

WELCOME

Fine Grained PL/SQL and
SQL Dependency
Analysis

DOAG Conference 2012

Philipp Salvisberg

20th November 2012

BASEL BERN LAUSANNE ZÜRICH DÜSSELDORF FRANKFURT A.M. FREIBURG I.BR. HAMBURG MÜNCHEN STUTTGART WIEN

1

2012 © Trivadis

Fine Grained PL/SQL and SQL Dependency Analysis
20-November-2012

trivadis
makes IT easier. ■ ■ ■

About Me

- With Trivadis since April 2000
 - Senior Principal Consultant
 - Partner
 - Member of the Board of Directors
 - philipp.salvisberg@trivadis.com
 - www.trivadis.com

- Member of the **trivadis**
performanceteam

- Main focus on database centric development with Oracle DB
 - Application Development
 - Business Intelligence
 - Application Performance Management
- Over 20 years experience in using Oracle products



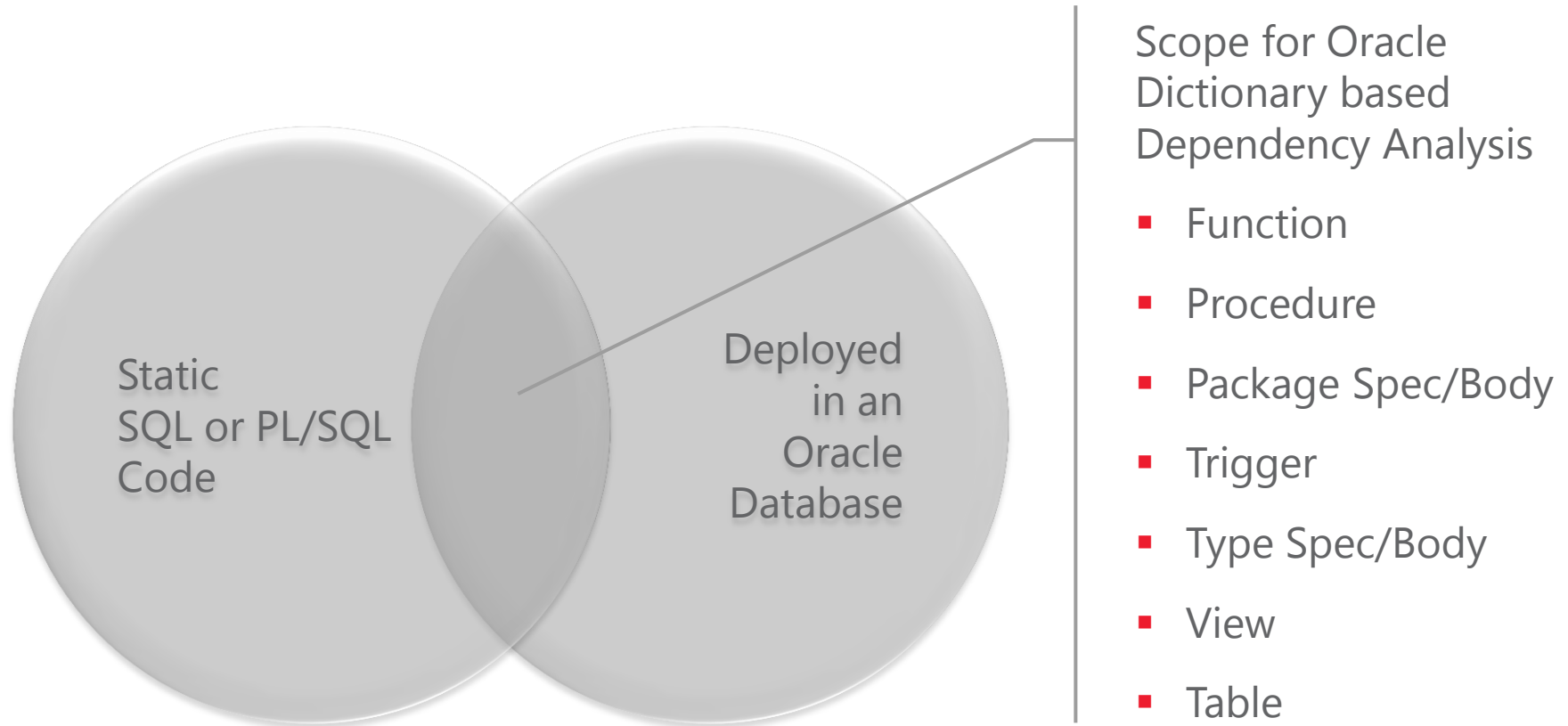
AGENDA

1. Introduction to PL/SQL and SQL Analysis using Oracle Data Dictionary
2. Solution Approach for Analysis beyond the Oracle Data Dictionary
3. Analysis #1 – SQL Statements using Hints
4. Analysis #2 – Tables, Views used in Queries
5. Analysis #3 – Tables, Views used in DML Statements
6. Analysis #4 – Views exposing specific Table Column
7. Core Messages

Reasons for PL/SQL and SQL (Dependency) Analysis

- Evaluate the impact of a change
 - E.g. impact of data model amendments
 - E.g. impact of PL/SQL source code amendments
- Understand the impact of a deployment or recompilation
 - E.g. invalidated objects, requested locks at deployment time
- Understand an application better
 - E.g. after ownership change
