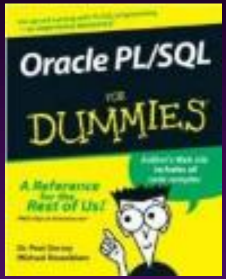
A Developer's Approach to Code Management

Michael Rosenblum

www.dulcian.com

Who Am I? – "Misha"

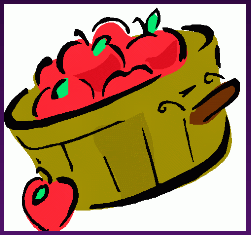
- Oracle ACE
- Co-author of 3 books
  - *PL/SQL for Dummies*
  - *Expert PL/SQL Practices*
  - *Oracle PL/SQL Performance Tuning Tips & Techniques*



- Known for:
  - SQL and PL/SQL tuning
  - Complex functionality
    - Code generators
    - Repository-based development

Code Management???

- The biggest problem:
  - No agreement about what is MANAGEMENT
  - Even less agreement about what is CODE
- Results:
  - Instead of comparing concepts we usually compare implementations ➔ therefore we are comparing apples to oranges!!!
    - … because each implementation implies a very specific set of requirements.



Definitions (1)

- Code = Anything that defines and implements business rules
  - Programs
  - Structural rules (constraints!)
  - Metadata + Generators (programs that write programs that …)

Definitions (2)

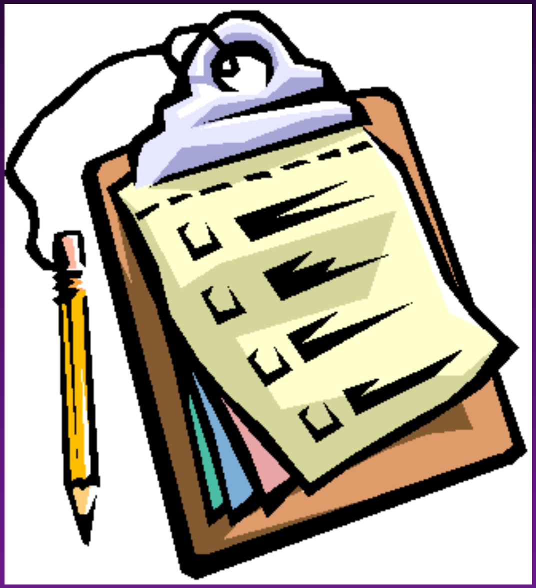
- Management = understanding how your code base transforms over a period of time and being able to explain:
  - What changes happen?
  - How the changes happen?
  - Why the changes happen?

Who benefits?

- Implementations:
  - Management-oriented approach
    - Key question: “Who done it?”
    - Solution: Try to preserve every line change and associate it with a specific person ➔ foremost forensic tool (blaming game!)
    - Problem: Large systems are quickly overwhelmed by the total volume of micro-changes
  - Development-oriented approach
    - Key question: “How do we create the next release?”
    - Solution: Managing macro-changes instead of micro-changes
    - Problem: Requires a different level of organization

Agenda

- I. Area:
  - Database
- II. Language:
  - PL/SQL
- III. Level:
  - Development-oriented approach



Versioning-“Lite”

