Do-It-Yourself Data Migration

Dr. Paul Dorsey &
Michael Rosenblum
Dulcian, Inc.

June 5, 2012
Who Am I? - Paul

- **Been Around FOREVER**
  - Spoke at almost every big Oracle conference since ’93
  - First inductee to SELECT Hall of Fame

- **Wrote lots of books**
  - Designer, Developer, JDeveloper, PL/SQL

- **Won lots of awards**
  - One of initial 6 ACE Directors
    - First one fired

- **Built lots of big systems**
  - Air Force Recruiting
  - Ethiopian Ministry of Finance and Economic Development Budget System

- **Known for:**
  - Thick Database approach
  - Business Rules
Who Am I? – “Misha”

◆ Oracle ACE
◆ Co-author of 2 books
  - PL/SQL for Dummies
  - Expert PL/SQL Practices
◆ Won ODTUG 2009 Speaker of the Year
◆ Known for:
  - SQL and PL/SQL tuning
  - Complex functionality
    - Code generators
    - Repository-based development
Data Migration Process

Sources

- Legacy Systems
- Departmental Systems

move data

Destination

- OLTP Repositories
- Warehouse
Migration – The Hidden Nightmare

- Can be as expensive as development... or MORE
- Data cleansing issues can be political
- All systems have LOTS of dirty data.
- 20-40% of all attributes will generate 1 or more issues when reviewed.
- Every migration is unique.
- It is NOT just ETL!!!
Current Migration Project

- **Air Force Recruiting:**
  - **Sources**
    - Active Duty
      - OLTP
      - Warehouse
    - Air Force Reserve
  - **Data volume:**
    - 500 GB of data, 5000 users
    - 800 tables, 7000 attributes
    - 20 years of data (including several architectural shifts)
  - **Target**
    - “Total Force” (similar to Reserve)
  - **Challenges**
    - Overlapping data between Active Duty and Reserve
Tasks

- Profile source
  - How dirty is the data?
- Cleanse data
- Write migration script
- Execute script in test
  - Revise, revise, revise
- Validate migration script
- Execute in production
Profile Tables

◆ Gather information about each table:
  - # of rows
  - Storage space
  - # of attributes
  - Semantics
  - Speed of growth
  - Usage
  - Redundant? Used?
  - Data activity per time

◆ Tools exist to do this but it’s easy to write your own
  - May be easier than finding, installing and customizing a COTS tool.
Gather information about each column:

- Semantics
- Validation, Data Type
- 10 most/least common values
- 10 highest/lowest values
- % null
- # of distinct values
- format_mask, count (*)
  - Group by format_mask
Overloaded Tables and Columns

- 1 physical table > 1 logical table
- 1 physical attribute > 1 logical attribute
- Rules change over time.
Obfuscated Data

- Hard to profile unless…
  - Algorithm is non-random
- Letter → Letter is not OK
  - Decipherable
- Deterministic
  - A → A¹ always
- Length preserved
- Capital letters preserved
- Special characters preserved
- BUT – Why are 17% of people named “Smith”? 
What will profile tell you? (1)

- Identify table errors
  - Tables not used
  - Overloaded tables
  - 1-1 with some other table
What will profile tell you? (2)

- Identify attribution issues (there may be errors)
  - Domain mismatch
  - Out of range
  - Missing NOT NULL constraints
  - Never used
  - Old format masks
  - Out of list domain values
  - Should have check constraints
  - Overloading
  - FK cardinality (too many or not enough)
Cleansing Data

- What can you do to deal with issues discovered during profiling?
  - Throw away bad data?
  - Fix data in sources
  - Fix data in staging area with script
  - Fix data in migration script (usually a bad idea)
  - Relax constraints in target and accept data
    - Maybe fix later and add constraints
  - Enable NO-VALIDATE
    - Evil! Bad! DO NOT USE!
Key questions to ask:

- How many architectural shifts have taken place?
- How much data is really affected?
- How old is it?

