Data Tracking: On the Hunt for Information About Your System

Michael Rosenblum & Grigoriy Novikov

Dulcian, Inc.

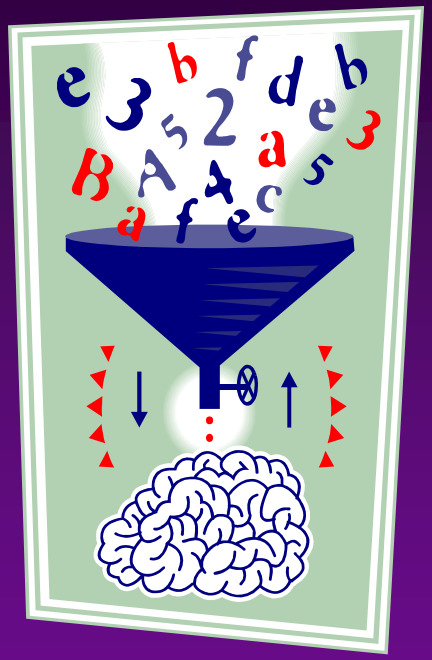
www.dulcian.com

Who are we?
“Misha” and “Grisha”

- Database fire-fighting squad:
  - New feature research
  - SQL and PL/SQL tuning
  - Complex functionality
    - Code generators
    - Repository-based development

Sources of Information

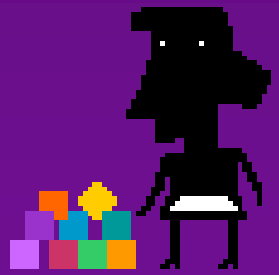
- Provided by Oracle:
  - Data Dictionary views
  - Built-in logging mechanisms
  - Built-in tracing mechanisms
- Custom solutions:
  - Code instrumentation
  - Code instrumentation
  - More code instrumentation ☺



Data Dictionary

Data Dictionary for Developers

- Good news:
  - There are a lot of GUI tools on the market.
- Bad news:
  - Without GUI tools, too many developers become powerless.
- Extra benefit:
  - Repository-based development allows you to build very efficient generic solutions.



Data Dictionary 101

- Static data dictionary views
  - USER_* - Everything that directly belongs to the current user
  - ALL_* - Own + everything that was granted to the user from other users
  - DBA_* – Everything

- Dynamic data dictionary views:
  - V$_* - Use real-time data and provide the most up-to-date information
  - Warning: Majority of those views have to be granted to your users by DBAs

Static Data Dictionary (1)

- Main structural information
  - *_OBJECTS
  - *_TABLES
  - *_TAB_COLUMNS
  - *_INDEXES
  - *_IND_COLUMNS/*_IND_EXPRESSIONS
  - *_CONSTRAINTS
  - *_CONS_COLUMNS
  - *_SEQUENCES
  - ALL/DBA_DIRECTORIES

Static Data Dictionary (2)

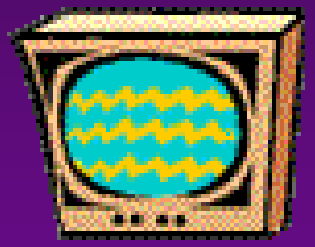
- Code
  - *_SOURCE
  - *_VIEWS
  - *_TRIGGERS
  - *_TYPES/*_TYPE_METHODS
- Advanced code
  - *_PROCEDURES
  - *_ARGUMENTS
- PL/Scope
  - *_IDENTIFIERS

Static Data Dictionary (3)

- Special info
  - *_DEPENDENCIES
  - Fine-grain dependency
    - Officially Oracle does NOT provide data dictionary views to work with this info.
    - A number of people found a workaround ☺
      - DBA_DEPENDENCY_COLUMN © Toon Koppelaars and Rob Van Wijk
      - DBA_DEPENDENCY_ARGS – my own variation (with and without PL/Scope)

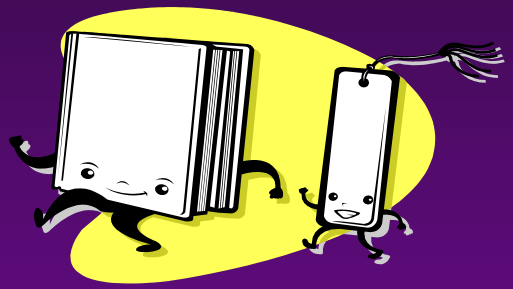
Static Data Dictionary (4)