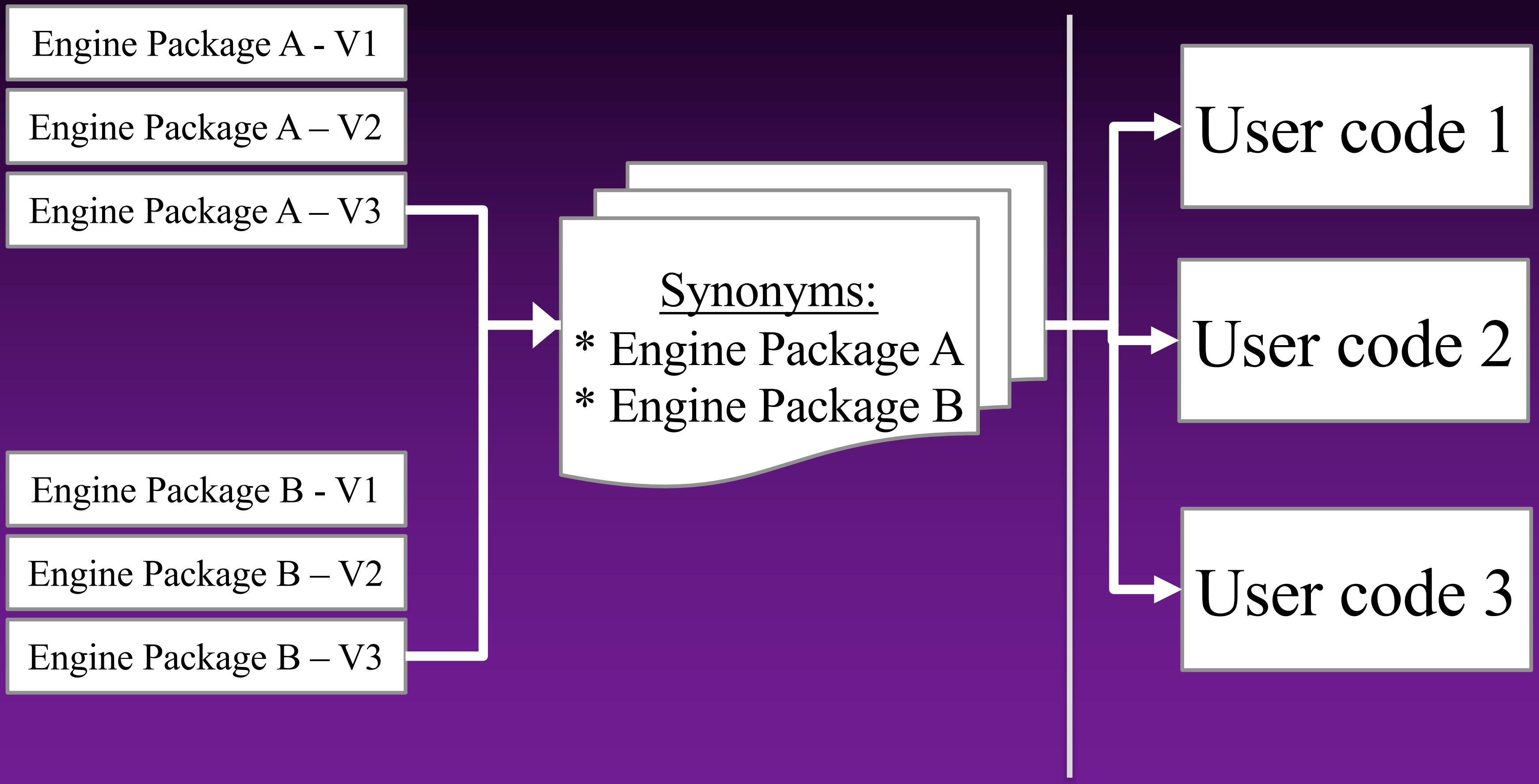
K.I.S.S.

- Please, remember:
  - There is no such thing as “all-or-nothing” approach!
  - Complexity/cost of the solution should match the scope of a problem

Synonym Manipulation

- Solution:
  - Synonyms for external references + unique object names for all versions:
    - … i.e. synonym A_PKG + package A_PKG_V1, A_PKG_V2 etc.
- Downside:
  - Recompilation of referenced objects
- Useful in:
  - Environments with clear separation of engine code vs. customer code

Synonyms - Illustration



Triggers

- Solution:
  - Setting up BEFORE/AFTER DDL triggers in relevant schemas
    - Database-level triggers must be disabled before any Oracle patches ➔ high cost of error ➔ strongly NOT recommended
    - Invalid triggers would block ANY DDL from being fired
    - BEFORE-triggers also work as security features
      - Example: blocking TRUNCATE command
    - Exceptions raised in AFTER-trigger would not impact execution itself