- Ensure standards are met
 - E.g. querying or changing tables through a defined API only

Primary Scope



Oracle Data Dictionary Views

before Oracle 11g Release 1

- DBA_DEPENDENCIES
- DBA_SOURCE
- DBA_PROCEDURES
- DBA_ARGUMENTS
- DBA_TYPES
- DBA_TYPE_METHODS
- DBA_TRIGGERS
- DBA_VIEWS
- DBA_TABLES
- ...

Oracle 11g Release 1 and newer

- DBA_IDENTIFIERS

```
-- enable PL/SCOPE for a session
ALTER SESSION SET PLSCOPE_SETTINGS =
  'IDENTIFIERS:ALL';

-- enable PL/SCOPE system wide
ALTER SYSTEM SET PLSCOPE_SETTINGS =
  'IDENTIFIERS:ALL' SCOPE = BOTH;

-- recompile PL/SQL objects to
-- populate PL/SCOPE dictionary
...
```

Oracle Data Dictionary Granularity vs. Analysis Needs

DBA_DEPENDENCIES

- Object (Table, View, Package, Package Body, ...)
- Internally there's more, but not exposed, see Rob van Wijk's post about DBA_DEPENDENCY_COLUMNS <http://rwijk.blogspot.com/2008/10/dbadependencycolumns.html>



DBA_IDENTIFIERS

- PL/SQL identifier (variable, function, ...) with usage (call, reference, ...)
- Context as hierarchy of usage_id (usage_context_id = parent)
- But no support for SQL (Select, Insert, Update, Delete, Merge)



AGENDA

1. Introduction to PL/SQL and SQL Analysis using Oracle Data Dictionary
2. **Solution Approach for Analysis beyond the Oracle Data Dictionary**
3. Analysis #1 – SQL Statements using Hints
4. Analysis #2 – Tables, Views used in Queries
5. Analysis #3 – Tables, Views used in DML Statements
6. Analysis #4 – Views exposing specific Table Column
7. Core Messages

