Oracle to Postgres Migration
Considerations, Hurdles, and possible Solutions

Presented by Gurjeet Singh
May 19, 2011
Agenda

- Schema Migration
- Data Type Migration
- Data Migration
- Business Logic Migration
- Other Objects
- Connectors / Drivers / Libraries
- Application / SQL Migration
- DBA Migration
- Tools
- Ora2pg
Schema Migration
A.K.A “User” in Oracle
Oracle gives every user her own schema, by default
  • Create a user and schema by the same name
  • The first component in search_path is $user, by default

Identifiers
• Names of schema, tables, columns, functions, …
• Oracle converts them to UPPER CASE, unless quoted
• Postgres converts them to lower case, unless quoted
• You're safe if application quotes/does not quote the identifiers
  • Consistency is the key
Schema Migration

• **Tables**
  - CREATE TABLE is mostly compatible, except
    - Global Temporary table
      - Use LOCAL TEMP tables
    - Partition clauses
      - Use Inheritance, Triggers, and CHECK Constraints
    - INITTRANS, MAXEXTENTS... a.k.a *Storage Clause*
      - Remove them
    - PCTFREE : Use fillfactor

• **Columns**
  - Virtual Columns: Use views
  - Data Types <discussed separately>
Schema Migration

- **Constraints**
  - Primary Key, Foreign Key, Unique, CHECK, NOT NULL
    - They all work pretty much the same

- **Indexes**
  - Btree / Descending: Works
  - Reverse Key / Bitmap / Join: Not implemented (yet)
  - Global: Feature not available
    - Write BEFORE INSERT/UPDATE triggers; very expensive

- **Partitions**
  - Hash, List, Range
    - All work, provided you follow the previous slide
Schema Migration

- **Tablespace**
  - Not really the same thing as Oracle, but serves the same purpose
Data Type Migration
Data Type Migration

- **VARCHAR, VARCHAR2, NVARCHAR, NVARCHAR2**
  - Convert to VARCHAR or TEXT
- **CHAR, NCHAR**
  - Convert to CHAR
- **CLOB, LONG**
  - Convert to VARCHAR or TEXT

- **Note: TOAST**
  - Totally transparent to application.
  - Size limit $2^{30}-1$ (1 GB)
Data Type Migration

- **NUMBER**
  - `BIGINT`, `INT`, `SMALLINT`, `REAL`, `REAL`, `DOUBLE PRECISION`
    - Good Performance, but less control on scale
- **NUMERIC**
  - Unlimited size (implementation specific)
  - Low performance
- **BINARY_INTEGER, BINARY_FLOAT, ...**
- Convert to `INTEGER`, `FLOAT`, ...
- **BLOB, RAW, LONG RAW**
  - Convert to `BYTEA`; requires additional work in application migration
Data Type Migration

- **Date**
  - DATE or TIMESTAMP
  - Also consider timezone effects; TIMESTAMP WITH TIMEZONE
- **DATE arithmetic**
  - DATE + integer
    - Create an overloaded '+' OPERATOR
  - ORAFCE provides last_day, add_months, …
  - TIMESTAMP – TIMESTAMP: Oracle: NUMBER, Postgres: INTERVAL
- **NLS_DATE_FORMAT**
  - Controls output of TO_CHAR and TO_DATE functions
  - In Postgres, controlled by locale settings
  - Note: DateStyle GUC variable
Data Migration
Data Migration

- **Data**
  - Use GUI tools
    - If data type conversion was smooth
    - If database size is not a restriction
  - Use ETL style
    - Use custom application to export in plain-text, CSV
    - Use scripting (Perl!) for transforming
    - Use COPY FROM to load
      - Avoid WAL logging by creating/truncating the table in same transaction
    - Upside: Allows parallel loads
    - Downside: Requires custom development
Data Migration

• **Sequences**
  • Extract `sequence_name.nextval`
  • Use Postgres' `setval('sequence_name', value)`

• **Speeding up**
  • Avoid transaction logging (WAL), as noted previously
  • Defer Index creation until after data load
    - Consider deferring Primary Key and Unique constraints, too; requires that you defer Foreign Key constraints
Business Logic Migration
Business Logic Migration

• **General**
  
  • RETURN becomes RETURNS
  • EXECUTE IMMEDIATE becomes EXECUTE
  • SELECT without INTO becomes PERFORM
    - PERFORM has the same syntax as a full blown SELECT
  • You *must* chose a language
    - CREATE OR REPLACE FUNCTION fn( a INOUT) RETURNS INT AS $$DECLARE … BEGIN …. END; $$ LANGUAGE lang;
  • %TYPE, %ROWTYPE: works
  • cursor_name%ROWTYPE: Doesn't work; Use RECORD
  • REFCURSORS: No replacement; Use Set-Returning-Functions
Business Logic Migration

