A New View of Database Views: Through the Looking-Glass

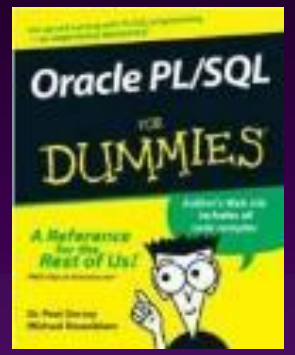
Michael Rosenblum

www.dulcian.com

DULCIAN INC.

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

Views???

- Common train of thought:
  - Stored SQL queries
  - Nothing to worry about
    - Reasonably transparent and stackable
    - Minimal impact
  - Have been around forever…..
    - (Usually implies)…and have never changed (???)

DULCIAN INC.

Views!!!

- "Thick database" approach:
  - Views can serve as an isolation layer
    - UI should never talk to tables!
      - Views can be easily changed.
      - Views are editionable (EBR).
    - Isolation can be bi-directional.
  - Views are complex.
    - Lots of new functionality in every version
    - Can impact Cost Based Optimizer (CBO) decisions
    - Side-effects and non-trivial impact

DULCIAN INC.

Outline

- To be discussed:
  - De-normalized views with INSTEAD OF triggers
  - Function-based views
  - Special cases:
    - Compound triggers
    - Cross-edition views
    - Various performance considerations ☺
- Will not be discussed:
  - Object views
  - XMLType views
  - Materialized views
  - JSON-Relational Duality Views

Views: Functionality & Features

DULCIAN INC.

The Idea

- View is (IMHO!):
  - Data set, broadly defined at design-time and formed at run-time.
  - The definition may mutate via:
    - Parameterized conditions
    - Dynamic SQL
  - This data set may accept DMLs
    - Only on a per-row basis (no statement-level actions)
    - Directly or via associated PL/SQL program units
    - Set is not persistent and may need to be rebuilt to accept DMLs.

The Reasoning

- Logical denormalization is great:
  - Less coding in the UI
  - Transparent system design
  - Long-term protection against data model changes
  - Simplified testing

Basics

Basic Code ☺

```
create or replace view scott.v_emp
as
select e.empno,
       e.ename,
       e.deptno,
       d.dname,
       e.sal
from scott.emp e,
     scott.dept d
where e.deptno=d.deptno
```

Direct DML…Sometimes

```
select *
from dba_updatable_columns
where table_name = 'V_EMP'
```

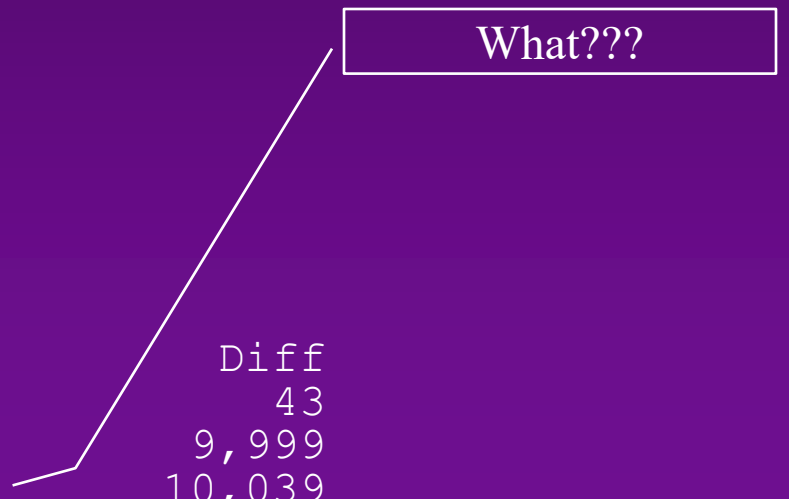
```
COLUMN_NAME     UPDATABLE       INSERTABLE     DELETABLE
--------------- --------------- -------------- ---------
EMPNO           YES             YES            YES
ENAME           YES             YES            YES
DEPTNO          YES             YES            YES
DNAME           NO              NO             NO
SAL             YES             YES            YES
```

INSTEAD-OF – Functionality

```
create or replace trigger scott.v_emp_iu
instead of update on scott.v_emp
begin
  if :new.sal!=:old.sal then
    if sys_context ('userenv','CLIENT_INFO')='SalMaint'
    then
      update emp
      set sal = :new.sal
      where empno = :old.empno;
    else
      raise_application_error
           (-20001,'Not enough privileges');
    end if;
 else
    update emp
    set ename = :new.ename
    where empno = :old.empno;
  end if;
end;
```

INSTEAD-OF – Extra Cost

```
exec scott.runStats_pkg.rs_start;
alter trigger scott.v_emp_iu disable;
begin
    for i in 1..10000 loop
        update scott.v_emp set ename=ename where empno = 7782;
    end loop;
end;
/
exec scott.runStats_pkg.rs_middle;

alter trigger scott.v_emp_iu enable;
begin
    for i in 1..10000 loop
        update scott.v_emp set ename=ename where empno = 7782;
    end loop;
end;
/
exec scott.runStats_pkg.rs_stop;

Run1 ran in 35 cpu hsecs
Run2 ran in 77 cpu hsecs
run 1 ran in 45.45% of the time

Name                                  Run1        Run2        Diff
STAT...CPU used by this session         36          79          43
STAT...execute count                10,037      20,036       9,999
STAT...session logical reads        40,441      50,480      10,039
```



DMLs Against Views

DML Implementation

- UPDATE on view = SELECT + UPDATE
- DELETE on view = SELECT + DELETE
- INSERT on view = Only INSERT

Test Case (1)

```
CREATE TYPE test_tab_ot AS OBJECT (owner_tx VARCHAR2(30),
                                   name_tx VARCHAR2(30),
                                   object_id NUMBER,
                                   type_tx VARCHAR2(30));
CREATE TYPE test_tab_nt IS TABLE OF test_tab_ot;

CREATE FUNCTION f_searchTestTab_tt (i_type_tx VARCHAR2) RETURN test_tab_nt
IS
    v_out_tt test_tab_nt;
begin
    SELECT test_tab_ot(owner, object_name, object_id, object_type)
    BULK COLLECT INTO v_out_tt
    FROM test_tab
    WHERE object_type = i_type_tx;
    dbms_output.put_line('Inside f_searchTestTab_tt:'||v_out_tt.count);
    RETURN v_out_tt;
END;
```