Guidelines for SQL and PL/SQL Analysis

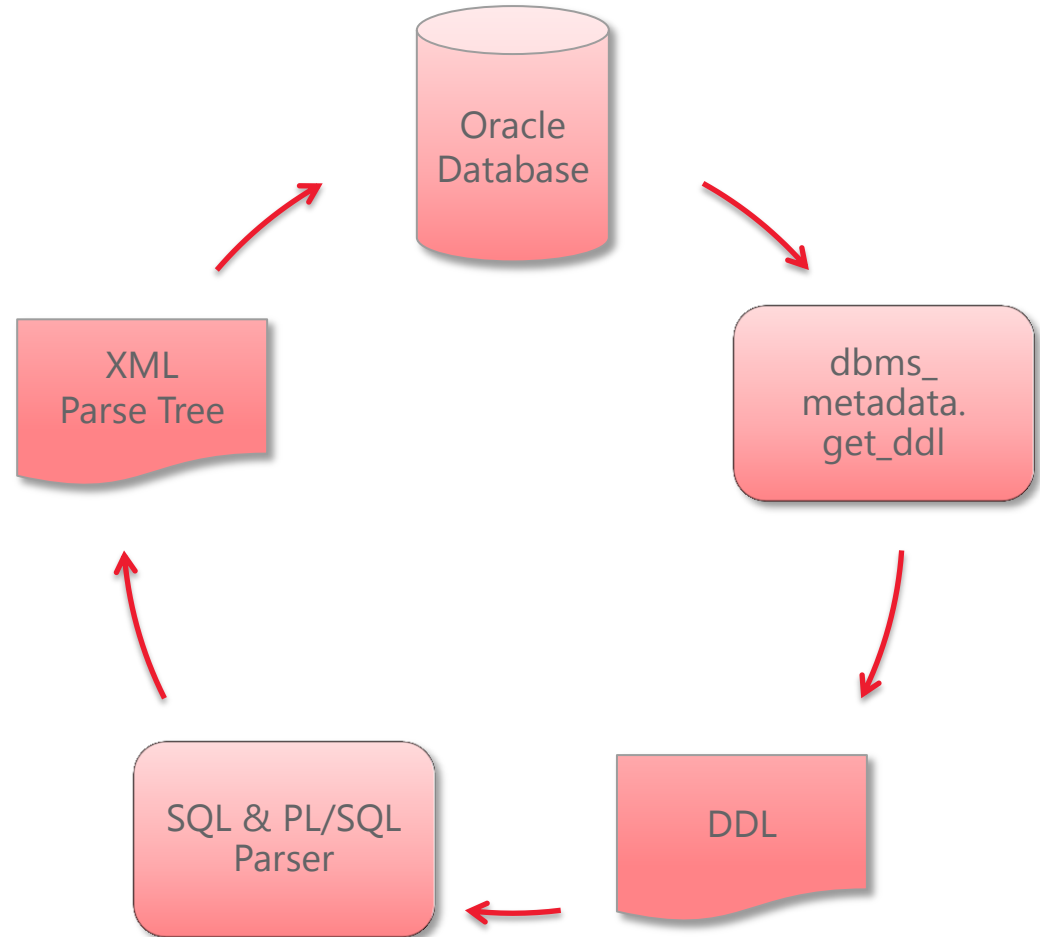
Apply the guidelines in the given priority:

1. Use DBA_DEPENDENCIES if feasible
2. Use other data dictionary views and combine them
3. Use PL/Scope if feasible
4. Create the DBA_DEPENDENCY_COLUMNS view if feasible
 - See <http://rwijk.blogspot.com/2008/10/dbadependencycolumns.html>
5. Use an Oracle parser if applicable for missing information only
 - E.g. UTL_XML.PARSEQUERY for SELECT statements
6. Use own or 3rd party parser as the last resort for missing information

Parsed Objects – Extend Oracle Data Dictionary

```
SQL> desc tvd_parsed_objects_t
```

Name	Type
OBJECT_ID	NUMBER
OWNER	VARCHAR2(30 CHAR)
OBJECT_NAME	VARCHAR2(128 CHAR)
OBJECT_TYPE	VARCHAR2(30 CHAR)
LAST_DDL_TIME	DATE
DDL_SOURCE	CLOB
PARSE_TREE	XMLTYPE



Populate and Refresh TVD_PARSED_OBJECTS_T

- Parser in Oracle Database, PL/SQL package to refresh
 - Pro: refresh available to all users via grant, ad-hoc parsing
 - Con: installation is slow and error-prone, inefficient use of DB server resources
- Parser as Web Service, PL/SQL package to refresh
 - Pro: refresh available to all users via grant, ad-hoc parsing
 - Con: needs a application server reachable from the database
- Parser as Standalone Application, OS script to refresh
 - Pro: easy setup, suitable for restricted environments
 - Con: needs access to standalone application for refresh, no ad-hoc parsing

- Create Web Service in Eclipse using Axis2
 - See <http://www.softwareagility.gr/index.php?q=node/29>
- Serialize Ecore model to XML (for Xtext grammars only)

```
private String serialize(PLSQLFile plsql) throws IOException {
    URI fileURI = URI.createFileURI("example.xml");
    Resource res = new XMLResourceFactoryImpl().createResource(fileURI);
    ByteArrayOutputStream os = new ByteArrayOutputStream();
    res.getContents().add(plsql);
    res.save(os, null);
    return new String(os.getBytes());
}
```