• General
  • Autonomous transactions
    - Feature not available
      • use DBLink contrib module for loopback connections
  • Ability to COMMIT/ROLLBACK within procedures (only)
    - Because of bounded size of ROLLBACK SEGMENTS
    - Postgres doesn't have rollback segments
    - Use EXCEPTION handling; implemented using SAVEPOINT
      • Not quite the same thing
  • REVERSE LOOPs require switching the start/end conditions
    - FOR i IN REVERSE 1..10 LOOP
    - FOR i IN REVERSE 10..1 LOOP
Business Logic Migration

- **Triggers**
  - Split them into trigger function and the trigger
    - Allows reuse of trigger code

  ```sql
  CREATE OR REPLACE FUNCTION my_trig_fn() RETURNS TRIGGER
  AS $$ ...
  $$ LANGUAGE xxx;
  
  CREATE TRIGGER tbl1_trig1 BEFORE UPDATE ON table
  EXECUTE PROCEDURE my_trig_fn();
  
  :NEW, :OLD
  - Become NEW, OLD
  
  UPDATING, INSERTING => Use TG_OP; consider TG_* variables
  
  Don’t forget to RETURN NEW in BEFORE triggers
Business Logic Migration

- Conditional triggers
  - Execute a trigger only if a condition matches
  - Postgres has it.
- Procedures
  - Postgres has only functions
  - Use RETURNS VOID
  - May need application changes
    - since calling convention in connectors (JDBC, etc.) matters
Business Logic Migration

- Functions
  - RETURN becomes RETURNS
  - Should provide parentheses () even for empty parameter list
    - CREATE FUNCTION fn() RETURNS ...
  - DEFAULT values for parameters
    - Works the same in Postgres
  - Can return pseudo type RECORD
    - The caller needs to know the column names
  - Can return set of records; RETURNS SETOF type
    - Oracle has TABLE FUNCTIONs
Business Logic Migration

- **Packages**
  - A group of variables, functions and procedures
  - Use schema to group functions
  - Use (temporary) tables to replace variables
  - No substitute for private functions, and variables
  - Package Body initialization code: not very often used
    - Call an initializer function in every member function
Business Logic Migration

- **Local functions**
  - Functions within functions, oh my...
    
    ```sql
    create or replace function fn( a int ) return int as
      function fn1( a int ) return int as
        begin
          dbms_output.put_line('World');
          return 1;
        end;
      begin
        dbms_output.put_line('Hello ');
        return fn1(a);
      end;
    end;
    ```
  - Feature not available in Postgres; use normal functions
Other Objects
Other Objects

- **Synonyms**
  - Feature not available
    - Use views for tables
    - Wrappers for functions
- **Database Links**
  - Feature not available
  - Use the dblink contrib module, and views
    - Doesn't allow @ notation, instead whole query is passed to a set-returning-function.
- **CONNECT BY**
  - Use WITH RECURSIVE; SQL compliant and very flexible
Other Objects

• **Materialized Views**
  
  • Create wrapper views
  
  • Jonathan Gardner
    
    - [http://tech.jonathangardner.net/wiki/PostgreSQL/Materialized_Views](http://tech.jonathangardner.net/wiki/PostgreSQL/Materialized_Views)

  • Dan Chak – Materialized Views that Work
    

• **Partitioning**
  
  • Roll your own using Inheritance, Triggers, CHECK constraints, and constraint_exclusion
Other Objects

- **Sequences**
  - Work pretty much the same way as in Oracle.
  - NOCACHE becomes CACHE 1 (or remove this clause)
  - MAXVALUE 9999999999999999999999999
    - reduce limit, or remove clause, \((2^{63})-1\)
  - .nextval, .currval
    - nextval('sequence_name')
    - currval('sequence_name')
- **ORDER/NOORDER**
  - Oracle needs this for Cluster/RAC setups
  - PG doesn't have it
Other Objects

- **Sequences (continued)**
  - NO{CACHE|MINVALUE|MAXVALUE|CYCLE}
    - Replace with NO {*}
    - e.g. NOMINVALUE becomes NO MINVALUE
Application Connectivity
(Drivers)
Application Connectivity

- **ODBC**
  - Works
- **JDBC**
  - Works
  - Consider turning off the autocommit flag in driver
- **.Net**
  - Npgsql
- **OCI**
  - Used by Pro*C programs
  - Oracle Forms
Application Migration
(Queries)
Application Migration

• **Object Names / Identifiers**
  
  • Names of schema, tables, columns, functions, …
  
  • Oracle converts them to *UPPER CASE*, unless quoted
  
  • Postgres converts them to *lower case*, unless quoted
  
  • You're safe if application quotes/does not quote the identifiers
    
    - Consistency is the key
Application Migration

- **Outer Join Syntax**
  - In Oracle, WHERE clause entries mark the NULL augmented side with a (+)
  - Oracle was ahead of the SQL Standards Committee
  - Postgres provides SQL Standard syntax {LEFT|RIGHT|FULL} [OUTER] JOIN; and so does Oracle.