Test Case (2)

```
create or replace view v_search_table as
select *
from table(f_searchtesttab_tt('TABLE'));

create or replace trigger v_search_table_iiud
instead of insert or update or delete on v_search_table
begin
    if inserting then
      dbms_output.put_line('Insert');
    elsif updating then
      dbms_output.put_line('Update');
    elsif deleting then
      dbms_output.put_line(`Delete');
    end if;
end;
```

DML Check

```
SQL> INSERT INTO v_search_table (object_id, name_tx)
  2   VALUES (-1,'A');
Insert
1 row created.

SQL> UPDATE v_search_table SET name_tx = 'Test`
  2   WHERE object_id = 5;
Inside f_searchTestTab_tt:1571
Update
1 row updated.

SQL> DELETE FROM v_search_table WHERE object_id = 5;
Inside f_searchTestTab_tt:1571
Delete
1 row deleted.
```

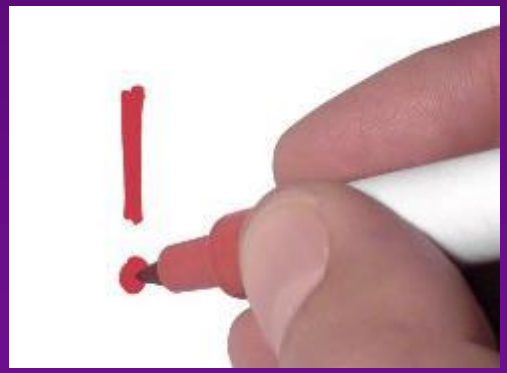
Function is not fired!

Process all to update one?

DULCIAN INC.

Important!

- To fire INSTEAD OF UPDATE/DELETE, Oracle must locate row first.
- Oracle must have the whole resulting set available to filter it
  - … but you can cheat ☺



Parameterized View (1)

```
CREATE PACKAGE global_pkg IS
    v_object_id NUMBER;
END;

CREATE FUNCTION f_searchTestTab_tt (i_type_tx varchar2) RETURN test_tab_nt is
    v_out_tt test_tab_nt;
BEGIN
    IF global_pkg.v_object_id IS NULL THEN
        SELECT test_tab_ot(owner, object_name, object_id, object_type)
        BULK COLLECT INTO v_out_tt
        FROM test_tab
        WHERE object_type = i_type_tx;
    ELSE
        SELECT test_tab_ot(owner, object_name, object_id, object_type)
        BULK COLLECT INTO v_out_tt
        FROM test_tab
        WHERE object_id = global_pkg.v_object_id;
    END IF;
    dbms_output.put_line
       ('Inside f_searchTestTab_tt:'||v_out_tt.count);
    RETURN v_out_tt;
END;
```

Global parameter

Usage (if needed)

Parameterized View (2)

```
SQL> BEGIN global_pkg.v_object_id:=5; END;
  2  /
SQL> UPDATE v_search_table SET name_tx = 'Test';
Inside f_searchTestTab_tt:1
Update
1 row updated.

SQL> DELETE FROM v_search_table;
Inside f_searchTestTab_tt:1
Delete
1 row deleted.
```

DULCIAN INC.

Compound Triggers on Views

DULCIAN INC.

Alternative to INSTEAD OF

- You can also create a COMPOUND trigger instead of an INSTEAD OF trigger ☺
  - Trigger with common program area
- Impact
  - Pro:
    - Common program area – possible to share code and global variables ➔ simulating statement-level BEFORE event (sorry, there is no way to simulate an AFTER event)
  - Con:
    - Not very stable in 11g

Monitoring tool

```
CREATE OR REPLACE PACKAGE counter_pkg IS
    v_nr NUMBER:=0;
    PROCEDURE p_check;
    procedure p_up;
END;
CREATE OR REPLACE PACKAGE BODY counter_pkg IS
    PROCEDURE p_check is
    BEGIN
        dbms_output.put_line('Fired:'||counter_pkg.v_nr);
        counter_pkg.v_nr:=0;
    END;
    procedure p_up is
    begin
        counter_pkg.v_nr:=counter_pkg.v_nr+1;
    end;
END;

create or replace function f_securityCheck_yn return varchar2 is
begin
    counter_pkg.p_up;
    if sys_context ('USERENV', 'CLIENT_INFO') ='SalMaint' then
        return 'Y';
    else
        return 'N';
    end if;
end;

create function f_change_nr (i_nr number) return number is
begin
    counter_pkg.p_up;
    return i_nr+1;
end;
```

Performance impact (1)

```
CREATE OR REPLACE TRIGGER v_search_table_IIUD
INSTEAD OF INSERT OR UPDATE OR DELETE ON v_search_table
BEGIN
    if f_securityCheck_yn  = 'Y' then
        IF INSERTING THEN       dbms_output.put_line('Insert');
        ELSIF UPDATING THEN     dbms_output.put_line('Update');
        ELSIF DELETING THEN     dbms_output.put_line('Delete');
        END IF;
    end if;
END;

create or replace trigger v_search_table_IIUD_comp
for insert or update or delete on v_search_table
COMPOUND TRIGGER
    v_check_yn varchar2(1);

    instead of each row is
    begin
        if v_check_yn is null then
            v_check_yn:=f_securityCheck_yn;
        end if;
        if v_check_yn = 'Y' then
            IF INSERTING THEN       dbms_output.put_line('Insert');
            ELSIF UPDATING THEN     dbms_output.put_line('Update');
            ELSIF DELETING THEN     dbms_output.put_line('Delete');
            END IF;
        end if;
    end instead of each row;
end;
```

Common area

Performance impact (2)

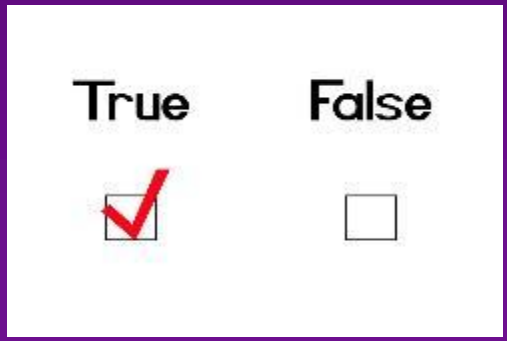
```
SQL> alter trigger v_search_table_IIUD_COMP disable;
SQL> alter trigger v_search_table_IIUD enable;
SQL> update v_search_table set name_tx = 'Test';
Inside f_searchTestTab_tt:1571
1571 rows updated.
SQL> exec counter_pkg.p_check;
Fired:1571

SQL> alter trigger v_search_table_IIUD_COMP enable;
SQL> alter trigger v_search_table_IIUD disable;
SQL> update v_search_table set name_tx = 'Test';
Inside f_searchTestTab_tt:1571
1571 rows updated.
SQL> exec counter_pkg.p_check;
Fired:1

SQL> exec dbms_application_info.set_client_info('SalMaint');
SQL> update v_search_table set name_tx = 'Test';
Inside f_searchTestTab_tt:1571
Update
Update
....
1571 rows updated.
SQL> exec counter_pkg.p_check;
Fired:1
```

Only one call!