- Use UTL_HTTP to call a web service
see <http://martin-mares.com/2010/08/oracle-db-consume-wsdl-webservice-in-plsql/>
- Get Parse Tree from DDL

```
CREATE OR REPLACE PACKAGE tvd_parser_pkg IS
    FUNCTION parse_plsql(p_source_in CLOB) RETURN XMLTYPE;
END tvd_parser_pkg;
/
```

- Refresh TVD_PARSED_OBJECTS_T (according LAST_DDL_TIME)

```
CREATE OR REPLACE PACKAGE tvd_parsed_objects_pkg authid CURRENT_USER IS
    PROCEDURE refresh(p_owner_in          IN VARCHAR2 DEFAULT NULL,
                     p_object_type_in    IN VARCHAR2 DEFAULT NULL,
                     p_object_name_in    IN VARCHAR2 DEFAULT NULL);
END tvd_parsed_objects_pkg;
/
```

AGENDA

1. Introduction to PL/SQL and SQL Analysis using Oracle Data Dictionary
2. Solution Approach for Analysis beyond the Oracle Data Dictionary
3. Analysis #1 – SQL Statements using Hints
4. Analysis #2 – Tables, Views used in Queries
5. Analysis #3 – Tables, Views used in DML Statements
6. Analysis #4 – Views exposing specific Table Column
7. Core Messages

Analysis #1 – SQL Statements using Hints

DEMO

- Usage within Select, Insert, Update, Delete, Merge
 - Consider all query blocks
 - Distinguish between hints and comments
- Example

```
SQL> SELECT object_name, object_type, position, hints, usage
       2     FROM tvd_object_hint_usage_v t
       3     WHERE owner = 'TVDCC';
```

OBJECT_NAME	OBJECT_TYPE	POSITION	HINTS	USAGE
TVD_SAMPLE_PACKAGE	PACKAGE BODY	1	/*+ a */	DeleteStatement
TVD_SAMPLE_PACKAGE	PACKAGE BODY	2	/*+ b */	InsertStatement
TVD_SAMPLE_PACKAGE	PACKAGE BODY	3	/*+ c */	QueryBlock
TVD_SAMPLE_PACKAGE	PACKAGE BODY	4	/*+ d */	UpdateStatement
TVD_SAMPLE_PACKAGE	PACKAGE BODY	5	/*+ e */	MergeStatement

```
CREATE OR REPLACE VIEW tvd_object_hint_usage_v AS
SELECT tobj.owner,
       tobj.object_name,
       tobj.object_type,
       tab.position,
       tab.hints,
       tab.usage
FROM tvd_parsed_objects_t tobj,
     XMLTABLE('for $i in //hints
              where substring($i/text(),3,1) = "+"
              return <hints
                 usage="{ $i/ancestor::node() [1]/@xsi:type} ">{ $i/text() }</hints>'
              PASSING tobj.parse_tree
              COLUMNS "HINTS" VARCHAR2(1000) PATH 'text()',
                       "USAGE" VARCHAR2(30)    PATH 'substring(@usage,7)',
                       position FOR ORDINALITY) tab;
```


AGENDA

1. Introduction to PL/SQL and SQL Analysis using Oracle Data Dictionary
2. Solution Approach for Analysis beyond the Oracle Data Dictionary
3. Analysis #1 – SQL Statements using Hints
4. Analysis #2 – Tables, Views used in Queries
5. Analysis #3 – Tables, Views used in DML Statements
6. Analysis #4 – Views exposing specific Table Column
7. Core Messages

- Usage within View, Function, Procedure, Trigger, Package Spec/Body, Type Body
 - Consider inner query blocks for subqueries
 - Procedure_name semantics according DBA_PROCEDURES
- Example

```
SQL> SELECT object_type, object_name, position AS pos, procedure_name,  
2         table_owner AS t_own, table_name  
3     FROM tvd_object_query_usage_v t  
4     WHERE owner = 'TVDC'  
5     ORDER BY object_name, object_type;
```

