WARNING

This presentation DOES NOT contain any of the following terms:
*performance, optimization, skew, RAC, diagnostics, tracing, 10053**

*Hotsos required reference to 10046 tracing included on slide 10

Realize the
Value of Information℠

Release Management for Database Applications

Dominic Delmolino
Vice President, System Architecture
Research Group
dominic.delmolino@agilex.com

Statements I want you to agree with

- Ease of maintenance is a competitive advantage which can decrease time to market for new features

- Auditing is not just for auditors

- If the QA and Operations folks think you're smart, everyone else will follow

- You'll lose weight jumping up and down for joy after you implement these ideas

Background

- 10 years of performance consulting at Oracle Corporation.
- 2 years as interim CTO of Savant Corporation during its transition to Precise.
- 5 years as Director of Database Engineering at Network Solutions.
- Recently joined Agilex Technologies' Research Group for consulting on data and systems architecture (http://www.agilex.com)
- Personal Oracle blog at http://www.oraclemusings.com/

On the Beauty of Maintenance

- To Maintain: To keep in a state of repair, efficiency and validity
- “How rare it is that maintaining someone else's code is akin to entering a beautifully designed building, which you admire as you walk around and plan how to add a wing or do some redecorating. More often, maintaining someone else's code is like being thrown headlong into a big pile of slimy, smelly garbage.” (Bill Venners)
- “It is a bad plan that admits of no modifications.” (Publilius Syrus)
- “It is impossible to retrofit quality, maintainability and reliability.” (Dr. Alan M. Davis)

Ease of Maintenance: A Competitive Advantage

Well maintained systems:

- Suffer fewer performance crises
- Allow for modifications with clearer understanding of impact
- Live longer
- Reflect positively on their contributors

Faster, Cheaper, Reliable, Efficient

Poorly maintained systems:

- Suffer from complex performance issues
- Call out for “re-implementations”
- Fail often
- Reflect poorly on their contributors

Slower, Expensive, Unreliable

The Zen of Maintenance

- Achieving the Zen of Maintenance begins with a focus on it as a goal during design and development
- This is not about Quality as an abstract, it's about imagining a desired state and developing toward it
- How easy do you want your code deployments to be?

Collective Code Ownership

How can you create maximum visibility for database code changes?

Single development database

- Instant synchronization
- Locking issues
- Overwrite issues
- Need to segment work appropriately
- Global changes immediately visible
- Allows for larger data volumes

Private development databases

- No locking issues
- No overwrite issues
- Delayed synchronization
- Synchronization conflicts
- No need to segment work
- Global change coordination requires extra work

IMHO: Single Database Wins Out

- The advantages:
  - Larger Data Sets (to identify performance issues)
  - Instant visibility
  - Basic locking
- Outweigh the “advantages” of private databases:
  - Privacy and scratch can be accomplished with private schemas instead of whole databases
- Plus, we now can take advantage of a global code audit trail

The Knock on Auditing

- Most people associate auditing with checking to see who updated, deleted or selected data in order to prepare themselves for the “special auditor” visit
- Auditing is covered in Oracle’s Security Guide
- You are warned that it will affect performance
- Yuck! Who wants that?

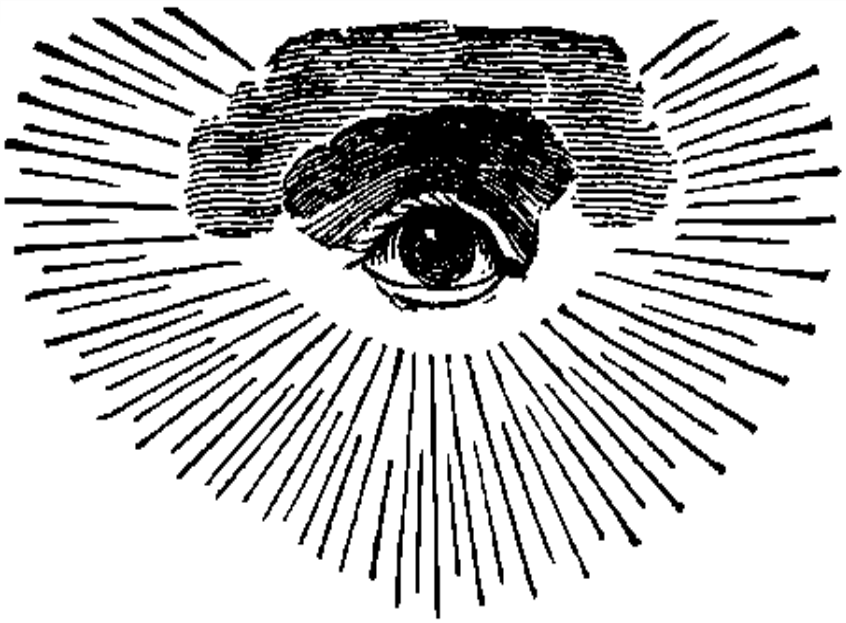
Auditing is the DDL equivalent of 10046 tracing

- Excellent recent blog post by Richard Foote on Oracle Diagnostic Tools
- In reference to trying to find out what happened during a "performance issue"
- DDL Auditing provides the same service for schema code changes – it lets you see exactly what happened to change the structure and code.