Dangers of Logical Primary Keys

The Real Story

- Synthetic primary keys on views
  - Sometimes required by front-end environments (to uniquely identify the row in the cache)
  - Extremely dangerous if blindly used for INSERT/UPDATE/DELETE



Test Case

```
CREATE OR REPLACE VIEW v_test_tab AS
SELECT 'Main|'||object_id pk_tx,
       a.*
FROM test_tab_main a
UNION ALL
SELECT 'Other|'||object_id pk_tx,
       a.*
FROM test_tab_other a;
```

Performance Impact

```
SQL> set autotrace traceonly explain
SQL> UPDATE v_test_tab
  2  SET object_name='Test'
  3  WHERE pk_tx='Main|1';

1 row updated.
Execution Plan
----------------------------------------------------------

Plan hash value: 3568876726
-------------------------------------------------------------------------------
|Id| Operation                | Name                | Rows  | Bytes | Cost (%CPU)|
-------------------------------------------------------------------------------
| 0| UPDATE STATEMENT         |                     |   500 | 45500 |   236   (1)|
| 1|  UPDATE                  | V_TEST_TAB          |       |       |            |
| 2|   VIEW                   | V_TEST_TAB          |   500 | 45500 |   236   (1)|
| 3|    UNION-ALL             |                     |       |       |            |
|*4|     TABLE ACCESS FULL    | TEST_TAB_MAIN       |    15 |   330 |     9   (0)|
|*5|     TABLE ACCESS FULL    | TEST_TAB_OTHER      |   485 | 14550 |   227   (1)|
-------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   4 - filter('Main|'||TO_CHAR("OBJECT_ID")='Main|1')
   5 - filter('Other|'||TO_CHAR("OBJECT_ID")='Main|1')
```

Full-table scans!

Optimization of UPDATEs

DULCIAN INC.

Resource Issues

- Problem
  - INSTEAD-OF UPDATE triggers often reference **all** columns of the underlying tables
    - … instead of trying to figure out what has been changed.
  - UNDO is generated for **all** columns mentioned in the UPDATE statement
    - … regardless of whether they were changed or not
    - … which causes major I/O overhead.
- Alternative:
  - Use Dynamic SQL to modify only changed columns
    - … i.e. trade CPU utilization for better I/O!

Regular Trigger

```
create or replace trigger v_test_tab_iu
instead of update on v_test_tab
begin
    if :new.object_type='TABLE' then
        update test_tab_main
        set owner=:new.owner, object_name=:new.object_name,
          subobject_name=:new.subobject_name,
          object_type=:new.object_type,
          created=:new.created, last_ddl_time=:new.last_ddl_time,
          timestamp=:new.timestamp, status=:new.status,
          temporary=:new.temporary, generated=:new.generated,
          secondary=:new.secondary, namespace=:new.namespace,
          edition_name=:new.edition_name, sharing=:new.sharing,
          editionable=:new.editionable,
          oracle_maintained=:new.oracle_maintained
        where object_id=:old.object_id;
    else
        update test_tab_other
        set << the same list of columns as above >>
        where object_id=:old.object_id;
    end if;
end;
```

Dynamic SQL Trigger (1)

```
create or replace trigger v_test_tab_dynamic_iu
instead of update on v_test_tab
declare
  v_main_rec test_tab_main%rowtype;
  v_mainupdate_tx varchar2(32767);
  v_other_rec test_tab_other%rowtype;
  v_otherupdate_tx varchar2(32767);
begin
  if :new.object_type='table' then

    -- compare old/new for each attribute
    if :old.object_name is null and :new.object_name is not null
    or :old.object_name is not null and :new.object_name is null
    or :old.object_name!=:new.object_name then
      v_main_rec.object_name:=:new.object_name;
      v_mainupdate_tx:=v_mainupdate_tx||
        case
          when v_mainupdate_tx is not null then ','
          else null
        end||chr(10)||
        '  object_name=v_rec.object_name';
    end if;
    ...
```

Store new value (if changed)

Dynamic SQL Trigger (2)

```
    v_main_rec.object_id:=:old.object_id;
    v_mainupdate_tx:=
      'declare '||chr(10)||
      '   v_rec test_tab_main%rowtype:=:1;'||chr(10)||
      'begin '||chr(10)||
      '  update test_tab_main ' ||chr(10)||
      '  set '||v_mainupdate_tx||chr(10)||
      '  where object_id=v_rec.object_id;'||chr(10)||
      'end;';
    execute immediate v_mainupdate_tx using v_main_rec;

    else
       <<  the same logic as above for test_tab_other >>
    end if;
end;
```

Performance Tradeoff

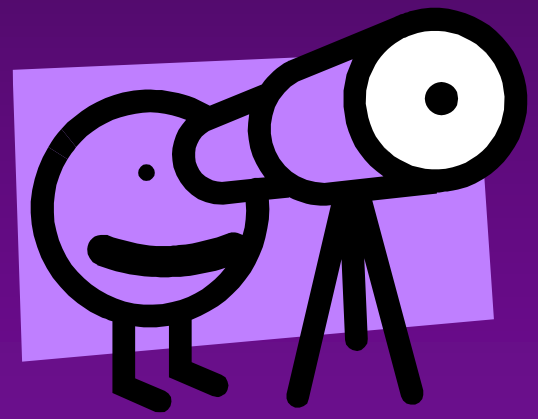
```
SQL> exec  runstats_pkg.rs_start;
    << enable Dynamic trigger>>
SQL> begin  2   for i in 1..10000 loop
  3     UPDATE v_test_tab SET object_name='Test'||i WHERE object_id = 5;
  4   end loop;
  5  END;
  6  /
SQL> exec  runstats_pkg.rs_middle;
  << enable regular trigger and rerun the same logic>>
SQL> exec  runstats_pkg.rs_stop;
Run1 ran in 390 cpu hsecs
Run2 ran in 496 cpu hsecs
run 1 ran in 78.63% of the time
Name                                    Run1        Run2        Diff
STAT...recursive calls                20,329      30,244       9,915
STAT...execute count                  20,102      30,095       9,993
STAT...undo change vector size     2,495,476     938,552  -1,556,924
STAT...redo size                   5,820,096   2,772,332  -3,047,764
Run1 latches total versus runs -- difference and pct
Run1        Run2        Diff       Pct
136,486     184,679      48,193     73.90%
```

More CPU time

Much less I/O

DULCIAN INC.