  ```sql
  SELECT e.name, d.name FROM emp e, dept d WHERE e.deptno = d.deptno (+)
  ```

  ```sql
  SELECT e.name, d.name FROM emp e LEFT JOIN dept d ON e.deptno = d.deptno
  ```
Application Migration

• INTERSECT
  • Becomes EXCEPT
• Function Call using named notation
  • => becomes :=
  • For example:
    var = fn( c => 10, a => 'xyz', b => 2.5);
    becomes
    var = fn( c := 10, a := 'xyz', b := 2.5);
• DUAL
  • Just a 1-row x 1-column table for expression evaluation
  • Orafce provides this table.
Application Migration

- **ROWNUM**
  - Use `ROW_NUMBER()` windowing function
  - Use as a wrapper around the main query, if needed.
- **ROWID**
  - Use CTID system column
    - May fail when used in conjunction with partitioning
  - Use OID column
    - Has performance implication since it creates an implicit index
- **Optimizer Hints**
  - Postgres doesn't have them, and doesn't want them.
  - Discard, or keep for future reference; they won't bite you
Edwards, who is a customer, is looking at this 

---

Oracle treats empty string " as NULL. Non-standard and confusing.

- " = " is not true
- Concatenation operator || disregards NULL semantics

SQL> select 'crazy' result from dual where ('a' || '') = ( 'a' || '' );
RESULT
-----
crazy

- Needs careful examination of queries comparing empty string
Built-in Functions
Builtin functions

• **NVL**
  • Provided by Orafce
  • Or use SQL standard COALESCE()
    • More flexible
• **DECODE**
  • Use the SQL Standard CASE clause
  • Postgres now has VARIADIC; it might be possible to implement this where all parameters' data types are same.
• **TO_CHAR()**
  • Postgres has this, but not very robust; requires testing of queries.
  • Orafce provides the 1-argument version
Builtin functions

- **SUBSTR()**
  - Postgres provides this.
  - Postgres also provides SQL standards compliant syntax
- **SYSDATE**
  - Use current_timestamp
DBA Migration
(Database Architecture)
DBA Migration

- Postgres' process architecture is similar to Oracle
  - Have them attend some of Bruce's talks :)
  - No Rollback Segments
  - SGA => ~ shared_buffers
  - PGA => ~ work_mem
  - PMON => Postmaster
  - TNS Listener => Postmaster
  - GRANT/REVOKE => Almost the same; mostly syntax change
Porting Tools
Porting Tools

- **Orafce**
  - A lot of Oracle compatibility functions
    - **DBMS_ALERT**
    - **DBMS_PIPE**
    - **UTL_FILE**
    - **DBMS_OUTPUT**
    - **DBMS_RANDOM**
    - Date operations
      - `to_char(param1)` for various data types
    - **DUAL** table
  - Packages for various platforms (RPM, .deb)
Porting Tools

- Ora2pg
  - Pretty advanced schema and data extraction
  - Extracts PL/SQL too; Packages, Functions, Procedures
  - Tries to convert PL/SQL
  - Export to file, multiple files, compressed
  - Export directly to Postgres
Porting Tools

- **DBD::Oracle**
  - Perl module
  - Develop your own extraction tools
  - Ora2pg uses this
  - Packages available for different platforms

- **Migration Tool Kit (MTK)**
  - Developed by EnterpriseDB
  - Mainly for Oracle to Postgres Plus Advanced Server migration
  - May help in Oracle to Postgres migration
  - Does not convert PL/SQL code
  - Maps data types
Ora2pg
Ora2pg

- **Export Schema**
  - Tables
    - PRIMARY KEY
    - UNIQUE
    - FOREIGN KEY
    - CHECK
  - Views
  - Sequences
  - Indexes
- **Export Privileges**
  - GRANT
Ora2pg

- Export partitions
  - Range
  - List
  - No Hash partitions (yet)
- Ability to export specific objects
- Ability to apply WHERE clause
- Export BLOB type as Postgres' BYTEA
- Export Oracle VIEWS into Postgres TABLEs
- Rudimentary PL/SQL to PL/PgSQL conversion help
- Platform independent