"Kids, imagine if we had a camera on mum and saw exactly what she was doing for the entire time she was away getting the milk. We could actually see where all the time was spent, see exactly what happened in the 2 hours she was away. We can account for every moment and see exactly what happened during the 1 hour and 50 minutes of "lost" time."

Benefits of DDL Auditing

- Allows you to fully track all database code and schema changes, including the SQL
- “Performance impact” identifies non-scalable DDL operations used in high-volume transactions
- Makes you look “all-seeing, all-knowing”



What you can audit

- You can audit the following object types:
  - CLUSTER
  - CONTEXT
  - DIMENSION
  - DIRECTORY
  - INDEX
  - MATERIALIZED VIEW
  - PROCEDURE
  - SEQUENCE
  - SYNONYM
  - TABLE
  - TRIGGER
  - TYPE
  - VIEW
- Special additions:
  - You'll need to audit ALTER TABLE and ALTER SEQUENCE specifically
  - You'll also want to audit GRANTS

What you can see

- Setting *audit_trail* to DB_EXTENDED allows you to capture audit statements in the database along with the SQL generating the audit record
- You can query DBA_AUDIT_TRAIL to see the resulting records
- Now you can see that different people worked on the same procedure, or that someone is working from home, or that an object that should not have been touched was changed
- Available information:
  - Operating system username of person making change (includes Windows username if coming from a client machine)
  - Oracle username
  - Host / Terminal (useful to see where people are working from)
  - Timestamp (actually a DATE)
  - Action / Action Name (operation performed)
  - Owner / Object Name (object worked on)
  - New Owner / New Name (captures rename and dependent objects for constraints)
  - SQL Text (extended only) also captures some recursive SQL (for example index creation for primary keys)

Not just for Oracle

- SQL Server 2005 added DDL triggers, see http://www.sqldbatips.com/presentations/PASS2006.ppt for “Building a DDL Audit Solution using SQL Server 2005”
- MySQL Binary Log, see http://pooteeweet.org/files/phptek06/database_schema_deployment.pdf
- DB2 db2audit
- Best part about Oracle’s solution is that all data is in a simple table (and you can create an XML log too!)

How database objects are different from regular code

- Structural changes to the database schema often require large scale data migration and/or transformation
- Generally, you don't "re-create" a table every time you modify it
- It's rare that you "re-install" a database
- You have requirements to "hot" patch or upgrade the schema while applications are accessing it

Look at your deployment requirements to see how to set up source control

"IT" Application:

- Custom / Bespoke internal enterprise or customer facing application
- Few installations, usually only one version running
- Large data volumes
- Directly related to your organization's operations
- Databases almost always "updated" and not "created"
- Flexible ways to minimize downtime through coordination
- Frequent updates

"ISV" Application:

- Commodity application
- Many installations, many versions in place
- New installations of many different versions
- Upgrades from lots of version combinations
- Highest priority is clean installs and upgrades – more so than minimizing downtime
- Fewer, larger updates

Differences in deployment packaging

"IT" Application

- Always upgrading, rarely installing from scratch
- Important to identify what was delivered in each upgrade
- Always need to minimize downtime
- Always need to deal with legacy data
- May need to allow multiple versions to keep running at the same time

"ISV" Application

- Always need to support "create database from scratch" for all supported versions
- May have less emphasis on detailing upgrade changes
- May be able to dictate downtime
- May not have to support multiple versions at the same time

Example deployment packages

“IT” Application

- Create v1 scripts
- Create v1 → v1.3 scripts
- Create v1.3 → v1.6 scripts
- Create v1.6 → v2.0 scripts
- Create v2.0 → v2.3 scripts

“ISV” Application

- Create v1 scripts
- Create v2 scripts
  - Create v1 → v2 scripts
- Create v3 scripts
  - Create v1 → v3 scripts
  - Create v2 → v3 scripts
  - Or, require running of v1 → v2 → v3 scripts for v1 installations

We'll focus on an "IT" application for now

- Emphasize upgrades over installs
- Need detailed lists of what's been changed in each upgrade
- Heavy emphasis on minimizing downtime, may have multiple versions running at the same time during cut-over and upgrade
- Well-known, legacy data readily available for testing
- The main purpose of source code control for this kind of application is to create delivery packages for deployment by operations and to ensure coordination between development, QA and production

Minimizing downtime thru having multiple versions running at the same time

- PL/SQL package versioning – a versioned API
- Every package name has a version number embedded in it
- Benefits include easy comparison / diff between versions from ***within*** the database (TOAD Object Compare)
- Do all packages get a new version number when a new version is created?
- Doesn't this defeat the purpose of source code control if we have PKG_V1 and PKG_V2?
- Doesn't 11g support package versioning?
- Are there better ways to do this with synonym management? Or file names in source code control?