OBJECT_TYPE	OBJECT_NAME	POS	PROCEDURE_NAME	T_OWN	TABLE_NAME
VIEW	TVD_OBJECT_QUERY_USAGE_V	1			TVD_PARSED_OBJECTS_T
PACKAGE BODY	TVD_PARSED_OBJECTS_PKG	1	REFRESH	SYS	DBA_OBJECTS
PACKAGE BODY	TVD_PARSED_OBJECTS_PKG	2	REFRESH		TVD_PARSED_OBJECTS_T
PACKAGE BODY	TVD_PARSED_OBJECTS_PKG	3	REFRESH		DBA_OBJECTS
FUNCTION	TVD_SAMPLE_FUNCTION	1	INNER_PROCEDURE	TVDC	TVD_PARSED_OBJECTS_T
...					

```
CREATE OR REPLACE VIEW tvd_object_query_usage_v AS
SELECT tobj.object_id, tobj.owner, tobj.object_type, tobj.object_name,
       'SELECT' AS operation, tab.position, upper(tab.procedure_name) AS procedure_name,
       upper(tab.table_owner) AS table_owner, upper(tab.table_name) AS table_name
FROM tvd_parsed_objects_t tobj,
     xmltable('for $i in //queryTableExpression[@qteName]
              let $items := $i/ancestor::items[not(@xsi:type =
                "plsql:CursorDeclarationOrDefinition") and position()=1]
              let $elements := $i/ancestor::elements[not(@xsi:type =
                "plsql:TableReference") and position()=1]
              let $proc := if ($items) then
                            concat($items/heading/sqlObject/@value,
                                    $items/heading/function/@value)
                          else
                            concat($elements/function/function/@value,
                                    $elements/procedure/procedure/@value,
                                    $elements/datatype/@value)
              return <result table="{ $i/@qteName}" schema="{ $i/@schema}" proc="{ $proc}"/>'
           passing tobj.parse_tree
           columns "TABLE_NAME" VARCHAR2(30) path '@table',
                  "TABLE_OWNER" VARCHAR2(30) path '@schema',
                  "PROCEDURE_NAME" VARCHAR2(30) PATH '@proc',
                  position FOR ordinality) tab;
```

AGENDA

1. Introduction to PL/SQL and SQL Analysis using Oracle Data Dictionary
2. Solution Approach for Analysis beyond the Oracle Data Dictionary
3. Analysis #1 – SQL Statements using Hints
4. Analysis #2 – Tables, Views used in Queries
5. Analysis #3 – Tables, Views used in DML Statements
6. Analysis #4 – Views exposing specific Table Column
7. Core Messages

Analysis #3 – Tables, Views used in DML Statements

- Usage within Function, Procedure, Trigger, Package Spec/Body, Type Body
 - Consider Insert, Update, Delete, Merge statements
 - Procedure_name semantics according DBA_PROCEDURES
- Example

```
SQL> SELECT object_type, object_name, operation AS op, position AS pos, procedure_name,  
2         table_owner AS t_own, table_name  
3     FROM tvd_object_dml_usage_v t  
4     WHERE owner = 'TVDCC';
```

OBJECT_TYPE	OBJECT_NAME	OP	POS	PROCEDURE_NAME	T_OWN	TABLE_NAME
PACKAGE BODY	TVD_PARSED_OBJECTS_PKG	INSERT	1	REFRESH	TVDCC	TVD_PARSED_OBJECTS_T
FUNCTION	TVD_SAMPLE_FUNCTION	INSERT	1	INNER_PROCEDURE	TVDCC	TVD_PARSED_OBJECTS_T
FUNCTION	TVD_SAMPLE_FUNCTION	INSERT	2	INNER_FUNCTION	TVDCC	TVD_PARSED_OBJECTS_T
FUNCTION	TVD_SAMPLE_FUNCTION	INSERT	3		TVDCC	TVD_PARSED_OBJECTS_T
PACKAGE BODY	TVD_SAMPLE_PACKAGE	INSERT	1	MOST_INNER_PROCEDURE	TVDCC	TVD_PARSED_OBJECTS_T
...						