Triggers – Example (1)

```
CREATE TABLE ddl_audit_tab (
ddl_type_tx       VARCHAR2(30),
object_type_tx    VARCHAR2(30),
object_name_tx    VARCHAR2(30),
ddl_date_dt       TIMESTAMP,
code_cl           CLOB);

CREATE OR REPLACE TRIGGER ddl_audit_trg BEFORE DDL ON SCHEMA
DECLARE
    v_lines_nr PLS_INTEGER;
    v_sql_tt ora_name_list_t; -- TABLE OF VARCHAR2(64)
    v_cl CLOB;

    PROCEDURE p_add (i_tx VARCHAR2) IS
    BEGIN
        dbms_lob.writeappend(v_cl,length(v_buffer_tx), v_buffer_tx);
    END;
...
```

TODO

Triggers – Example (2)

```
BEGIN
-- security section
  IF ora_dict_obj_name = 'DDL_AUDIT_TAB' THEN
    raise_Application_error(-20001,'Cannot touch DDL_AUDIT_TAB!');
  END IF;

  -- put DDL together
  v_lines_nr := ora_sql_txt(v_sql_tt);
  dbms_lob.createTemporary(v_cl,true,dbms_lob.call);
  FOR i IN 1..v_lines_nr LOOP
    p_add(v_sql_tt(i));
  END LOOP;

  -- store
  INSERT INTO ddl_audit_tab
    (ddl_type_tx,object_type_tx,object_name_tx,ddl_date_dt,code_cl)
  VALUES
    (ora_sysevent,ora_dict_obj_type,ora_dict_obj_name,SYSTIMESTAMP,v_cl);
END;
```

Triggers – Example (3)

```
SQL> CREATE TABLE tst1(a number);
Table created.
SQL> SELECT * FROM ddl_audit_tab;
DDL_TYPE_TX OBJECT_TYPE_TX  OBJECT_NAME_TX DDL_DATE_DT CODE_CL
----------- --------------- -------------- ----------- ----------------------------
CREATE      TABLE           TEST01         29-JAN-14   create table tst1(a number)

SQL> TRUNCATE TABLE ddl_audit_tab;
TRUNCATE TABLE ddl_audit_tab
               *
ERROR at line 1:
ORA-00604: error occurred at recursive SQL level 1
ORA-20001: Cannot touch DDL_AUDIT_TAB!
ORA-06512: at line 24
SQL> SELECT count(*) FROM ddl_audit_tab;
  COUNT(*)
----------
         1
```

Homegrown Versioning (1)

- Problem:
  - Classic 3-tier IT system ➔ significant downtime cost/efforts
  - Small part of a system has constant flow of change requests
  - Structure of requests is very clear
    - Take N parameters / Do something / Show results
- Conclusion:
  - The most efficient method is to introduce a localized repository-based purpose-built solution.