- Security
  - *_TAB_PRIVS – all objects, not just tables
  - *_SYS_PRIVS – explicitly granted privileges
  - *_ROLE_PRIVS – privileges via roles
- Special cases
  - *_LOBs
  - *_NETWORK_ACL_PRIVILEGES
  - *_PLSQL_OBJECT_SETTINGS
  - *_RECYCLEBIN
  - *_UPDATABLE_COLUMNS



Dynamic Data Dictionary (1)

- Active usage
  - V$PROCESS/V$PROCESS_MEMORY
  - V$SESSION
- Statistics
  - V$CLIENT_STATS
  - V$SESSTAT/V$SESS_IO
  - V$SYSSTAT
  - V$METRIC_* - to describe detected statistics



Dynamic Data Dictionary (2)

- Special cases:
  - V$DBLINK
  - V$PARAMETER
  - V$TEMPORARY_LOBS
  - V$TEMPSEG_USAGE
- Lookups
  - V$TIMEZONE_NAMES
  - V$RESERVED_WORDS

Dynamic Data Dictionary (3)

- SQL:
  - V$OPEN_CURSOR – useful to know if somebody does not close cursors
  - V$SQL_CURSOR – all cursors in the cache
  - V$SQL_SHARED_CURSOR – explains why there are multiple cursor versions
  - V$SQL (V$SQLAREA is an aggregate) – all SQL
  - V$SQL_BIND_* (CAPTURE,DATA,METADATA) – extracts values of bind variables

DULCIAN INC.™

Logging

Application Logging

- Advantages:
  - Customized information when needed
- Disadvantages:
  - Requires discipline of the whole development group
- Key technologies
  - Autonomous transactions
  - Conditional compilation

Indestructible Log (1)

```
create table t_log (
   id_nr number,
   timestamp_dt timestamp,
   log_tx varchar2(4000),
   log_cl CLOB,
   current_user varchar2(32) default
         sys_context('USERENV','CURRENT_USER'),
   ip_address varchar2(256) default
         sys_context('USERENV','IP_ADDRESS')
                     );
create sequence log_seq;
```

Indestructible Log (2)

```
create or replace package log_pkg
is
    procedure p_log (i_tx varchar2);
    procedure p_log (i_cl CLOB);
end;
/
create or replace package body log_pkg is
    procedure p_log (i_tx varchar2) is
        pragma autonomous_transaction;
    begin
      insert into t_log (id_nr, timestamp_dt, log_tx, log_cl)
      values (log_seq.nextval, systimestamp,
          case when length(i_tx)<=4000 then i_tx else null end,
          case when length(i_tx)>4000 then i_tx else null end);
      commit;
    end;

    procedure p_log (i_cl CLOB) is
        pragma autonomous_transaction;
    begin
        insert into t_log (id_nr, timestamp_dt,log_cl)
        values (log_seq.nextval, systimestamp,i_cl);
        commit;
    end;
end;
/
```

Indestructible Log (3)

```
declare
    v_tx varchar2(256);
begin
    log_pkg.p_log ('Before query:'||
        dbms_utility.format_call_stack);
    select ename
    into v_tx
    from scott.emp;
    log_pkg.p_log ('After query');
exception
    when others then
        log_pkg.p_log
          (dbms_utility.format_error_stack);
        log_pkg.p_log
          (dbms_utility.format_error_backtrace);
        raise;
end;
```

Conditional Compilation (1)

```
create or replace procedure p_conditional
is
    v_tx varchar2(256);
begin
  $if $$DebugTF $then
    log_pkg.p_log
       ('Before query:'||dbms_utility.format_call_stack);
  $end

  select ename
  into v_tx
  from scott.emp;

  $if $$DebugTF $then
       log_pkg.p_log ('After query');
  $end
exception
  when others then
     log_pkg.p_log(dbms_utility.format_error_stack);
     log_pkg.p_log
                  (dbms_utility.format_error_backtrace);
     raise;
end;
```