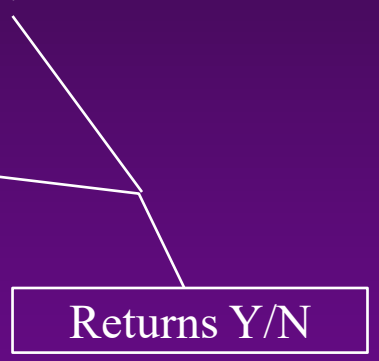
PL/SQL Functions and Views



Setup

- Functions can be used as columns in views:

```
CREATE FUNCTION f_isEastCoast_yn (i_deptno NUMBER)
RETURN VARCHAR2
IS
    v_out_yn varchar2(1);
BEGIN
    SELECT CASE WHEN loc IN ('NEW YORK', 'BOSTON') THEN 'Y'
                ELSE 'N'
           END
    INTO v_out_yn
    FROM dept
    WHERE deptno = i_deptno;
    RETURN v_out_yn;
END;

CREATE VIEW v_emp_loc
AS
SELECT emp.empno, emp.ename, emp.deptno,
         f_isEastCoast_yn(deptno) eastCoast_yn
FROM emp;
```



Problem

- Default behavior:

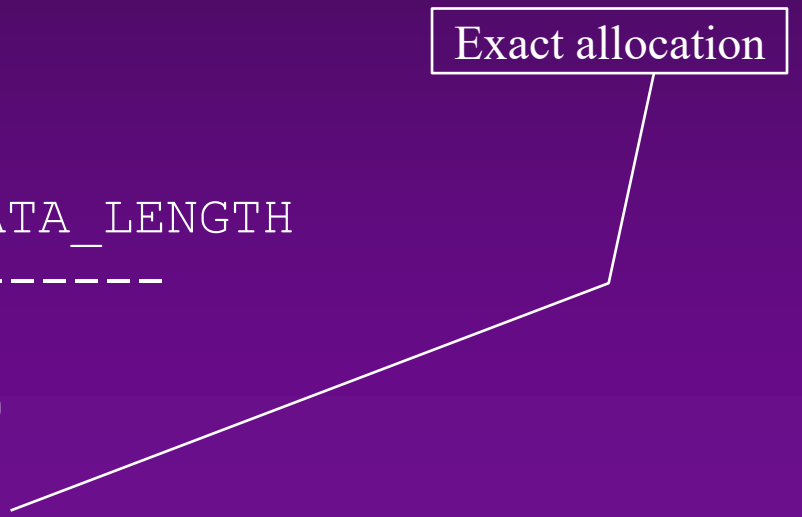
```
SQL> SELECT column_name, data_type, data_length
2 FROM user_tab_columns
3 WHERE table_name = 'V_EMP_LOC'
4 ORDER by column_id;

COLUMN_NAME     DATA_TYPE       DATA_LENGTH
--------------- --------------- -----------
EMPNO           NUMBER          22
ENAME           VARCHAR2        10
DEPTNO          NUMBER          22
EASTCOAST_YN    VARCHAR2        4000
```

Over-allocation can impact front-end and middle-tier
that may read the Oracle Data Dictionary.
In the case of materialized views – storage over-allocation

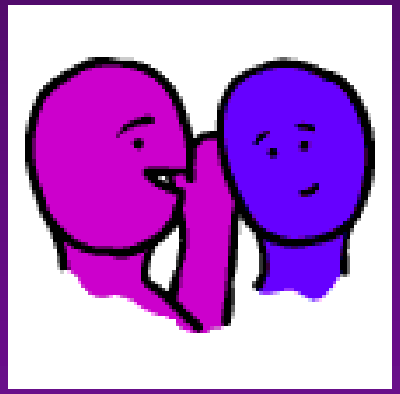
Solution

```
CREATE OR REPLACE VIEW v_emp_loc AS
SELECT emp.empno, emp.ename, emp.deptno,
   cast(f_isEastCoast_yn(deptno) as VARCHAR2(1)) eastCoast_yn
FROM emp
```

```
SQL> SELECT column_name, data_type, data_length
2 FROM user_tab_columns
3 WHERE table_name = 'V_EMP_LOC'
4 ORDER by column_id;

COLUMN_NAME          DATA_TYPE           DATA_LENGTH
-------------- --------------- -----------
EMPNO                NUMBER              22
ENAME                VARCHAR2            10
DEPTNO               NUMBER              22
EASTCOAST_YN         VARCHAR2            1
```



Hints and Views



DULCIAN INC.

Problem

- Yes, there are cases when hints are needed.
- Hints are easy to add if you have a single-level query.
- If you are using views:
  - Straightforward if you are lucky and can change underlying views. ☺
  - Doable otherwise, but may be tricky.

Solution #1

- Explicit naming of SELECT statements (20 char limit):

```
create or replace view scott.v_emp
as
select /*+ QB_NAME(V_EMP_Main) */
       e.empno,
       e.ename,
       e.deptno,
       d.dname,
       e.sal
from scott.emp e,
     scott.dept d
where e.deptno=d.deptno
```

Solution #1 - Proof

```
SQL> select /*+ FULL(@V_EMP_Main e) */ *
  2  from v_emp
  3  where empno=7369;
     EMPNO ENAME          DEPTNO DNAME                SAL
---------- ---------- ---------- -------------- ----------
      7369 SMITH              20 RESEARCH               800
Execution Plan
----------------------------------------------------------
Plan hash value: 3625962092
-------------------------------------------------------------------------
| Id  | Operation                     | Name    | Rows  | Bytes |
-------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |         |     1 |    30 |
|   1 |  NESTED LOOPS                 |         |     1 |    30 |
|   2 |   NESTED LOOPS                |         |     1 |    30 |
|*  3 |    TABLE ACCESS FULL          | EMP     |     1 |    17 |
|*  4 |    INDEX UNIQUE SCAN          | PK_DEPT |     1 |       |
|   5 |   TABLE ACCESS BY INDEX ROWID| DEPT    |     1 |    13 |
-------------------------------------------------------------------------
```