Ora2pg

- Many ways to export
  - Export to a single file
  - Export to multiple files
  - Compress output files using gzip or bzip
  - Export directly to Postgres (not recommended as first step)
• **Steps to export**
  
  • Everything is specified in a config file
    
    • Ora2pg –config config_file.conf
  
  • Define Oracle's connection parameters
    
    • ORACLE_HOME
    • ORACLE_DSN
      
      • dbi:Oracle:host=oradb_host.mydom.dom;sid=TEST
    • ORACLE_USER (recommended to use a sysdba/super-admin role)
    • ORACLE_PWD
    • USER_GRANTS = 0/1 (if running as non admin user)
    • TRANSACTION
      
      • readonly, readwrite, **serializable**, committed
Ora2pg

- Steps to export (continued)
  - Define objects to export
    - SCHEMA: Schema in Oracle
    - EXPORT_SCHEMA 0/1: Create a new schema in Postgres
    - PG_SCHEMA: Export into this Postgres schema (renaming)
    - SYSUSERS: Export objects owned by these system users too.
    - TYPE: What kind of export you want; can specify only one.
      - TABLE, VIEW, SEQUENCE, TABLESPACE
      - FUNCTION, PROCEDURE, PACKAGE
      - TRIGGER, GRANT, TYPE
      - DATA, COPY
      - PARTITION: Work-in-progress
Define objects to export (continued)

- TABLES: List of tables to export
- EXCLUDE: Export all tables, but not these
- WHERE: Apply a WHERE clause to tables being exported
  - WHERE touched_time >= '2010-01-01 00:00:00'
  - WHERE my_table[ ID=200 ]
  - WHERE mytab1[ id=100] date_created > '2010...' mytab2[ id = 54 ]

Modify structure

- MODIFY_STRUCT
  - MODIFY_STRUCT T_TEST1(id,dossier) T_TEST2(id,fichier)
- REPLACE_COLS
  - REPLACE_COLS T_TEST(dico : dictionary,dossier : folder)
Ora2pg

• Control the output
  • DATA_LIMIT: Limit number of incoming rows in memory
  • OUTPUT: output file name; .gz or .bz2
  • OUTPUT_DIR: Where to put output file(s)
  • BZIIP2: Location of bzip2 executable
  • FILE_PER_TABLE: One output file per table
  • FILE_PER_FUNCTION: One function/trigger per file
  • TRUNCATE_TABLE: Truncate the table before loading; DATA/COPY mode only
Ora2pg

- Import into Postgres
  - PG_DSN
    - dbi:Pg:dbname=pgdb;host=localhost;port=5432
  - PG_USER
  - PG_PWD
• **Control objects exported**
  
  • SKIP: List of schema constraint type to skip
    - pkeys, fkeys, ukeys, indices, checks
    - SKIP indices, checks
  
  • KEEP_PKEY_NAMES
    - Keep Primary Key names from Oracle.
  
  • FKEY_DEFERRABLE
    - Mark all Foreign Keys as deferrable
  
  • DEFER_FKEY
    - Defer deferrable Foreign Keys during data load.
  
  • DROP_FKEY
    - Drop Foreign Keys before data load, and recreate them later
Control objects exported (continued)

- **DROP_INDEXES**
  - Drop Indexes before data load, and recreate them afterwards
- **DISABLE_TABLE_TRIGGERs**: 0/USER/ALL
  - Disable triggers before data load, and recreate them afterwards
- **DISABLE_SEQUENCE**
  - Disable altering of sequences during data load.
- **DATA_TYPE**
  - Map Oracle data types to Postgres data types
  - **DATA_TYPE** DATE:timestamp, LONG: text, LONG RAW: text
Ora2pg

- Control objects exported (continued)
  - CASE_SENSITIVE
    - Control identifiers' lower/upper case conversion
  - ORA_RESERVED_WORDS
    - List of words to escape before loading into Postgres
Ora2pg

- **Encoding conversion**
  - **NLS_LANG**
    - Set it to Oracle's encoding
      - NLS_LANG AMERICAN_AMERICA.UTF8
  - **BINMODE**
    - Workaround for Perl's "Wide character in print"
      - BINMODE utf8
        - Results in: binmode OUTFH, ":utf8";
  - **CLIENT_ENCODING**
    - Workaround for: ERROR: invalid byte sequence for encoding "UTF8": 0xe87472
      - CLIENT_ENCODING LATIN9
Thank You