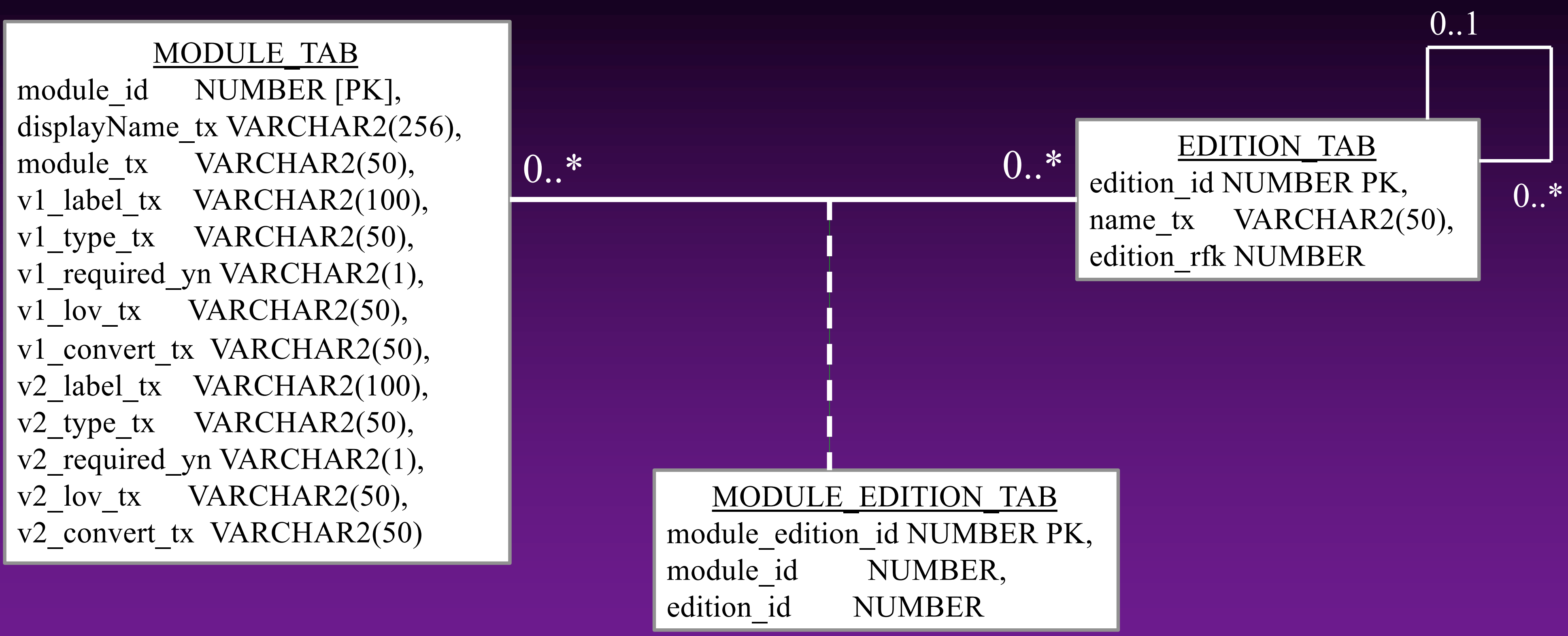
Homegrown Versioning (2)

- Solution:
  1. The system must store a list of registered modules in the repository.
  2. Each module must satisfy the following conditions:
     - Take up to 5 input parameters (some optional, some mandatory).
     - Return formatted CLOB as an output
  3. The system has a notion of editions that can be associated with the module.
  4. The system uses the default edition.
  5. Each user may have access to different editions instead of the default.

Sample

```
CREATE FUNCTION f_getEmp_CL (i_job_tx VARCHAR2, i_hiredate_dt DATE:=NULL)
RETURN CLOB
IS
    v_out_cl CLOB;
    PROCEDURE p_add(pi_tx VARCHAR2) IS BEGIN
        dbms_lob.writeappend(v_out_cl,length(pi_tx),pi_tx);
    END;
BEGIN
    dbms_lob.createtemporary(v_out_cl,true,dbms_lob.call);
    p_add('<html><table>');
    FOR c IN (SELECT '<tr>'||'<td>'||empno||'</td>'||
                     '<td>'||ename||'</td>'||'</tr>' row_tx
              FROM emp
              WHERE job = i_job_tx
              AND hiredate >= NVL(i_hiredate_dt,add_months(sysdate,-36))
              ) LOOP
        p_add(c.row_tx);
    END LOOP;
    p_add('</table></html>');
    RETURN v_out_cl;
END;
```

Data Model



MODULE_TAB
module_id NUMBER [PK],
displayName_tx VARCHAR2(256),
module_tx VARCHAR2(50),
v1_label_tx VARCHAR2(100),
v1_type_tx VARCHAR2(50),
v1_required_yn VARCHAR2(1),
v1_lov_tx VARCHAR2(50),
v1_convert_tx VARCHAR2(50),
v2_label_tx VARCHAR2(100),
v2_type_tx VARCHAR2(50),
v2_required_yn VARCHAR2(1),
v2_lov_tx VARCHAR2(50),
v2_convert_tx VARCHAR2(50)