Analysis #3 – View TVD_OBJECT_DML_USAGE_V (1)

```
CREATE OR REPLACE VIEW TVD_OBJECT_DML_USAGE_V AS
SELECT tobj.object_id,
       tobj.owner,
       tobj.object_type,
       tobj.object_name,
       'INSERT' AS operation,
       tab.position,
       upper(tab.procedure_name) AS procedure_name,
       upper(tab.table_owner) AS table_owner,
       upper(tab.table_name) AS table_name
FROM tvd_parsed_objects_t tobj,
     xmltable('for $i in //dmlExpressionClause[dmlName/@value
               and ancestor::statements/@xsi:type="plsql:InsertStatement"]
               let $items := $i/ancestor::items[1]
               let $elements := $i/ancestor::elements[1]
               let $proc := if ($items) then
                           concat($items/heading/sqlObject/@value,
                                   $items/heading/function/@value)
                           else
                           concat($elements/function/function/@value,
                                   $elements/procedure/procedure/@value,
                                   $elements/datatype/@value)
               return <result table="{ $i/dmlName/@value}" schema="{ $i/@schema}" proc="{ $proc}"/>'
       passing tobj.parse_tree
       columns "TABLE_NAME" VARCHA2(30) path '@table',
              "TABLE_OWNER" VARCHA2(30) path '@schema',
              "PROCEDURE_NAME" VARCHA2(30) PATH '@proc',
              position FOR ordinality) tab ...
```

Analysis #3 – View TVD_OBJECT_DML_USAGE_V (2)

```
... UNION ALL
SELECT tobj.object_id,
       tobj.owner,
       tobj.object_type,
       tobj.object_name,
       'UPDATE' AS operation,
       tab.position,
       upper(tab.procedure_name) AS procedure_name,
       upper(tab.table_owner) AS table_owner,
       upper(tab.table_name) AS table_name
FROM tvd_parsed_objects_t tobj,
     xmltable('for $i in //dmlTableExpressionClause[dmlName/@value
                and ancestor::statements/@xsi:type="plsql:UpdateStatement"]
              let $items := $i/ancestor::items[1]
              let $elements := $i/ancestor::elements[1]
              let $proc := if ($items) then
                            concat($items/heading/sqlObject/@value,
                                    $items/heading/function/@value)
                          else
                            concat($elements/function/function/@value,
                                    $elements/procedure/procedure/@value,
                                    $elements/datatype/@value)
              return <result table="{ $i/dmlName/@value}" schema="{ $i/@schema}" proc="{ $proc}"/>'
           passing tobj.parse_tree
           columns "TABLE_NAME" VARCHA2(30) path '@table',
                  "TABLE_OWNER" VARCHA2(30) path '@schema',
                  "PROCEDURE_NAME" VARCHA2(30) PATH '@proc',
                  position FOR ordinality) tab ...
```

Analysis #3 – View TVD_OBJECT_DML_USAGE_V (3)

```
... UNION ALL
SELECT tobj.object_id,
       tobj.owner,
       tobj.object_type,
       tobj.object_name,
       'DELETE' AS operation,
       tab.position,
       upper(tab.procedure_name) AS procedure_name,
       upper(tab.table_owner) AS table_owner,
       upper(tab.table_name) AS table_name
FROM tvd_parsed_objects_t tobj,
     xmltable('for $i in //dmlTableExpressionClause[dmlName/@value
               and ancestor::statements/@xsi:type="plsql:DeleteStatement"]
               let $items := $i/ancestor::items[1]
               let $elements := $i/ancestor::elements[1]
               let $proc := if ($items) then
                           concat($items/heading/sqlObject/@value,
                                   $items/heading/function/@value)
                           else
                           concat($elements/function/function/@value,
                                   $elements/procedure/procedure/@value,
                                   $elements/datatype/@value)
               return <result table="{ $i/dmlName/@value}" schema="{ $i/@schema}" proc="{ $proc}"/>'
           passing tobj.parse_tree
           columns "TABLE_NAME" VARCHA2(30) path '@table',
                  "TABLE_OWNER" VARCHA2(30) path '@schema',
                  "PROCEDURE_NAME" VARCHA2(30) PATH '@proc',
                  position FOR ordinality) tab ...
```