Conditional Compilation (2)

```
SQL> exec p_conditional
BEGIN p_conditional; END;
*
ERROR at line 1:
ORA-01422: exact fetch returns more than requested number of rows
ORA-06512: at "SCOTT.P_CONDITIONAL", line 18
ORA-06512: at line 1
SQL> select count(*) from t_log;
  COUNT(*)
----------
         2
SQL> alter procedure p_conditional compile
  2 plsql_ccflags='DebugTF:TRUE' reuse settings;
Procedure altered.
SQL> exec p_conditional
BEGIN p_conditional; END;
*
ERROR at line 1:
ORA-01422: exact fetch returns more than requested number of rows
ORA-06512: at "SCOTT.P_CONDITIONAL", line 18
ORA-06512: at line 1
SQL> select count(*) from t_log;

  COUNT(*)
----------
         5
SQL>
```

System Logging

- Levels of information:
  - Core info
    - Process
    - Session
  - Granular info
    - Client
    - Module
    - Action

- Why bother?
  - StateLESS implementation spawns logical session between multiple physical sessions.

Setting Granular Info (1)

```
-- Client Stuff
Begin
   -- set it to anything you want to describe
   -- the session. Otherwise useless
   DBMS_APPLICATION_INFO.SET_CLIENT_INFO
     ('This is my test-run');

   -- Key setting for debugging!
   -- This ID is traceable.
   DBMS_SESSION.SET_IDENTIFIER ('misha01');
end;
/

-- Visibility:
select sid, client_info, client_identifier
from v$session
```

Setting Granular Info (2)

```
-- Client Stuff
Begin
   -- Additional info: module and action
   DBMS_APPLICATION_INFO.SET_MODULE
      (module_name=>'HR',
       action_name=>'SALARY_MAINT');
end;
/

-- Visibility:
select sid, module, action
from v$session
```

DULCIAN INC.

Introduction to Oracle Trace

DULCIAN INC.

Attention!

- WARNING:
  - This is a really advanced topic.
  - It requires access to the server file system.
  - It requires coordination of both development and DBA teams.

What is trace? - History

- Oracle developers are human. (I hope ☺)
  - They code.
  - They instrument their code (as we all do) with output messages.
  - They DEBUG using those messages.
- A long time ago (~1992) Oracle decided to let end-users utilize the internal debugging mechanism.
  - Initially to help servicing SR
  - Later to solve problems by themselves

DULCIAN INC.

The trace is…

- Oracle Trace is an internal tool that became available to end users.
- It makes sense in its entirety only to Oracle software engineers.
- It generates a lot of obscure data that could be explained (somewhat) by either built-in Oracle tools or by reading a lot of additional literature.

Trace Events

- The most common
  - 10046 – main trace event
    - Level 1 – parse/execute/fetch
    - Level 4 = 1+ bind variables
    - Level 8 = 1 +waits
    - Level 12 = 1 + waits + binds
  - 10053 – optimizer
    - Level 1 – stats and computations
    - Level 2 – computations only
- Hundreds of others (some are VERY obscure!)

How do you enable trace?

- Direct “ALTER SESSION”
  - The most flexible option
  - Many additional parameters
- DBMS_MONITOR
  - More “civilized” interface
  - Covers only 10046
- Lots of other options (ORADEBUG, DBMS_SUPPORT etc)

How do you aggregate trace?

- TRCSESS
  - Allows multiple files to be consolidated into a single one by specified parameter (session/client ID/module/action)
- TKPROF
  - Making trace files human-readable
  - Good news: Allows aggregation
  - Bad news: You could miss key information (therefore, reading the raw trace is always a good skill to have)

Special Types of Trace (1)

- Single SQL trace

```
ALTER SESSION/SYSTEM SET EVENTS
  'SQL_TRACE [SQL:sql_id|sql_id]
  wait=true|false,
  bind=true|false,
  plan_stat=never|first_execution|
                  all_executions,
  level=12'
```

Example of Single SQL Trace(1)

```
-- Code to be reviewed
declare
    v_tx varchar2(10):='CLERK'; -- 'MANAGER'
    v_nr number;
begin
    select /*+ MISHA_EMP*/ count(*)
    into v_nr
    from scott.emp
    where job=v_tx;
end;

-- Add extra info for identification
alter session set
  tracefile_identifier = 'mishaA'|'mishaB'
```