Solution #2

- Find Oracle-generated object alias:

```
SQL> explain plan for select * from v_emp where empno=7369;
Explained.
SQL> select * from table(dbms_xplan.display(null,null,'All'));
-----------------------------------------------------------------
| Id  | Operation                     | Name    | Rows  | Bytes |
-----------------------------------------------------------------
|   0 | SELECT STATEMENT              |         |     1 |    30 |
|   1 |  NESTED LOOPS                 |         |     1 |    30 |
|   2 |   TABLE ACCESS BY INDEX ROWID| EMP     |     1 |    17 |
|*  3 |    INDEX UNIQUE SCAN          | PK_EMP  |     1 |       |
|   4 |   TABLE ACCESS BY INDEX ROWID| DEPT    |     1 |    13 |
|*  5 |    INDEX UNIQUE SCAN          | PK_DEPT |     1 |       |
Query Block Name / Object Alias (identified by operation id):
-------------------------------------------------------------
   1 - SEL$F5BB74E1
   2 - SEL$F5BB74E1 / E@SEL$2
   3 - SEL$F5BB74E1 / E@SEL$2
   4 - SEL$F5BB74E1 / D@SEL$2
   5 - SEL$F5BB74E1 / D@SEL$2
Predicate Information (identified by operation id):
---------------------------------------------------
   3 - access("E"."EMPNO"=7369)
   5 - access("E"."DEPTNO"="D"."DEPTNO")
```

Solution #2 - Proof

```
SQL> select /*+ FULL(@SEL$2 e) */ *
  2  from v_emp
  3  where empno=7369;

     EMPNO ENAME          DEPTNO DNAME              SAL
---------- ---------- ---------- -------------- ----------
      7369 SMITH              20 RESEARCH              800

Execution Plan
----------------------------------------------------------
Plan hash value: 3625962092
-------------------------------------------------------------------
| Id  | Operation                    | Name    | Rows  | Bytes |
-------------------------------------------------------------------
|   0 | SELECT STATEMENT             |         |     1 |    30 |
|   1 |  NESTED LOOPS                |         |     1 |    30 |
|   2 |   NESTED LOOPS               |         |     1 |    30 |
|*  3 |    TABLE ACCESS FULL         | EMP     |     1 |    17 |
|*  4 |    INDEX UNIQUE SCAN         | PK_DEPT |     1 |       |
|   5 |   TABLE ACCESS BY INDEX ROWID| DEPT    |     1 |    13 |
-------------------------------------------------------------------
```

DULCIAN INC.

Keep in mind…

- Naming all subqueries costs you nothing, but makes production support much easier.
- Warnings:
  - Names may be ignored:
    - If two or more query blocks have the same name
    - If the same query block is hinted twice with different name
  - Beware of query transformation!

Editioning Views

DULCIAN INC.

Concept

- Editioning view (Edition-Based Redefition):
  - Can be created only for edition-enabled users
  - Can have multiple versions
  - Lots of restrictions (no joins, conditions, grouping etc.) ~ more or less a synonym to the table

```
-- Parent Edition
create or replace editioning view scott.v_emp_ed
as select empno, ename, hiredate from scott.emp;

-- Child Edition (new column TIMESTAMP instead of DATE)
create or replace editioning view scott.v_emp_ed
as select empno, ename, hiredate_ts from scott.emp;
```

New Functionality

- Editioning views may have the same triggers as tables

```
-- Child edition: only new data
create or replace trigger v_emp_ed_bu
before update on v_emp_ed for each row
Begin
    if :new.hiredate_ts <= to_timestamp('20150101','YYYYMMDD')
    then
       raise_application_error(-20001,'Data is too old');
    end if;
end;

-- Parent edition: only old data
create or replace trigger v_emp_ed_bu
before update on v_emp_ed for each row
Begin
    if :new.hiredate > to_date('20150101','YYYYMMDD')
    then
       raise_application_error(-20001,'Data is too new');
    end if;
end;
```

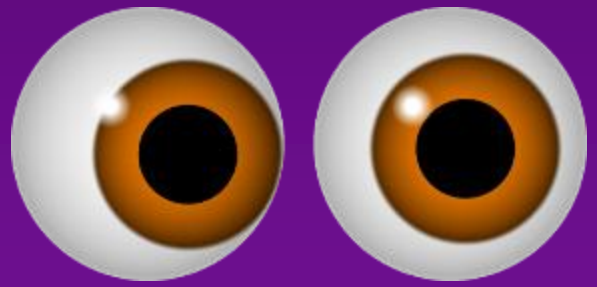
DULCIAN INC.

Syntax News in 12c+

DULCIAN INC.

Views

- Multiple triggers of the same type:
  - FOLLOWS <trigger> clause
- Triggers can be created in DISABLED state.
- Create invalid view (for later recompilation):
  - FORCE clause



Views with Constraints (1)

- Making view non-updatable
  - WITH READ ONLY clause

```
create or replace view scott.v_emp as
select e.empno,e.ename, e.deptno, d.dname, e.sal
from scott.emp e,
     scott.dept d
where e.deptno=d.deptno
with read only constraint V_EMP_RO
```

```
SQL> select constraint_name, constraint_type, table_name
  2  from user_constraints
  3  where table_name = 'V_EMP';

CONSTRAINT_NAME   CONSTRAINT_TYPE TABLE_NAME
----------------- --------------- --------------------
V_EMP_RO          O               V_EMP
```

Views with Constraints (2)

- WITH CHECK OPTION clause ~ Check constraint
  - Insert/Update for single-table views
  - Update is validated for multi-table views
    - Insert causes "ORA-01733: virtual column not allowed here"
    - Delete blocked if you have self-join

```
create or replace view scott.v_emp as
select e.empno, e.ename, e.deptno, d.dname, e.sal
from scott.emp e,
     scott.dept d
where e.deptno=d.deptno
and   d.deptno !=30
with check option constraint V_EMP_C

SQL> select constraint_name, constraint_type, table_name
  2  from user_constraints
  3  where table_name = 'V_EMP';

CONSTRAINT_NAME    CONSTRAINT_TYPE TABLE_NAME SEARCH_CONDITION
---------------- --------------- ---------- ----------------
V_EMP_C            V               V_EMP      NULL
```

You cannot see condition ☹

DULCIAN INC.