Tantalizing support for object versions “editions” in 11g

- From $OH/rdbms/admin/cdcore.sql:

```
remark "_CURRENT_EDITION_OBJ" describes all objects visible in the current
remark edition. Starting with release 11 of the Oracle DB, any views exposing
remark metadata for versionable objects (package, function, procedure, object
remark type, view, synonym, library, trigger and assembly) must use this view
remark instead of obj$.
remark
remark "_ACTUAL_EDITION_OBJ" describes all actual objects (not stubs) in all
remark editions. Use this view instead of obj$ to describe all versions of
remark a versionable object.
```

- Create “editions” of packages and other objects
- Alter session to refer to a specific edition name
- Not yet implemented in 11g R1, but error messages have been defined
- Perhaps this will replace the clunky method from the prior slide

Moving on...

- During development, database engineers work directly against single development database – they ***do not work with script files***.
  - Locking provided by database or IDE (TOAD source control for procedures / packages)
  - Supports rapid development without switching to filesystem and worrying about packaging
  - Backup and versioning provided by database backup
- Since code is directly edited and compiled, and all changes are tracked by DDL auditing, source code control packaging is delayed until delivery into Integration or QA test environments
- If we can generate code out of the database, we eliminate user scripting errors

Assembling a deployment package

- Use your new-found knowledge about DDL auditing to assemble a change list:
  - Find all new objects
  - Ignore objects created / destroyed during development
  - Find all alterations
  - Question why objects have been modified
  - Identify which developers are responsible for which object changes
- Take advantage of all methods to fill out your change list: DDL audit trail, schema comparisons from IDEs (TOAD, SQL Developer), developer meetings to review who changed what and why

Determine your standards for source code delivery

- Directory structure ideas:
  - /admin – for files that retain the same name but need to be versioned (i.e., init.ora)
  - /v01_00_00 – for version 1 package, includes initial table creates (may also have subdirectories for ddl, plsql, dml, and installation scripts)
  - /v02_00_00 – for scripts that upgrade v1 to v2
- File naming ideas:
  - One file per object
    - Pro: easy to trace / track, probably not a lot of files for v1 to v2 upgrade, easy to generate
    - Con: lots of files for v1
  - File name includes object owner, object type, object name, action indicator (create, modify, drop), project or bug ticket number
    - Example action indicators: mk (create), ch (modify), rm (drop)
    - Sample file names
      - mk-tbl-scott-emp.sql
      - ch-tbl-scott-emp-13058.sql

Use code generation techniques to create scripts

- Automatically generate the necessary file names
- Ensure consistent code formatting
- Inject source code control tags, possible other tags
- Ridiculously easy since the advent of DBMS_METADATA package
- May need special modifiers for your environment (storage option replacement, online index creation, statistics computation)
- May need to hand-code ALTER scripts

Spend more time on deployment sequencing instead of script coding

- What can be pre-deployed without affecting the system?
  - All new objects, including new package versions.
  - Can we do any staging of data conversions?
- What requires an outage?
  - Some table modifications
- What can be done post-deployment?
  - Conversion of older data?
- Think about sequencing (dependencies) and reducing any downtime required
- If you're lucky enough to have a nightly copy of production used for "read-only" reporting, test and re-test your installation scripting.

Create installation / upgrade scripts that are easy to understand

- Getting close to A:INSTALL
- Name your installation scripts with sequencing, schema and database names:

  install-v01-fin-ddl-pre-001.sql
  install-v01-fin-plsql-pre-002.sql
  install-v01-fin-ddl-out-001.sql
  install-v01-fin-dml-pst-001.sql

- Create an installation guide that explains the sequencing, timing and purpose for each script

Results

- Developers focus on new features instead of source code scripting and control
- Code change tracking done automatically using DDL auditing
- More time can be spent on analyzing deployment to minimize downtime
- Code generation done automatically to ensure consistent scripts, clearly identifiable file names, injection of necessary tags

In the end...

- More focus on features, less on change control bookkeeping
- Auditing is your friend
- Clear and consistent identification of changes makes you look like a genius
- Weight loss epidemic occurs as everyone jumps for joy!

Realize the
Value of Information℠

Thank You!

Questions?