers/
```

```
  types/
```

```
  views/
```

```
data/
```

```
config/
```

```
reports/
```

```
Generating generic configuration file
```

```
Creating script export_schema.sh ...
```

```
Creating script import_all.sh ...
```

Configuring Ora2PG

- Modify config/ora2pg.conf to customize the settings for a specific environment
- Connection information should be the same as the analysis phase

```
# Set the Oracle home directory
ORACLE_HOME      /usr/lib/oracle/12.1/client64

# Set Oracle database connection (datasource, user, password)
ORACLE_DSN       dbi:Oracle:host=mydb.mydom.fr;sid=SIDNAME;port=1521
ORACLE_USER      system
ORACLE_PWD       manager
```

Configuring Ora2PG

- Configure the schema section so a schema is created and exported instead of using the public schema

```
#-----  
# SCHEMA SECTION (Oracle schema to export and use of schema in PostgreSQL)  
#-----  
  
# Export Oracle schema to PostgreSQL schema  
EXPORT_SCHEMA    1  
  
# Oracle schema/owner to use  
SCHEMA    CHANGE_THIS_SCHEMA_NAME  
  
# Enable/disable the CREATE SCHEMA SQL order at starting of the output file.  
# It is enable by default and concern on TABLE export type.  
CREATE_SCHEMA    1
```


Configuring Ora2PG

- Sometime names of objects matter

```
# By default, primary key names in the source database are ignored, and  
# default key names are created in the target database. If this is set  
# to true, primary key names are kept.
```

```
KEEP_PKEY_NAMES 0
```

```
# By default all object names are converted to lower case, if you  
# want to preserve Oracle object name as-is set this to 1. Not recommended  
# unless you always quote all tables and columns on all your scripts.
```

```
PRESERVE_CASE 0
```

Configuring Ora2PG

- Sometime names of objects matter

```
# Enable this directive to rename all indexes using tablename_columns_names.  
# Could be very useful for database that have multiple time the same index  
# name or that use the same name than a table, which is not allowed  
# Disabled by default.  
INDEXES_RENAMING          0
```

```
# Enable this directive if you have tables or column names that are a reserved  
# word for PostgreSQL. Ora2Pg will double quote the name of the object.  
USE_RESERVED_WORDS       1
```

Configuring Ora2PG

- Control the output to match the development and source control process

```
FILE_PER_CONSTRAINT      1
FILE_PER_INDEX           1
FILE_PER_FKEYS           1
FILE_PER_TABLE           1
```

Configuring Ora2PG

- Map NUMBER correctly

```
# If set to 1 replace portable numeric type into PostgreSQL internal
# type. If you have monetary fields or don't want rounding issues with
# the extra decimals you should preserve the same numeric(p,s).
PG_NUMERIC_TYPE 1
```

```
# If NUMBER without precision are set to DEFAULT_NUMERIC (see bellow).
PG_INTEGER_TYPE 1
```

```
# NUMBER() without precision are converted by default to bigint only if
# PG_INTEGER_TYPE is true. You can overwrite this value to any PG type,
# like integer or float.
DEFAULT_NUMERIC bigint
```

Configuring Ora2PG

- Convert Oracle syntax

```
# Enable this configuration directive to allow export of all PL/SQL code
# even if it is marked as invalid. The 'VALID' or 'INVALID' status
# applies to functions, procedures, packages and user defined types.
EXPORT_INVALID 0
```

```
# Enable PLSQL to PLPSQL conversion. Default enabled.
PLSQL_PGSQL 1
```

```
# Ora2Pg can replace all conditions with a test on NULL by a call to
# the coalesce() function to mimic the Oracle behavior where empty
# field are considered equal to NULL.
NULL_EQUAL_EMPTY 1
```

```
# If you don't want to export package as schema but as simple functions
# you might also want to replace all call to package_name.function_name.
PACKAGE_AS_SCHEMA 1
```

Handling Errors

- Turning off stopping on errors will allow the script to run to completion
- Allows for handling of errors in bulk

```
# Set it to 0 to not include the call to \set ON_ERROR_STOP ON in all SQL
# scripts. By default this order is always present.
STOP_ON_ERROR          0
```

Converting a Schema

