Neue Wege zur Oracle-Migration

Laurenz Albe

laurenz.albe@cybertec.at

Cybertec

Swiss PGDay 2018

CYBERTEC
The PostgreSQL Database Company

Die Problemstellung

DB-Migration besteht aus mehreren Teilen:

- Objektdefinitionen migrieren
  (`CREATE TABLE`, Constraints, Indexe …)
- Tabellendaten migrieren
- In der DB gespeicherten Code migrieren
  (Funktionen, Trigger, Packages, …)
- Applikation anpassen (Treiber, SQL-Dialekt)

DDL migrieren

- DDL muss aus Metadaten konstruiert werden
  - `CREATE TABLE` ist zwar im SQL-Standard
  - … aber nicht die Oracle-Erweiterungen
- Datentypen müssen angepasst werden
  - `NUMBER` wird `integer`, `double precision` oder `numeric`?
  - `DATE` wird `date` oder `timestamp`?
    (Oracle `DATE` ist auf die Sekunde genau)
  - `BLOB` wird `bytea` oder Large Object?

Tabellendaten migrieren

Export aus Oracle ist nicht einfach:

- sqlplus ist möglich für Scripts, aber ziemlich unbrauchbar
- SQL Developer kann CSV exportieren, ist aber unbequem (GUI), kann keine `BLOB`s
- Oracle Foreign Data Wrapper funktioniert
- Oracle macht das wohl absichtlich schwierig

Code in der DB migrieren

- Funktionen, Prozeduren, Packages, Trigger
- Code muss von Hand übersetzt werden
  - PL/SQL ist PL/pgSQL ähnlich, aber nur sehr einfache Funktionen laufen ohne Eingriffe
- Packages umschreiben zu Funktionen in Schema
- Betriebssystemzugriffe aus PL/SQL (Files lesen, E-Mail) besser in PL/Perl oder PL/Python umschreiben
- PL/Java ist eine Idee für Java-Funktionen

Applikation anpassen

- Sollte einfach sein: Oracle und PostgreSQL verwenden SQL, und das ist standardisiert, aber
  - niemand unterstützt den kompletten Standard
  - jeder erweitert den Standard
- Anpassen an geänderte Datentypen
- Applikationscode händisch anpassen, **testen**
- Modul “orafce” bietet Oracle-Funktionen
- Abstraktionsschichten (ORM) helfen viel

SQL-Differenzen: Joins

- Oracle hat spezielle Syntax für Outer Joins:

```
SELECT b.col1, a.col2
FROM base_table b, attributes a
WHERE b.id=a.b_id(+);
```

- muss in Standard-Syntax übersetzt werden:

```
SELECT b.col1, a.col2
FROM base_table b
   LEFT JOIN attributes a ON b.id=a.b_id;
```

SQL-Differenzen: Leerstrings

- Oracle behandelt Leerstrings wie NULL
- anderes Verhalten beim Zusammenfügen:
  `('astring' || NULL) IS NOT NULL`
- Umgehen in PostgreSQL mit `coalesce`:
  `coalesce(col1, '') || coalesce(col2, '')`

SQL-Differenzen: Alias

- Oracle unterstützt diese bequeme Syntax:

```
SELECT *
    FROM (SELECT …);
```

- PostgreSQL und der Standard fordern ein Alias:

```
SELECT *
    FROM (SELECT …) alias;
```

SQL-Differenzen: Sequences

- SQL-Standard Syntax: `NEXT VALUE FOR asequence`
- Oracle Syntax: `asequence.NEXTVAL`
- PostgreSQL Syntax: `nextval('asequence')`
- Oracle lässt `NEXTVAL` in `DEFAULT`-Klauseln nicht zu
  → Vereinfachung in PostgreSQL möglich
- Jetzt, wo beide „Identity Column”-Syntax unterstützen, wird dieses Problem weniger werden

Werkzeuge zur Oracle-Migration

- Es gibt einige kommerzielle Werkzeuge
- Es gibt ora2pg (https://ora2pg.darold.net/)
  - Frei verfügbare Open Source
  - Bewährt (gibt es schon lange)
  - Viele Features (PL/SQL-Migration, Report zur Aufwandsabschätzung)
  - Funktioniert recht gut

Schwachstellen von ora2pg

- Viele Features führen zu vielen Fehlern
- Produziert „monolitisches” SQL-Skript, das dann von Hand bearbeitet werden muss.
- Keine Unterstützung für „bewegte Ziele” (Schemaänderungen parallel zur Migration)

Wünsche an ein neues Werkzeug

- Muss nicht alles können (PL/SQL-Code, …)
  Lieber einfach halten – die schwierigen Dinge bleiben Handarbeit
- Einzelne Objektdefinitionen bequem editieren können
- Mit „beweglichen Zielen“ umgehen können (innerhalb gewisser Grenzen)

Idee: Oracle-FDW verwenden

- Oracle Foreign Data Wrapper ist gut für Datenmigration (kann auch mit ora2pg verwendet werden):
  - Übersetzt zwischen Datentypen
  - Übersetzt Zeichenverschlüsselung (Encoding)
  - Direkter Datentransfer ohne Zwischenspeicherung
- Wenn oracle_fdw für die Daten funktioniert,
  warum nicht auch für die Metadaten?
- So kann alles in PL/pgSQL in PostgreSQL geschehen!
- Leicht zu schreiben, keine Portabilitätsprobleme!