Possible solutions:

- Active, changeable data – MIGRATE
- Active, read-only data – WAREHOUSE
- Old, changeable, “once in a while” – SIMPLIFIED STORAGE STRUCTURE
  - Figure out a way to restore into active data
- Old, dirty data, read-only – SIMPLIFIED STORAGE STRUCTURE
Handling Previously Archived Data

◆ If restoration is a requirement, need to ensure that data can be restored to the new system.
  ➢ There may be multiple archive formats.
  ➢ Readability of old tapes may be questionable.
  ➢ Data quality is unknown
    ▪ you can sample / you cannot profile.

◆ Best approach:
  ➢ Introduce an archive viewer
  ➢ Never promise 100% restoration to new system
    ▪ Only restore whatever possible + provide 100% access to old data
Runtime Objects

- What about objects in process?
- Are there things “open” at the end of the day?
- MINIMIZE THESE!
How to Migrate

1. Use a tool
   - Limited
   - Expensive

2. Write migration script by hand
   - Possible
   - Risky

3. Use a repository
   - We already had one.
   - Easier than it looks.
Writing a Migration Script

- Use a repository!
Migration Validation

◆ Internal:
  ➢ Validate data within all records moved.
  ➢ Use a repository.

◆ External:
  ➢ Replicated reports
  ➢ UI screens
Implementation Options

**Big Bang**
- All at once
- All legacy dies

**Parallel**
- Much harder!
Big Bang Approach

◆ Concept:
  ➢ All data is moved at once.
  ➢ New system is turned on; old system is turned off
  ➢ Downtime for the migration

◆ Good:
  ➢ Single point of success/failure
  ➢ Only “one way” transformation needed

◆ Bad:
  ➢ Requires significant downtime
  ➢ Higher risk of complete failure
  ➢ All users have to be trained at once.
“Direct” Parallel Approach

◆ Concept
  ➢ Both systems run in parallel
  ➢ No (or minimal downtime)
  ➢ Data and users are transferred gradually.

◆ Pros:
  ➢ Less impact on business
  ➢ Shorter/no downtime
  ➢ Staged user transition
  ➢ More opportunities to fix detected issues

◆ Cons:
  ➢ Requires bi-directional data transformation in real time
    ▪ Multiplies development cost!
    ▪ Going to the older architecture is always harder!
  ➢ Reports cross two systems
  ➢ Problems with external services (multiple points of entry)
“Special” Parallel Approach

◆ Concept
  - Only old ➔ new (in stages) – no reverse mapping
  - Clear separation of migrated/non-migrated data
  - Locking mechanism
    ▪ Migrated data should not be touched.
  - Constant data profiling
    ▪ To catch unexpected bad data

◆ Pros:
  - Significantly less development needed.
  - No data conflict resolution between old and new.

◆ Cons:
  - Reports are still an issue.
  - No fallback to the old system.
No “Black Box” Approach

- External services MUST be tested.
  - Both inbound and outbound
  - Request this testing well in advance

- Services rely on coordination of many teams and multiple networks.
  - Align all requirements in initial migration plan

- Often cannot control teams from both sides
  - Be patient!
Special System Modules

- Parts of the system (other than data) are also important:
  - Geocoding
  - Email/FTP/Internal Web Services
  - Document management
- Must test and ensure that behavior did not change
- In cases of architectural changes (i.e. different geocoding modules) need to find all product-specific fixes/workarounds
Conclusions

- Migration takes a long time.
- It entails more than just moving data.
- There are lots of players.
- There are lots of moving parts.
- There are lots of decisions to be made.
- It is not just ETL!
Dulcian’s BRIM® and Formspider® Environments

- Full business rules-based development environment
- For Demo
  - Write “BRIM” or “Formspider” on business card
Contact Information

- Dr. Paul Dorsey – paul_dorsey@dulcian.com
- Dulcian website - www.dulcian.com

Latest book: *Oracle PL/SQL for Dummies*