```
./export_schema.sh
[=====>] 529/529 tables (100.0%) end of scanning.
[=====>] 13/13 objects types (100.0%) end of objects auditing.
[=====>] 529/529 tables (100.0%) end of scanning.
[=====>] 529/529 tables (100.0%) end of table export.
[=====>] 1/1 packages (100.0%) end of output.
[=====>] 108/108 views (100.0%) end of output.
[=====>] 4/4 sequences (100.0%) end of output.
[=====>] 0/0 triggers (100.0%) end of output.
[=====>] 35/35 functions (100.0%) end of functions export.
[=====>] 21/21 procedures (100.0%) end of procedures export.
...
[=====>] 1/1 packages (100.0%) end of output.
[=====>] 108/108 views (100.0%) end of output.
[=====>] 0/0 triggers (100.0%) end of output.
[=====>] 35/35 functions (100.0%) end of functions export.
[=====>] 21/21 procedures (100.0%) end of procedures export.
```

Check the Results

- Visually inspect the resulting tables
./schema/tables/tables.sql
- Look for column defaults that should be modified

...

```
created timestamp NOT NULL DEFAULT LOCALTIMESTAMP,  
createdby bigint NOT NULL,  
updated timestamp NOT NULL DEFAULT LOCALTIMESTAMP,  
updatedby bigint NOT NULL,
```

...

Wrapper Functions

- It is efficient to create sysdate as a STABLE SQL function
- Allows for control of the result in the testing phase

```
CREATE OR REPLACE FUNCTION training.sysdate()  
  RETURNS timestamp AS  
$$  
  SELECT CURRENT_TIMESTAMP::timestamp(0);  
$$ LANGUAGE sql STABLE;
```

Loading the Schema

```
./import_all.sh -?  
usage: import_all.sh [options]
```

Script used to load exported sql files into PostgreSQL in practical manner allowing you to chain and automatically import schema and data.

options:

```
-a                import data only  
-b filename      SQL script to execute just after table creation to fix  
-f              force no check of user and database existing and do not  
-i              only load indexes, constraints and triggers  
-I              do not try to load indexes, constraints and triggers  
-j cores        number of connection to use to import data or indexes  
-n schema       comma separated list of schema to create  
-P cores        number of tables to process at same time for data import  
-s              import schema only, do not try to import data  
-t export       comma separated list of export type to import  
-x              import indexes and constraints after data  
-y              reply Yes to all questions for automatic import
```

Loading the Schema

- Try importing the schema that was automatically converted

```
./import_all.sh -y -s -h localhost -d training -U jim -o jim > out
```

- There WILL be errors

```
psql:./schema/packages/oos_util_validation/is_number_package.sql:44:  
ERROR: unrecognized exception condition "value_error"  
CONTEXT: compilation of PL/pgSQL function "is_number" near line 9  
psql:./schema/packages/oos_util_validation/is_date_package.sql:5:  
ERROR: current transaction is aborted, commands ignored until end  
of transaction block
```

Loading the Schema

- Fix all syntax errors
- Many issues will be very simple to rectify with looking at the error in context of the wider code base

```
diff packages/oos_util_validation/is_number_package.sql \
    ../orig/packages/oos_util_validation/is_number_package.sql
37c37
<     when value_error then
- - -
>     when others then
```

Loading the Schema

- Skip missing dependencies during the first pass
- Note: This will cause all subsequent objects of a given type to fail

```
psql:./schema/views/M_INOUT_LINE_VT_view.sql:55:  
ERROR: function productattribute(bigint) does not exist  
LINE 10: COALESCE(COALESCE(pt.Name,p.Name)||productAttribute(iol.M_A...  
                                     ^  
HINT: No function matches the given name and argument types. You might  
need to add explicit type casts.  
psql:./schema/views/RV_CLICK_MONTH_view.sql:5:  
ERROR: current transaction is aborted, commands ignored until end  
of transaction block  
psql:./schema/views/RV_CLICK_MONTH_view.sql:15:  
ERROR: current transaction is aborted, commands ignored until end  
of transaction block
```

Loading the Schema

- Usually, views are the most sensitive to errors
- Loading them last will allow all dependencies to be created first

```
./import_all.sh -y -s \  
-t TYPE, TABLE, PACKAGE, SEQUENCE, FUNCTION,  
  PROCEDURE, PARTITION, VIEW, MVIEW  
-h localhost -d training -U jim -o jim
```

Testing

Testing

- Unit Testing
- Data Validation
- An full QA cycle

Unit Testing

- A unit test framework is needed
- Commonly already available for the application code
- Can be simply SQL script fed to SQL*Plus and psql
- Access to the source database is only needed to generate expected results

Unit Testing

- Create at least 1 test case for each database object
- There should be enough cases for have full code coverage across each object
- All permutations of parameters should be tested

Unit Testing

- Start with a simple function