Was ist ein Foreign Data Wrapper?

- Damit kann man „Fremdtabellen” erstellen, die sich wie normale Tabellen benehmen.
- SQL-Statements werden zur externen Datenquelle „umgeleitet”.
- Im SQL-Standard vorgesehen.
- Es gibt FDW für alle möglichen Datenquellen.

Architektur von ora_migrator

- Zwei Hilfsschemata: Oracle and PG-„Stage”
- Oracle-Stage enthält Fremdtabellen für Oracle-Metadaten (Tabellenspalten, constraints, …)
- PG-Stage enthält Momentaufnahme der Daten aus der Oracle-Stage + „übersetzte” Werte (kleingeschriebene Namen, PostgreSQL-Datentypen)
- PG-Stage ist editierbar (Datentypen, PL/SQL-Code)
- PG-Stage kann aus der Oracle-Stage aktualisiert werden (funktioniert für “normale” Schemaänderungen)

ora_migrator Voraussetzungen

- oracle_fdw-Extension installieren
- „Foreign Server” und „User Mapping” erzeugen, User muss den Oracle-Katalog lesen können (z.B. `SELECT ANY DICTIONARY`-Recht)
- oracle_fdw-Konfiguration testen mit
  `SELECT oracle_diag('servername');`
- ora_migrator-Extension mit `CREATE EXTENSION` installieren

Migration durchführen (1)

- `oracle_migrate_prepare`: Stage-Schemas anlegen
- Tabellen in der PG-Stage editieren
- `oracle_migrate_mkforeign`: legt Schemas und „Foreign Tables” für Oracle-Tabellen an
- `oracle_migrate_tables`: erzeugt normale Tabellen und migriert Daten (Fehler werden zu Warnings)
- Parallelisierung: einzelne Tabellen migrieren mit `oracle_materialize('schema', 'table')`

Migration durchführen (2)

- `oracle_migrate_functions`: erzeugt Funktionen
- `oracle_migrate_triggers`: erzeugt Trigger
- `oracle_migrate_views`: erzeugt Views
- `oracle_migrate_constraints`: erzeugt Constraints und Indexe
- `oracle_migrate_finish`: löscht Stage-Schemata
- `DROP EXTENSION oracle_fdw CASCADE;`
  löscht alle Spuren des Migrationsprozesses

Migration „mit einem Klick”

- Einfache Datenbanken (kein Editieren nötig) kann man mit einem Aufruf migrieren:

```
SELECT oracle_migrate(
    server       => 'oraserver',
    only_schemas => '{ORASCHEMA}');
```

- Funktionen und Trigger werden nicht migriert

Probleme bei der Migration (1)

- Null-Bytes in Oracle-Strings

  ```
  invalid byte sequence for encoding "UTF8": 0x00
  ```

- Beste Lösung: in Oracle reparieren
- Problem umgehen:

```
ALTER FOREIGN TABLE s.tab OPTIONS (
   DROP schema, SET table '(SELECT id,
     replace(str, chr(0), '''') AS str, …
     FROM s.tab)');
```

Probleme bei der Migration (2)

- „Illegale” Zeichen in Oracle:
  `invalid byte sequence for encoding "UTF8": 0x80`
- Kann passieren, weil Oracle nicht prüft, wenn Client-Encoding = Server-Encoding
- Wenn „echtes” Encoding der Daten bekannt:
  - PostgreSQL DB mit „echtem” Encoding anlegen
  - Option `nls_lang` auf Foreign Data Wrapper auf Oracle Server-Encoding setzen

Geht das nicht bequemer?

- Tabellen mit `UPDATE`-Statements editieren macht keinen Spaß
- Cybertec entwickelt gerade ein GUI, das ora_migrator angenehmer macht:
  - Funktionen für die Migration werden in der richtigen Reihenfolge aufgerufen
  - Bequemer Editor, um die Metadaten in der PostgreSQL-Stage zu editieren

Wo finde ich ora_migrator?

- ora_migrator ist Open Source auf Github: https://github.com/cybertec-postgresql/ora_migrator
- Bitte ausprobieren, Feedback und Anregungen sind willkommen

Danke für die Aufmerksamkeit!

Noch Fragen?

CYBERTEC
The PostgreSQL Database Company