Analysis #3 – View TVD_OBJECT_DML_USAGE_V (4)

```
... UNION ALL
SELECT tobj.object_id,
       tobj.owner,
       tobj.object_type,
       tobj.object_name,
       'MERGE' AS operation,
       tab.position,
       upper(tab.procedure_name) AS procedure_name,
       upper(tab.table_owner) AS table_owner,
       upper(tab.table_name) AS table_name
FROM tvd_parsed_objects_t tobj,
     xmltable('for $i in //intoClause[@table
              and ancestor::statements/@xsi:type="plsql:MergeStatement"]
              let $items := $i/ancestor::items[1]
              let $elements := $i/ancestor::elements[1]
              let $proc := if ($items) then
                           concat($items/heading/sqlObject/@value,
                                   $items/heading/function/@value)
                           else
                           concat($elements/function/function/@value,
                                   $elements/procedure/procedure/@value,
                                   $elements/datatype/@value)
              return <result table="{ $i/@table }" schema="{ $i/@schema }" proc="{ $proc }"/>'
       passing tobj.parse_tree
       columns "TABLE_NAME" VARCHAR2(30) path '@table',
              "TABLE_OWNER" VARCHAR2(30) path '@schema',
              "PROCEDURE_NAME" VARCHAR2(30) PATH '@proc',
              position FOR ordinality) tab;
```

AGENDA

1. Introduction to PL/SQL and SQL Analysis using Oracle Data Dictionary
2. Solution Approach for Analysis beyond the Oracle Data Dictionary
3. Analysis #1 – SQL Statements using Hints
4. Analysis #2 – Tables, Views used in Queries
5. Analysis #3 – Tables, Views used in DML Statements
6. Analysis #4 – Views exposing specific Table Column
7. Core Messages

Analysis #4 – Views exposing specific Table Column

- Usage in Select List (column expression) is relevant only
- Consider recursivity and column name changes
- Example

```
SQL> SELECT schema_name, view_name, column_name FROM  
2 TABLE(tvd_coldep_pkg.get_dep('sh', 'costs', 'unit_cost'));
```

SCHEMA_NAME	VIEW_NAME	COLUMN_NAME
SH	PROFITS	UNIT_COST
SH	PROFITS	TOTAL_COST
SH	GROSS_MARGINS	GROSS_MARGIN
SH	GROSS_MARGINS	GROSS_MARGIN_PERCENT

- This is solvable using an Oracle parser
 - E.g. UTL_XML.PARSEQUERY, see <http://www.salvis.com/blog/?p=117>

Analysis #4 – View SH.PROFITS (existing)

Which view columns use COSTS.UNIT_COST?

```
CREATE OR REPLACE VIEW PROFITS AS
SELECT s.channel_id,
       s.cust_id,
       s.prod_id,
       s.promo_id,
       s.time_id,
       c.unit_cost,
       c.unit_price,
       s.amount_sold,
       s.quantity_sold,
       c.unit_cost * s.quantity_sold TOTAL_COST
FROM   costs c, sales s
WHERE  c.prod_id = s.prod_id
       AND c.time_id = s.time_id
       AND c.channel_id = s.channel_id
       AND c.promo_id = s.promo_id;
```

Analysis #4 – View SH.GROSS_MARGINS (new)

Which view columns use PROFITS.UNIT_COST, PROFITS.TOTAL_COST?

```
CREATE OR REPLACE VIEW GROSS_MARGINS AS
WITH gm AS
  (SELECT time_id, revenue, revenue - cost AS gross_margin
   FROM (SELECT time_id,
                unit_price * quantity_sold AS revenue,
                total_cost AS cost
         FROM profits))
SELECT t.fiscal_year,
       SUM(revenue) AS revenue,
       SUM(gross_margin) AS gross_margin,
       round(100 * SUM(gross_margin) / SUM(revenue), 2)
         AS gross_margin_percent
FROM gm
INNER JOIN times t ON t.time_id = gm.time_id
GROUP BY t.fiscal_year
ORDER BY t.fiscal_year;
```

Analysis #4 – View SH.REVENUES (new)

Which view columns use GROSS_MARGINS.GROSS_MARGIN,
GROSS_MARGINS.GROSS_MARGIN_PERCENT?