```
create or replace FUNCTION charAt
(
  p_string  VARCHAR2,
  p_pos     NUMBER
)
RETURN VARCHAR2
AS
BEGIN
  RETURN SUBSTR(p_string, p_pos, 1);
END;
```

Unit Testing

- At a minimum, a positive test case should be created

```
SQL> SELECT charAt('PostgreSQL', 4) AS a FROM dual;
```

```
A
```

```
-----
```

```
t
```

Unit Testing

- Each parameter should be tested for NULL

```
SQL> SELECT charAt(null, 4) AS a FROM dual;
```

```
A
```

```
SQL> SELECT charAt('PostgreSQL', null) AS a FROM dual;
```

```
A
```

Unit Testing

- Out of bounds conditions should be tested

```
SQL> SELECT charAt('PostgreSQL', -1) AS a FROM dual;
```

```
A
```

```
-----
```

```
L
```

```
SQL> SELECT charAt('P', 4) AS a FROM dual;
```

```
A
```

```
-----
```

Unit Testing

- Dates are parameters should be carefully considered

```
create or replace FUNCTION addDays
(
  p_date      DATE,
  p_days      NUMBER
)
RETURN DATE
AS
BEGIN
  RETURN TRUNC(p_date) + p_days;
END;
```

Unit Testing

- Do not create test cases that change over time

```
SQL> SELECT addDays(SYSDATE, 7) AS a FROM dual;
```

```
A
```

```
-----
```

```
02-APR-18
```


Unit Testing

- The results of the test case should be immutable

```
SQL> SELECT addDays(to_date('2000-01-01', 'YYYY-MM-DD'), 7) AS a  
2 FROM dual;
```

```
A
```

```
-----
```

```
08-JAN-00
```

Unit Testing

- Multiple code paths need to be tested

```
create or replace FUNCTION acctBalance
(p_Account_ID IN NUMBER, p_AmtDr IN NUMBER, p_AmtCr IN NUMBER)
RETURN NUMBER
AS
...
BEGIN
    v_balance := p_AmtDr - p_AmtCr;
    IF (p_Account_ID > 0) THEN
        SELECT AccountType, AccountSign
           INTO v_AccountType, v_AccountSign
          FROM C_ElementValue
         WHERE C_ElementValue_ID=p_Account_ID;
        -- Natural Account Sign
        IF (v_AccountSign='N') THEN
            IF (v_AccountType IN ('A','E')) THEN
                v_AccountSign := 'D';
            ELSE
                ...
            END IF;
        END IF;
    END IF;
END;
```

Unit Testing

```
SQL> SELECT acctBalance(587,11,22) AS a FROM dual;
```

```
A
```

```
-----
```

```
11
```

```
SQL> SELECT acctBalance(590,11,22) AS a FROM dual;
```

```
A
```

```
-----
```

```
11
```

```
SQL> SELECT acctBalance(471,11,22) AS a FROM dual;
```

```
A
```

```
-----
```

```
-11
```

```
SQL> SELECT acctBalance(-1,11,22) AS a FROM dual;
```

```
A
```

```
-----
```

```
-11
```

Unit Testing

- Running the test cases against PostgreSQL the first time usually produces many errors
- Remember: PostgreSQL does not fully “compile” the function until it is first run

```
> SELECT charAt('PostgreSQL', 4) AS a;
ERROR: function substr(text, bigint, integer) does not exist
LINE 1: SELECT SUBSTR(p_string, p_pos, 1)
           ^
HINT: No function matches the given name and argument types. You
might need to add explicit type casts.
QUERY: SELECT SUBSTR(p_string, p_pos, 1)
CONTEXT: PL/pgSQL function charat(text,bigint) line 3 at RETURN
```

Unit Testing

- Fix the errors to return the expected results
- Hint: Use this as an opportunity to make simple improvements

```
diff CHARAT_function.sql ../orig/CHARAT_function.sql
11c11
<     RETURN SUBSTR(p_string, p_pos::int, 1);
...
>     RETURN SUBSTR(p_string, p_pos, 1);
17c17
< IMMUTABLE;
...
> STABLE;
```

Unit Testing

- Analyze the results to determine the correct course of action for different result
 - Fix the PostgreSQL function?
 - Fix the expected results?