EDITION_TAB
edition_id NUMBER PK,
name_tx VARCHAR2(50),
edition_rfk NUMBER

MODULE_EDITION_TAB
module_edition_id NUMBER PK,
module_id NUMBER,
edition_id NUMBER

Data Repository

```
-- register modules
INSERT INTO module_tab (module_id,displayName_tx,module_tx,
           v1_label_tx, v1_type_tx, v1_required_yn,
           v2_label_tx, v2_type_tx, v2_required_yn, v2_convert_tx)
VALUES (100, 'Filter Employees by Job/Hire Date', 'f_getEmp_cl',
        'Job','TEXT','Y','Hire Date',
        'DATE','N','TO_DATE(v2_tx,''YYYYMMDD'')');
INSERT INTO module_tab (module_id,displayName_tx,module_tx,
                                                 v1_label_tx, v1_type_tx, v1_required_yn)
VALUES (101, 'Filter Employees by Job', 'f_getEmp_cl','Job','TEXT','Y');

-- create two editions
INSERT INTO edition_tab (edition_id, name_tx, edition_rfk) VALUES (10, 'Default', null);
INSERT INTO edition_tab (edition_id, name_tx, edition_rfk) VALUES (11, 'New Edition',10);
-- associate modules with editions
INSERT INTO module_edition (me_id,module_id,edition_id) values (20,100,10);
INSERT INTO module_edition (me_id,module_id,edition_id) values (21,101,11);
-- associate users with editions
INSERT INTO user_edition (ue_id, user_tx, edition_id) values (30,'HR',10);
INSERT INTO user_edition (ue_id, user_tx, edition_id) values (31,'OE',11);
```

Query Repository

```
SQL> SELECT m.module_id, m.displayname_tx
     FROM module_tab m,
          module_edition me
     WHERE m.module_id = me.module_id
     AND   me.edition_id IN (SELECT edition_id
                             FROM user_edition
                             WHERE user_tx = 'HR');
 MODULE_ID DISPLAYNAME_TX
---------- ----------------------------------------
       100 Filter Employees by Job/Hire Date

SQL> SELECT m.module_id, m.displayname_tx
     FROM module_tab m,
          module_edition me
     WHERE m.module_id = me.module_id
     AND   me.edition_id in (SELECT edition_id
                             FROM user_edition
                             WHERE user_tx = 'OE');
 MODULE_ID DISPLAYNAME_TX
---------- ----------------------------------------
       101 Filter Employees by Job
```

Wrapper

```
CREATE OR REPLACE FUNCTION f_wrapper_cl (i_module_id NUMBER,
                               i_v1_tx VARCHAR2:=null,
                               i_v2_tx VARCHAR2:=null)
RETURN CLOB
IS
    v_out_cl CLOB;
    v_sql_tx VARCHAR2(32767);
    v_rec module_tab%ROWTYPE;
BEGIN
    SELECT * INTO v_rec FROM module_tab WHERE module_id=i_module_id;
    IF v_rec.v1_label_tx IS NOT NULL THEN
        v_sql_tx:=nvl(v_rec.v1_convert_tx,'v1_tx');
    END IF;
    IF v_rec.v2_label_tx IS NOT NULL THEN
        v_sql_tx:=v_sql_tx||','||nvl(v_rec.v2_convert_tx,'v2_tx');
    END IF;

    v_sql_tx:='DECLARE '||chr(10)||
              ' v1_tx VARCHAR2(32767):=:1;'||CHR(10)||
              ' v2_tx VARCHAR2(32767):=:2;'||CHR(10)||
              'BEGIN '||CHR(10)||
              ' :out:='||v_rec.module_tx||'('||v_sql_tx||');'||CHR(10)||
              'END;';
    EXECUTE IMMEDIATE v_sql_tx USING i_v1_tx,i_v2_tx, OUT v_out_cl;
    RETURN v_out_cl;
END;
```