Example of Single SQL Trace (2)

```
-- Find in V$SQLAREA a query you need
select *
from v$sqlarea
where sql_text like '%MISHA_EMP%'

-- Enable trace
alter system set events
  'sql_trace [sql:c1mnus9wqgz3b]
  wait=true,bind=true,
  plan_stat=all_executions,level=12'

-- later - disable trace
alter system set events
  'sql_trace [sql:c1mnus9wqgz3b] off'
```

Example of Single SQL Trace (3)

```
-- Run blocks in two sessions - got two files
ora11g_ora_1996_mishaA.trc
ora11g_ora_4264_mishaB.trc

-- Aggregate two files into a single one:
C:\>trcsess output=c:\temp\misha_agg.trc
        c:\temp\*misha*.trc service=SYS$USERS

-- Make it readable:
C:\>tkprof c:\temp\misha_agg.trc
        c:\temp\misha_agg_review.txt
```

<Show trace and aggregated info>

Special Types of Trace (2)

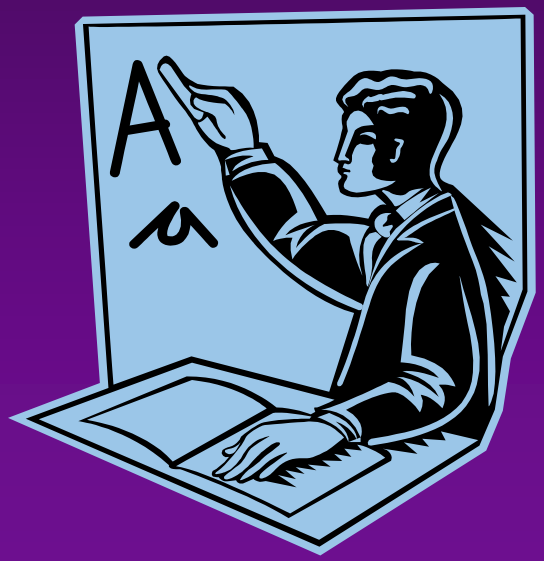
- Process trace

```
ALTER SESSION/SYSTEM SET EVENTS
  'SQL_TRACE {process:pid=}…' or
  'SQL_TRACE {process:pname=}…' or
  'SQL_TRACE {process:orapid=}…'
```

- Why bother?
  - Multi-threaded environments
  - Specially named Oracle processes (like Data Pump)

DULCIAN INC.

Logging/Tracing Use Case



Real World Example

- Environment:
  - Stateless implementation
    - Users have logical sessions during the day between logon and logoff.
    - Logical sessions consist of multiple physical sessions.
  - Users work with multiple modules.
- Tasks:
  - Trace a logical session of a single user.
  - Trace activities related to a module.

Setting

```
-- Login Procedure
create or replace procedure p_login
   (in_user_tx varchar2)
is
begin
   DBMS_SESSION.SET_IDENTIFIER (in_user_tx);
end;

-- Maintenance procedure
create or replace procedure p_updateSal
    (in_empno_nr number, in_sal_nr number) is
begin
    dbms_application_info.set_module
       ('SALARY_MAINT', 'UPDATE');
    update emp
       set sal = in_sal_nr
     where empno = in_empno_nr;
    dbms_application_info.set_module
        (null,null);
end;
```

Tracing Status

```
-- Trace user MISHA
-- Trace module SALARY_MAINT
-- check in DBA_ENABLED_TRACES
-- Afterwards disable ALL individually
begin
    dbms_monitor.client_id_trace_enable(
        CLIENT_ID=>'Misha')

    dbms_monitor.serv_mod_act_trace_enable(
        service_name   => 'SYS$USERS',
        module_name    => 'SALARY_MAINT',
        waits          => true,
        binds          => true,
        plan_stat      =>  'ALL_EXECUTIONS');
end;
```

Actions of User #1:

```
-- login
SQL> connect SCOTT/TIGER@ora11g
SQL> exec p_login('Misha')
PL/SQL procedure successfully completed.
SQL> select sal from scott.emp where empno=7369;
       SAL
----------
      1000

SQL> connect SCOTT/TIGER@ora11g
SQL> exec p_login('Misha')
PL/SQL procedure successfully completed.
SQL> exec p_updateSal(7369,1000)
PL/SQL procedure successfully completed.
SQL> exit
```