SQL Text Expansion (1)

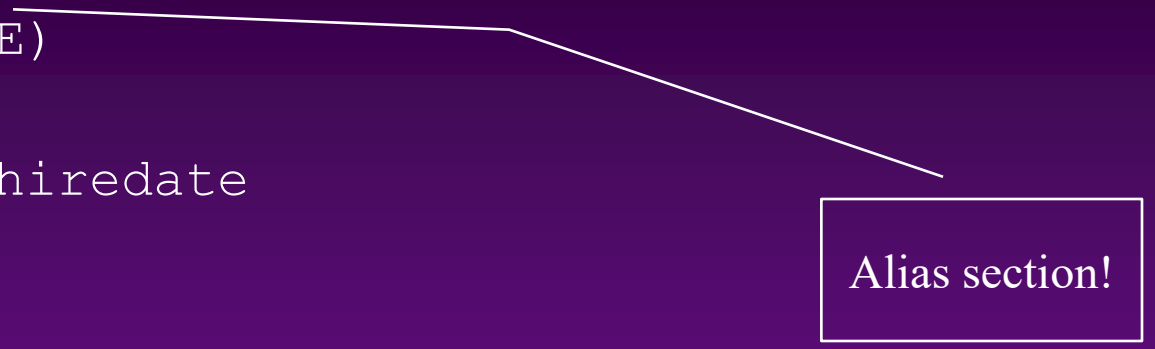
- Problem:
  - Views can be based on views that can be based on views …
  - How can you read what is really accessed?
- Solution (12c-only)
  - DBMS_UTILITY.EXPAND_SQL_TEXT

SQL Text Expansion (2)

```
SQL> declare
  2      v_tx varchar2(32767);
  3  begin
  4      dbms_utility.expand_sql_text
  5            ('select * from scott.v_emp', v_tx);
  6      dbms_output.put_line(v_tx);
  7  end;
  8  /

SELECT "A1"."EMPNO" "EMPNO","A1"."ENAME" "ENAME","A1"."DEPTNO"
"DEPTNO","A1"."DNAME" "DNAME","A1"."SAL" "SAL" FROM  (SELECT "A3"."EMPNO"
"EMPNO","A3"."ENAME" "ENAME","A3"."DEPTNO" "DEPTNO","A2"."DNAME"
"DNAME","A3"."SAL" "SAL" FROM "SCOTT"."EMP" "A3","SCOTT"."DEPT" "A2" WHERE
"A3"."DEPTNO"="A2"."DEPTNO" AND "A3"."EMPNO"<>30) "A1"
```

Invisible Columns

```
SQL> create or replace view v_emp_invisible (
  2       empno,
  3       ename,
  4       hiredate INVISIBLE)
  5  as
  6  select empno, ename, hiredate
  7  from emp
  8  /

SQL> select *
  2  from v_emp_invisible;

EMPNO      ENAME
---------- ----------
7839       KING
```



Real Business Cases

DULCIAN INC.

Storytelling

- #0 Views and front-end developers
- #1: When to stop
- #1: Views as architectural tools

DULCIAN INC.

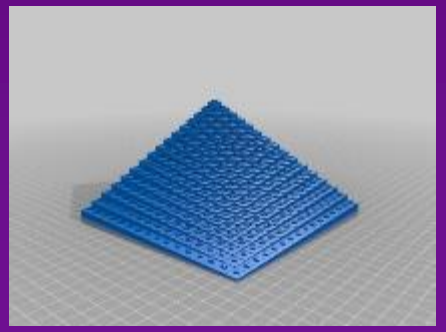
Story #0: Views and Front-End

How to Talk to Front-End Developers

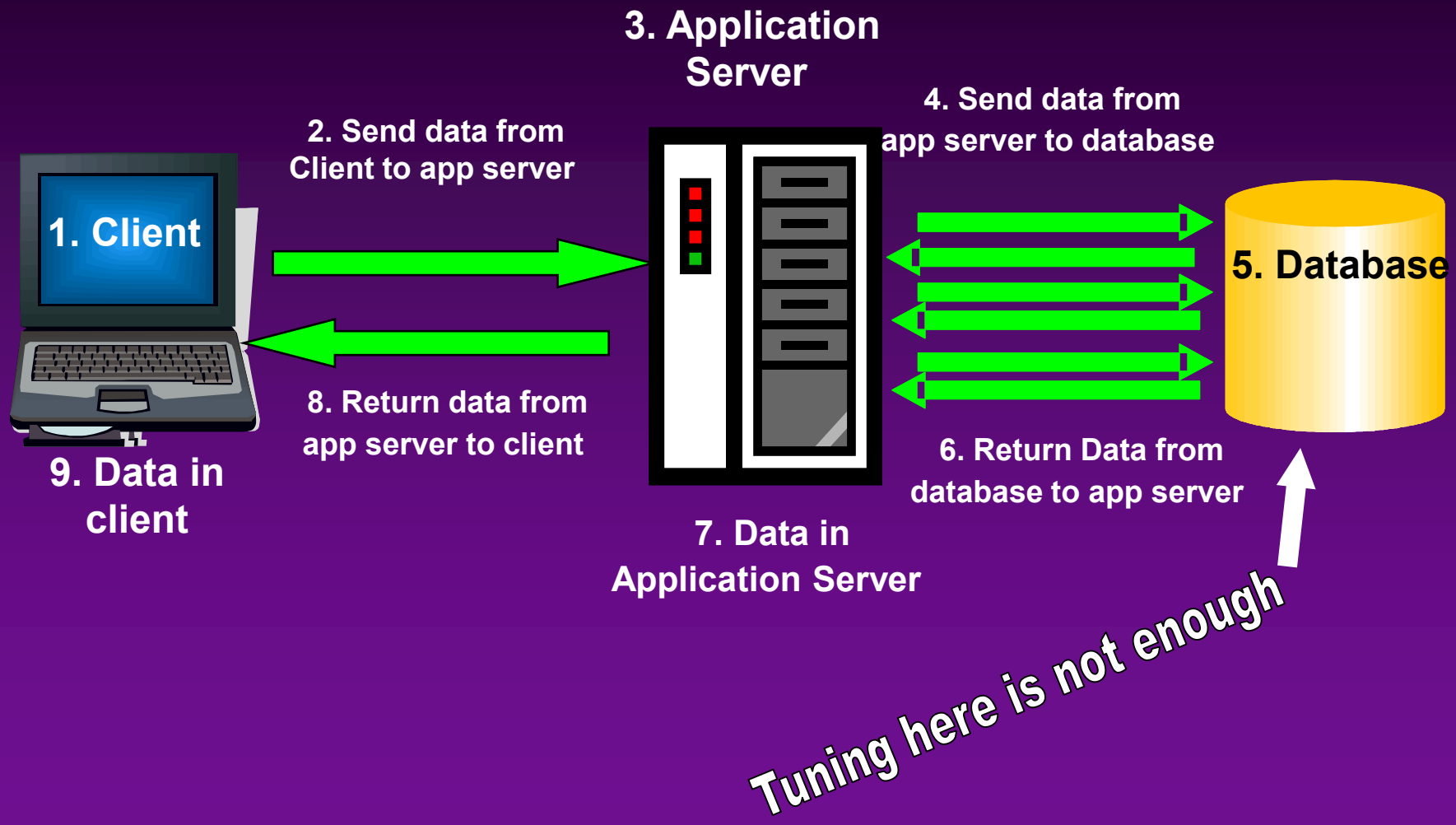
- Problem:
  - How to explain the importance of views to front-end developers?
- Solution:
  - Explain the idea of roundtrip minimization!