Usage

- Safe solution:
  - All user-enterable data is passed via bind variables.
  - All structural elements are selected from the repository.

```
SQL> SELECT f_wrapper_cl (100,'PRESIDENT','19001010')
  2> FROM DUAL;
F_WRAPPER_CL(100,'PRESIDENT','19001010')
----------------------------------------------------------------
<html><table><tr><td>7839</td><td>KING</td></tr></table></html>
```

Edition-Based Redefinition (EBR)

Real Life

- Major challenge:
  - Incremental roll-over, i.e. co-existence of old and new code base

- Manual solutions = nightmare!

- Alternative (starting Oracle 11gR2) – Edition-Based Redefinition

What is EBR?

- Enabling editions:
  - Done for a specified user:
  - Editionable objects are uniquely identified by name and edition
    - i.e. multiple versions of the same object at the same time.
  - Editions are shared across the database.
    - Default =ORA$BASE
    - All other editions are children/[grand]children of ORA$BASE
    - Editions are linked in a chain (ORA$BASE – Edition 1 – Edition 2).
  - One current edition in the session
    - … but you can change it with ALTER SESSION.
  - For the new session – the current edition is
    - … either the default [ALTER DATABASE DEFAULT EDITION…]
    - … or explicitly specified in the connection string.
  - Special editioning views with cross-edition triggers
    - Fire different code in Parent/Child edition

What objects are editionable?

- As of Oracle 12c:
  - SYNONYM
  - VIEW
  - SQL translation profile
  - All PL/SQL object types:
    - FUNCTION
    - LIBRARY
    - PACKAGE and PACKAGE BODY
    - PROCEDURE
    - TRIGGER
    - TYPE and TYPE BODY

But…

- Key restriction:
  - Non-editioned objects cannot depend upon editioned ones

Key Improvements in Oracle 12c (1)

- New clauses for materialized views and virtual columns

```
-- [ evaluation_edition_clause ]
EVALUATE USING { CURRENT EDITION | EDITION edition | NULL EDITION }

-- [ unusable_before_clause ]
UNUSABLE BEFORE { CURRENT EDITION | EDITION edition }

-- [ unusable_beginning_clause ]
UNUSABLE BEGINNING WITH {CURRENT EDITION | EDITION edition| NULL EDITION}
```

- Changed granularity of what can/cannot be editioned
  - 11gR2: Editioned-enabled schema means that ALL types/objects become editioned.
  - 12c: You can edition-enable only some types of objects:

```
ALTER USER user ENABLE EDITIONS [ FOR type [, type ]... ]
```

Key Improvements in Oracle 12c (2)

- You can explicitly make potentially editionable objects NON-editionable:
  - … for example, to build function-based indexes

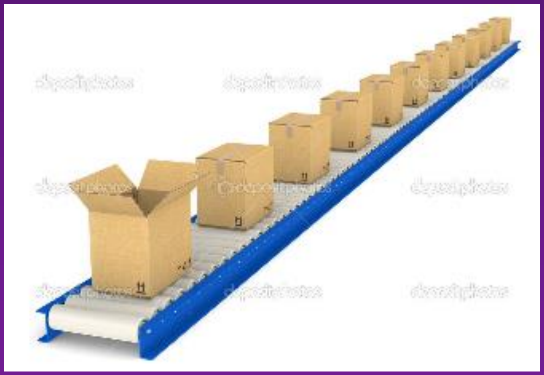
```
SQL> CREATE USER ebr1 IDENTIFIED BY ebr1
  2  DEFAULT TABLESPACE USERS TEMPORARY TABLESPACE TEMP
  3  ENABLE EDITIONS; -- enable editions either directly/via ALTER USER
User created.

SQL> CREATE NONEDITIONABLE FUNCTION ebr1.f_toDate_udf
  2      (i_tx VARCHAR2, i_format_tx VARCHAR2:='YYYYMMDD')
...
 15  /
Function created.
SQL> CREATE INDEX ebr1.test_idx ON ebr1.test_tab(f_toDate_udf(ddl_tx));
Index created.
```

Impact for Code Management?