```
SQL> SELECT charAt('PostgreSQL', -1) AS a FROM dual;
```

```
A
```

```
-----
```

```
L
```

```
> SELECT charAt('PostgreSQL', -1) AS a;
```

```
a
```

```
---
```

Unit Testing

- Outputted date styles will be different

```
SQL> SELECT addDays(to_date('2000-01-01', 'YYYY-MM-DD'), 7) AS a
      2 FROM dual;
```

```
A
-----
08-JAN-00
```

```
> SELECT addDays(to_date('2000-01-01', 'YYYY-MM-DD'), 7) AS a;
```

```

a
-----
2000-01-08 00:00:00
```

Unit Testing

- Change the test case to create consistent results for both databases

```
SQL> SELECT to_char(addDays(to_date('2000-01-01', 'YYYY-MM-DD'), 7), 'YYYY-MM-DD')  
2 FROM dual;
```

```
A  
-----  
2000-01-08
```

```
> SELECT to_char(addDays(to_date('2000-01-01', 'YYYY-MM-DD'), 7), 'YYYY-MM-DD')  
a
```

```
-----  
2000-01-08
```


Data Validation

- Determine the amount of validation required for each development stage
 - Development
 - QA
 - Staging
 - Production
- Develop a plan and strategy for each stage

Data Validation

- Spot checking by getting the counts

```
SQL> SELECT count (*) FROM AD_ELEMENT;
```

```
COUNT (*)
```

```
-----  
2155
```

```
> SELECT count (*) FROM AD_ELEMENT;
```

```
count
```

```
-----  
2155
```

Data Validation

- Check some data by using aggregates

```
SQL> SELECT sum(ad_element_id), sum(length(name)) FROM AD_ELEMENT;
```

```
SUM(AD_ELEMENT_ID)  SUM(LENGTH(NAME))
```

```
-----
```

```
3702371           29254
```

```
> SELECT sum(ad_element_id), sum(length(name)) FROM AD_ELEMENT;
```

```
sum | sum
```

```
-----+-----
```

```
3702371 | 29254
```

Data Validation

- To be entirely sure all data is migrated accurately, checksums must be calculated for all rows and all columns
- There are several open source and commercial tools available

Migrating Data

Migrating Data

- Files
- Foreign Data Wrapper
- Replication

File Export

- Pros
 - Ideal for development and testing environments
 - Repeatable with a constant data set
 - No direct access to the source database is necessary

- Cons
 - Moving the data twice
 - Requires the source database to be quiesced

File Export

- Export the data out of the source system as a file
 - Scripts
 - UI like SQL Developer
 - ETL Tools
- The best format is usually CSV but may be different based on the data
- Load the data using the COPY command

Copy

- COPY is an SQL command, so FROM/TO are with respect to the server
 - Most other databases have a “load utility” which pushes data
- COPY FROM
 - Loads data into the database (pull-in)
- COPY TO
 - Exports data from the database

Copy (cont.)

```
COPY table_name [ ( column_name [, ...] ) ]  
  FROM { 'filename' | PROGRAM 'command' | STDIN }  
  [ [ WITH ] ( option [, ...] ) ]  
  
COPY { table_name [ ( column_name [, ...] ) ] | ( query ) }  
  TO { 'filename' | PROGRAM 'command' | STDOUT }  
  [ [ WITH ] ( option [, ...] ) ]
```

Copy (cont.)

- COPY FROM loads data
- COPY FROM will use free space if available, otherwise data is loaded at end of table
 - Acts just like a stream of INSERT's
 - Cannot specify REPLACE, APPEND etc
- Other ops on table continue as normal
- Can only load tables, not views
- All triggers and constraints will be applied
- RULE processing will not be performed

File Export

- Ora2PG also supports exporting and importing data as files

```
ora2pg -t COPY -o data.sql -b data -c config/ora2pg.conf
```

```
import_all.sh -a -h localhost -d training -U jim -o jim
```

Foreign Data Wrapper

- Pros
 - Data is moved only once so the performance is good
 - Transformations can be performed using SQL
- Cons
 - Direct access is required between the source and destination
 - Requires the source database to be quiesced
 - Does not work in all environments such as RDS

Foreign Data Wrapper

- Create a foreign table for each table to migrate