Practical Aspects (1)

- Existing implementations:
  - Multiple separate requests to populate the screen
  - Total number of roundtrips sometimes in thousands for a single screen
    - Each request is cheap.
    - Total cost on all levels (DB/network/app server) is huge.



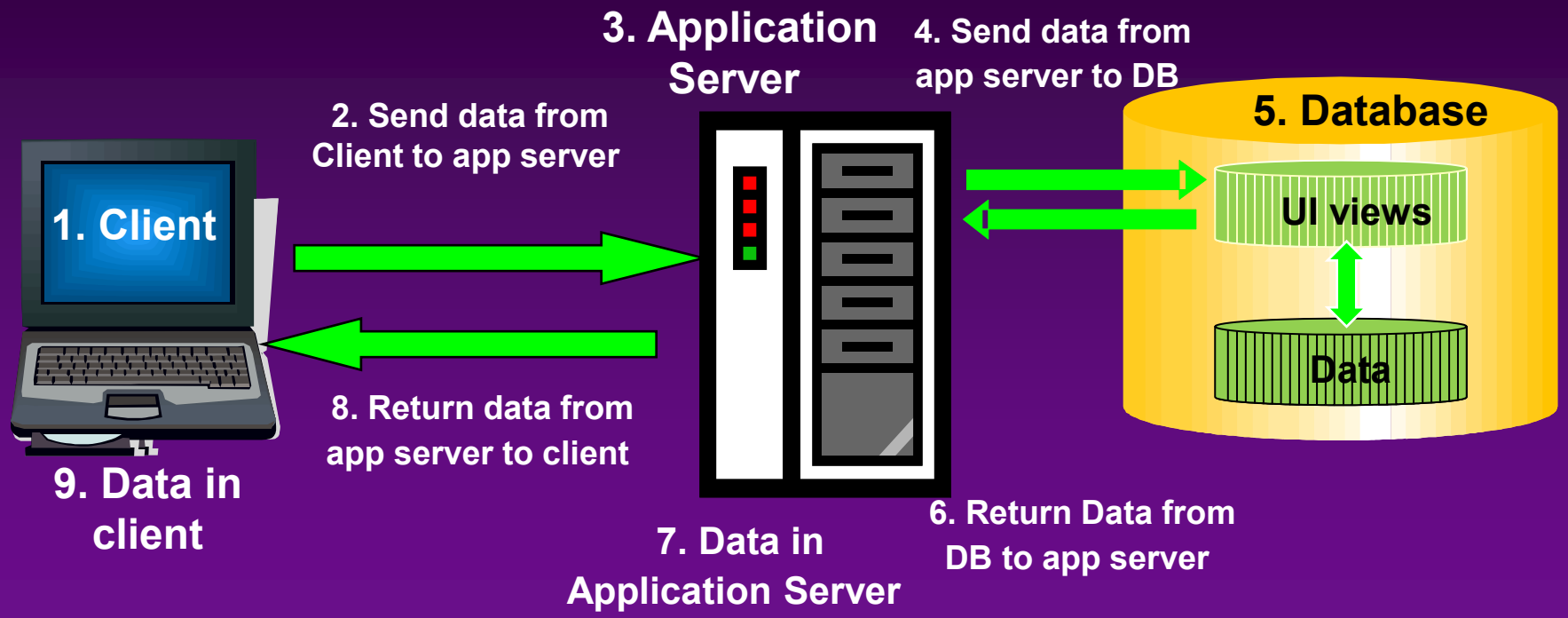
Web Application Architecture



Practical Aspects (2)

- Alternative:
  - A view where 1 row gets all data to populate one screen → 1 round-trip
    - Since not a lot of data is being returned, the complexity of such a view can be pretty high.
    - Very often, you trade off higher memory utilization for less network requests.

Web Application Architecture



Story #1: When to stop?

Dangers of Views

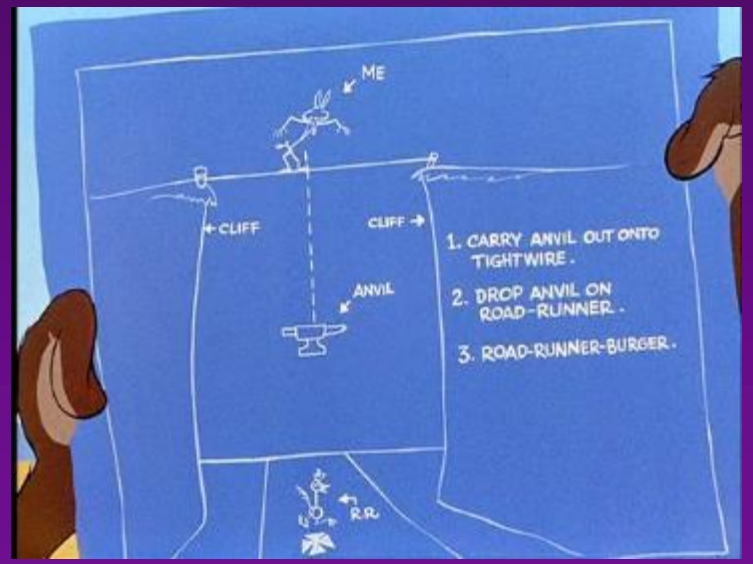
- Views are definitions, not data
  - Calling view multiple times = doing the same job multiple times.
  - There may be a moment when persisting query results may be more efficient.
  - Implementation of persistency depends upon your requirements
    - Materialized views (lots of options)
    - Tables + manual refresh scripts

DULCIAN INC.