- You can
  - Create logical packaging of server-side code base
    - … i.e. clearly separate different code groups
  - Have multiple versions of the code in the database <u>at the same time</u>
    - … i.e. you can compare behavior/performance exactly in the same conditions (data/hardware).
  - Quickly switch between versions without any installation required
    - … i.e. shorten response time in an emergency.
- You cannot
  - Easily edition structural elements (i.e. tables/indexes etc.)
    - … although you can play synonym games which ARE editionable.
  - Easily edition data
    - … although you can introduce temporary data visibility rules.

Production Environments
and
Performance-Related Code Management



Deployment Architectural Flaw

- Condition:
  - Code is constantly moving between DEV/TEST/PROD
  - DEV <> TEST <> PROD!
- Problem:
  - How can you be sure that functionally correct changes don't negatively impact performance???
    - ... Well, you don't ☹

Areas of Interest

- ◆ Hardware/Networking
  - ➢ … because even the smallest firewall setting can be disastrous.
- ◆ Data volume
  - ➢ … because PROD is ALWAYS larger than TEST.
- ◆ User volume
  - ➢ … because Oracle has lots of shared resources, you can encounter unexpected bottlenecks.

Most Important Deployment Question

- You HAVE to have a clear answer BEFOREHAND:
  - If anything goes wrong, how do you fall back?
- Why?
  - More time to figure this out – more losses/more bugs
    - … and more stress on everybody
  - Management should understand costs/risks associated with code versioning.
    - … otherwise you get into continuous deployment nightmare (aka Agile Development ☺)

Approaches

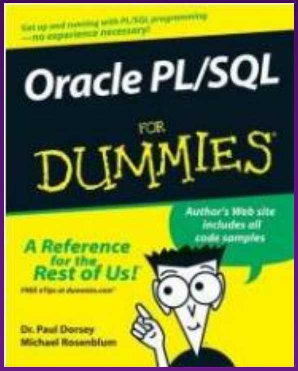
- 1. Entire system versioning
  - Recovery is based on complete backup of the system
    - … preferably on separate hardware
- 2. Limited-scope code modification
  - Recovery is based on knowing exactly what changed
  - … preferably via metadata-based form (EBR, repositories)
- 3. Everything else
  - … sorry, no idea what to do ☹

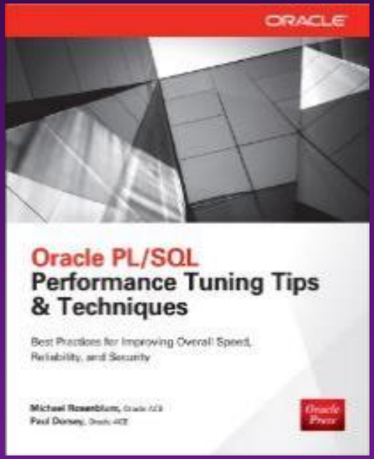
Summary

- Fixing problems of existing systems is one of the main development tasks in any organization
  - … so, you have to think about code management from the very beginning.
- Logical notion of "editions" helps thinking about code deployments
  - … whether you use EBR or not.
- Some concepts are common:
  - Micro-managing your changes <> good code versioning
  - Performance problems are resolved only when they are deployed to PROD and there are no side effects.
  - Successful code versioning leads to better overall system performance.
  - The best way to validate performance is to have old/new code coexist at the same time [hint: EBR!]

Contact Information

- Michael Rosenblum – mrosenblum@dulcian.com
- Dulcian, Inc. website - www.dulcian.com
- Blog: wonderingmisha.blogspot.com





Available NOW:

Oracle PL/SQL Performance Tuning Tips & Techniques