Actions of User #2:

```
-- login
SQL> connect SCOTT/TIGER@ora11g
SQL> exec p_login('John')
PL/SQL procedure successfully completed.
SQL> select sal from scott.emp where empno=7499;
       SAL
----------
      2000

SQL> connect SCOTT/TIGER@ora11g
SQL> exec p_login('John')
PL/SQL procedure successfully completed.
SQL> exec p_updateSal(7499,2000)
PL/SQL procedure successfully completed.
SQL> exit
```

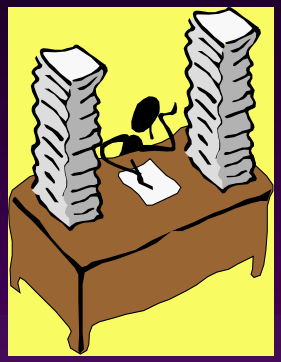
Aggregation

```
-- Generated three files
ora11g_ora_4352_Emp_John.trc
ora11g_ora_4692_Emp_Misha.trc
ora11g_ora_4140_Emp_Misha.trc

-- Aggregate by client:
C:\>trcsess output=c:\temp\misha_client.trc
        c:\temp\*emp*.trc clientid=Misha

-- Aggregate by module:
C:\>trcsess output=c:\temp\misha_module.trc
        c:\temp\*emp*.trc module=SALARY_MAINT

-- run TKPROF to make it readable
```



<Show trace and aggregated info>

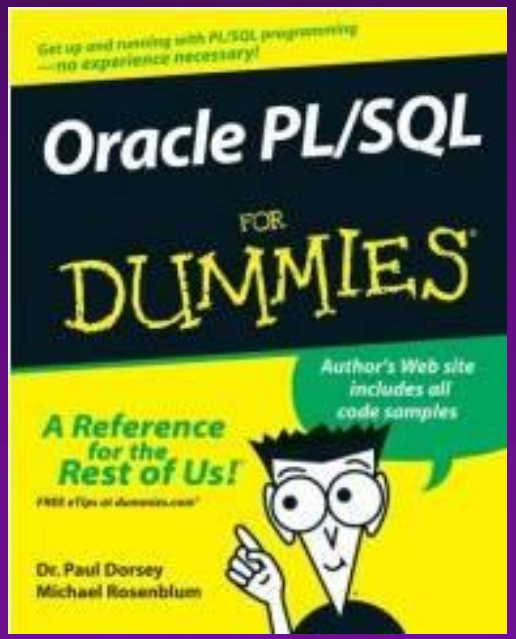
DULCIAN INC.

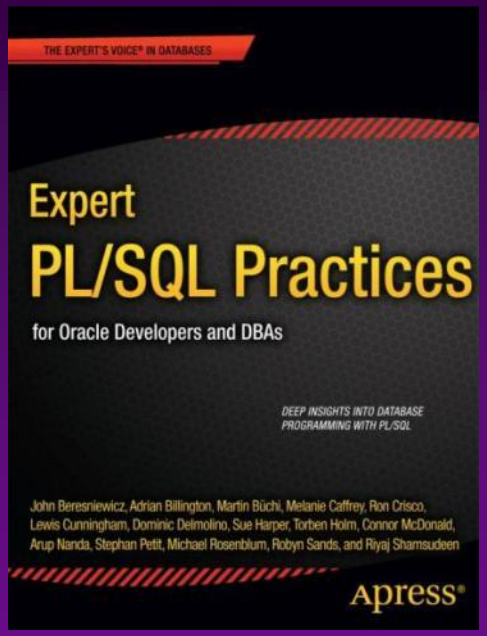
Summary

- You need to understand your own system not only now, but a couple of years later.
  - Debugging messages are crucial for job security ☺
- Oracle provides tons of useful information.
  - As long as you know how to interpret it ☺
- Good tracing needs good logging
  - THIS Oracle cannot ask the gods for something it doesn't know ☺

Contact Information

- Michael Rosenblum – mrosenblum@dulcian.com
- Grigoriy Novikov – gnovikov@dulcian.com
- Blog – wonderingmisha.blogspot.com
- Website – www.dulcian.com





Available now:
Expert PL/SQL Practices