Real World Example

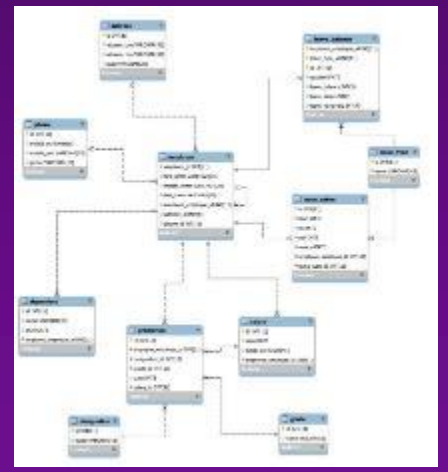
- Existing solution:
  - Reporting module is based on very complex views
  - After corporate merge, data volume quadrupled
  - Reports started to slow down.
- Revised Solution:
  - Core views became materialized with nightly full refresh.
  - Users gladly accepted daily data lag.
- Advantage of using views from the beginning:
  - Simple server-side change did the job!

Story #2: Views as Architectural Tools



Data Modeling Problem

- ERD data model
  - Can be perfectly implemented as tables+constraints
  - Very limited
- Class model
  - Much better tool to describe the world
  - Cannot be directly mapped to table structure



Solution

- Class model:
  - Table - perfectly relational
  - Package - implements UML-related functionality
  - View – presents class exactly as specified in class model

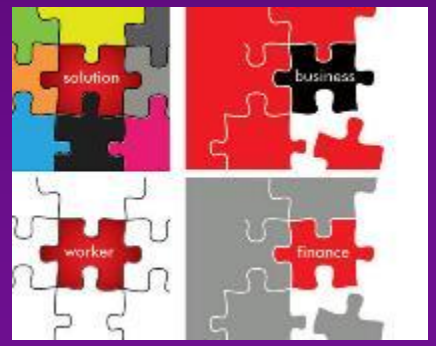


Illustration (1)

- Abstract class:

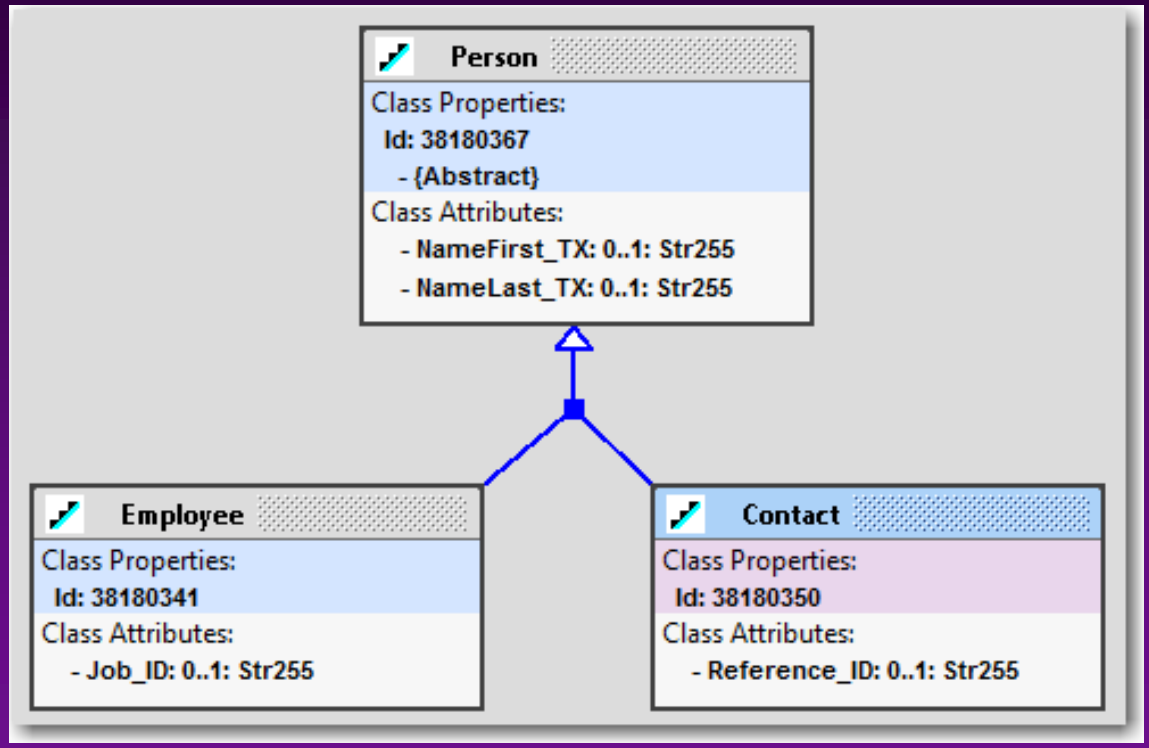


Illustration (2)

- Generated view:

```
CREATE OR REPLACE VIEW person
AS
select
 EMPLOYEE_OID PERSON_OID
,'Employee' PERSON_CD
,to_char(NameLast_TX||' '||NameFirst_TX) PERSON_DSP
,NAMEFIRST_TX NAMEFIRST_TX
,NAMELAST_TX NAMELAST_TX
from EMPLOYEE

 union all

select
 CONTACT_OID PERSON_OID
,'Contact' PERSON_CD
,to_char(NameLast_TX||' '||NameFirst_TX) PERSON_DSP
,NAMEFIRST_TX NAMEFIRST_TX
,NAMELAST_TX NAMELAST_TX
from CONTACT
```

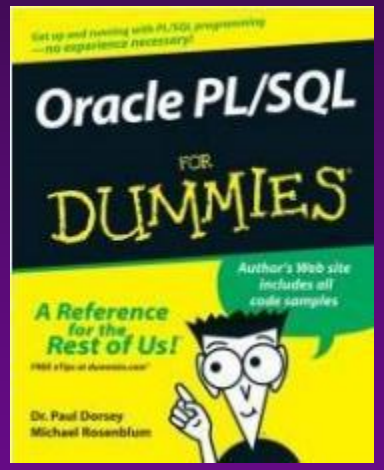
DULCIAN INC.

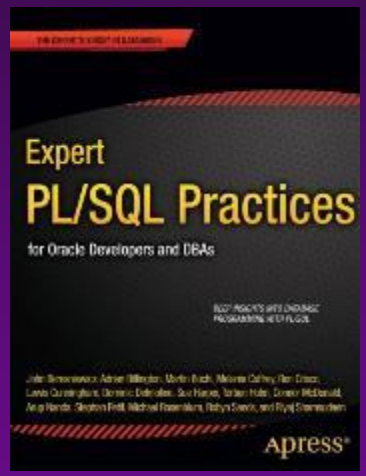
Summary

- Views are significantly more than just stored queries
  - ... because they are interfaces to the outside world (and even Oracle agrees ☺).
- Effective utilization of the database is impossible without deep knowledge of PL/SQL
  - … since it is closer to your data than any other language environment.
- It is important to check for new (or forgotten) features
  - … because there are LOTS of them!

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