```
CREATE SERVER oracle_server FOREIGN DATA WRAPPER
  oracle_fdw OPTIONS (dbserver 'ORACLE_DBNAME');
```

```
CREATE USER MAPPING FOR CURRENT_USER
  SERVER oracle_server
  OPTIONS (user 'scott', password 'tiger');
```

```
CREATE FOREIGN TABLE oracle.dept (
  deptno      int,
  dname       varchar(14),
  loc         varchar(13)
)
SERVER oracle_server
OPTIONS ( schema 'scott', table 'dept');
```

Foreign Data Wrapper

- Load the data using INSERT statements

```
INSERT INTO dept
SELECT deptno, dname, loc FROM oracle.dept;
```

Replication

- Pros
 - The databases can be synced with no downtime of the source
 - Requires minimal downtime for the production switch over
 - Can be setup far in advance of the cut over date

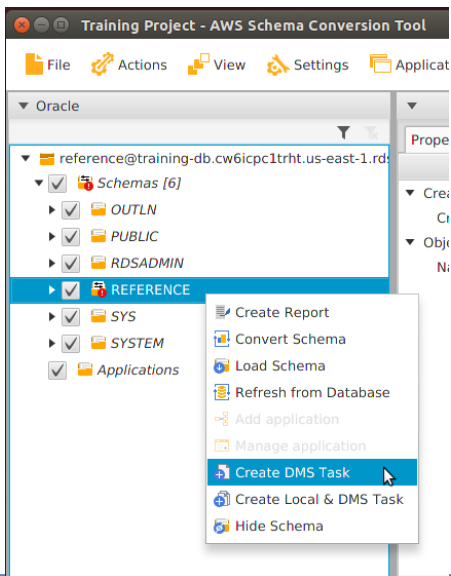
- Cons
 - Can take a very long time to synchronize
 - No transformations on the data being migrated
 - Trigger based systems put extra load on the source

Replication

- There are several cross database replication products including open source options
 - SymmetricDS
 - HVR
 - AWS DMS
- If there is already a replication tool in place, use it

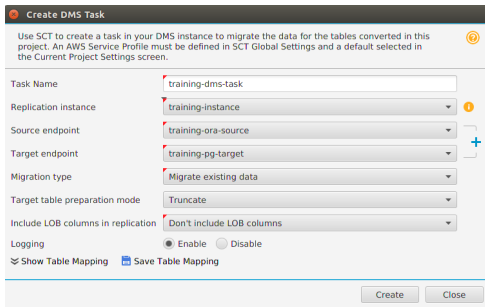
AWS Database Migration Service

- A data migration task can be created from inside of SCT
- Note: Some configuration necessary through the AWS console



AWS Database Migration Service

- Decide on a one time migration or an ongoing replication
- Usually, truncating the data is recommended
- LOBs can take a while to migrate



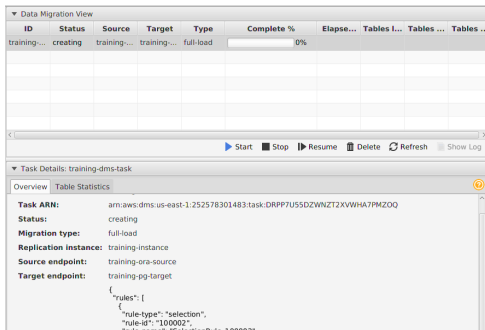
The screenshot shows the 'Create DMS Task' console window. At the top, there is a title bar 'Create DMS Task' and a help icon. Below the title bar is a descriptive text: 'Use SCT to create a task in your DMS instance to migrate the data for the tables converted in this project. An AWS Service Profile must be defined in SCT Global Settings and a default selected in the Current Project Settings screen.' Below this text are several configuration fields:

- Task Name: training-dms-task
- Replication instance: training-instance
- Source endpoint: training-ora-source
- Target endpoint: training-pg-target
- Migration type: Migrate existing data
- Target table preparation mode: Truncate
- Include LOB columns in replication: Don't include LOB columns
- Logging: Enable Disable

At the bottom, there are two buttons: 'Create' and 'Close'. There are also expandable sections for 'Show Table Mapping' and 'Save Table Mapping'.

AWS Database Migration Service

- Start the migration task
- Check for errors



The screenshot displays the AWS Database Migration Service console. At the top, there is a 'Data Migration View' section with a table listing migration tasks. The table has columns for ID, Status, Source, Target, Type, Complete %, Elapse..., Tables I..., Tables ..., and Tables ... The first row shows a task with ID 'training-...', Status 'creating', Source 'training-...', Target 'training-...', Type 'full-load', and Complete % '0%'. Below the table, there are control buttons: Start, Stop, Resume, Delete, Refresh, and Show Log.

Below the table, there is a 'Task Details: training-dms-task' section. It has two tabs: 'Overview' and 'Table Statistics'. The 'Overview' tab is selected, showing the following details:

- Task ARN:** arn:aws:dms:us-east-1:252578301483:task:DRPP7U55DZWNZT2XVWHA7PMZOO
- Status:** creating
- Migration type:** full-load
- Replication instance:** training-instance
- Source endpoint:** training-ora-source
- Target endpoint:** training-pg-target

The 'Table Statistics' tab is also visible, showing a JSON structure for rules:

```
{
  "rules": [
    {
      "rule-type": "selection",
      "rule-id": "100002",
      "rule-name": "SelectionRule 100002"
    }
  ]
}
```

Application Migration

Language Matters

- ORMs (Ruby Python)
- Java
- .NET
- C/C++

Finding Dynamic SQL

- Search for built-in functions
 - SYSDATE
 - NVL
 - DECODE
 - ROWNUM
- Search for system catalogs
 - ALL_TABLES
- Search for DUAL

Data Types

- Many applications need to be changed as data types change
- This is commonly seen around the Oracle NUMBER type
 - Frequently move to BIGINT in PostgreSQL
 - Application code may be treating the columns as INT or BIGDECIMAL

Tuning

Tuning

- Functionally, many things in PostgreSQL work similarly to other databases, but the underlying implementation is different causing different performance results
 - Partitioning
 - Exception Handling
 - Updates (Table Bloat)

Partitioning

- PostgreSQL does not have all partitioning types
 - Hash partitioning is currently not available
 - It is possible to mock the functionality, but performance is poor
- PostgreSQL does not perform well with a large number of partitions
 - Planning times increase based on the partitions
 - Daily partitions going back years are troublesome

Exception Handling

- An exception is an identifier in PL/pgSQL that is raised during execution
- It is raised when an error occurs or explicitly by the function
- It is either handled in the EXCEPTION block or propagated to the calling environment

```
[DECLARE]
BEGIN
    Exception/Error is Raised
EXCEPTION
    Error is Trapped
END
```

Exception Handling

- TIP: Use exceptions only when necessary, there is a large performance impact
 - Sub transactions are created to handle the exceptions

```
CREATE FUNCTION t1()  
  RETURNS void AS $$  
DECLARE  
  i integer;  
BEGIN  
  i := 1;  
END  
$$ LANGUAGE plpgsql;
```

Avg Time: 0.0017ms

```
CREATE FUNCTION t2()  
  RETURNS void AS $$  
DECLARE  
  i integer;  
BEGIN  
  i := 1;  
EXCEPTION  
  WHEN OTHERS THEN  
    RETURN;  
END  
$$ LANGUAGE plpgsql;
```

Avg Time: 0.0032ms

Updates

- PostgreSQL uses MVCC to allow for concurrent access to data
- Updates are essentially an INSERT and a logical DELETE
 - Makes updates very fast since
- Leaves around dead rows that need to be cleaned up
- Causes performance issues if not dealt with

What is MVCC?

- Multiversion Concurrency Control
- Allows Postgres to offer high concurrency even during significant database read/write activity
- Readers never block writers, and writers never block readers
- Reduces locking requirements, but does not eliminate locking

MVCC Behavior

Min **10**
Max

INSERT

Min 10
Max **20**

DELETE

Min 15
Max **30**

UPDATE

Min **30**
Max

- Visibility is driven by XID
- Tuples have an XMIN and XMAX

Updates

- Craft a maintenance plan to clean up dead rows
 - VACUUM
- AutoVacuum does a good job for most workloads
- More extreme behaviors require custom maintenance
- Try to design the bloat out of the application
 - Combine multiple updates on a row into a single operation
 - Separate highly updated columns into different tables

Production Cut Over

Fall-back Plan

- Have a fall-back plan for the transition period
- Do not plan on falling back after the roll-out
 - Plan on falling forward
 - Having bi-directional replication does not work in practice

After Production

- Set monitoring baselines
- Plan and adjust the maintenance schedule
- Enjoy PostgreSQL