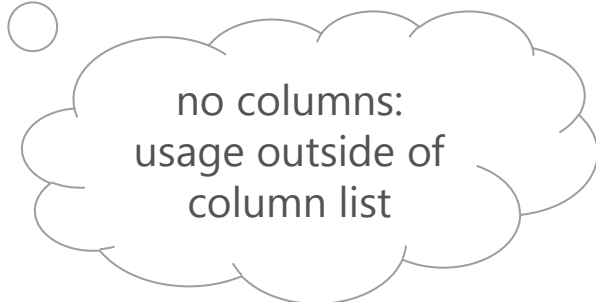
```
CREATE OR REPLACE VIEW REVENUES AS  
SELECT fiscal_year, revenue  
FROM gross_margins;
```

no columns:
table used but no
sensitive column

Analysis #4 – View SH.SALES_ORDERED_BY_GM (new)

Which view columns use PROFITS.UNIT_COST, PROFITS.TOTAL_COST?

```
CREATE OR REPLACE VIEW SALES_ORDERED_BY_GM AS
SELECT channel_id,
       cust_id,
       prod_id,
       promo_id,
       time_id,
       amount_sold,
       quantity_sold
FROM profits
ORDER BY (unit_price - unit_cost) DESC;
```



no columns:
usage outside of
column list

Analysis #4 – Package Body tvd_coldep_pkg (1)

```
CREATE OR REPLACE PACKAGE BODY tvd_coldep_pkg IS
  FUNCTION get_dep(p_schema_name IN VARCHAR2,
                  p_object_name IN VARCHAR2,
                  p_column_name IN VARCHAR2) RETURN tvd_coldep_1
    PIPELINED IS
  BEGIN
    -- query dictionary dependencies
    FOR v_dep IN (SELECT d.owner      AS schema_name,
                       d.name       AS view_name,
                       v.parse_tree AS parse_tree
                  FROM all_dependencies d
                  INNER JOIN tvd_parsed_objects_t v
                  ON v.owner = d.owner
                     AND v.object_name = d.name
                     AND v.object_type = d.type
                  WHERE d.referenced_owner = upper(p_schema_name)
                       AND d.referenced_name = upper(p_object_name)
                       AND d.type = 'VIEW')
    LOOP
      -- process every fetched view
      FOR v_views IN (SELECT VALUE(pv) coldep
                    FROM TABLE(process_view(v_dep.schema_name,
                                             v_dep.view_name,
                                             p_column_name,
                                             v_dep.parse_tree)) pv)
    LOOP ...
```


Analysis #4 – Package Body tvd_coldep_pkg (2)

```
... LOOP
    -- return column usages in v_dep.view_name
    PIPE ROW(v_views.coldep);
    -- get column usages of views using v_dep.view_name (recursive calls)
    FOR v_recursive IN (SELECT VALUE(dep) coldep
                        FROM TABLE(get_dep(v_views.coldep.schema_name,
                                           v_views.coldep.view_name,
                                           v_views.coldep.column_name)) dep)

        LOOP
            -- return column usages of recursive call
            PIPE ROW(v_recursive.coldep);
        END LOOP;
    END LOOP;
END LOOP;
END get_dep;

FUNCTION process_view(p_schema_name IN VARCHAR2,
                    p_view_name     IN VARCHAR2,
                    p_column_name   IN VARCHAR2,
                    p_parse_tree   IN xmltype) RETURN tvd_coldep_l IS
    v_search_l      tvd_coldep_l := tvd_coldep_l(tvd_coldep_typ(NULL,
                                                                NULL,
                                                                p_column_name));

    v_previous_count INTEGER := 0;
    v_coldep_l       tvd_coldep_l := tvd_coldep_l();
BEGIN ...
```

Analysis #4 – Package Body tvd_coldep_pkg (3)

```
... BEGIN
  -- get inline dependencies from secondary select lists, TODO: source/wildcard
  WHILE v_previous_count < v_search_1.count
  LOOP
    v_previous_count := v_search_1.count;
    FOR v_secondary IN (
      SELECT DISTINCT nvl(alias_name,
                          column_reference) AS alias_name
      FROM xmltable('for $i in //selected//*
                    where ($i/ancestor::fromList
                           or $i/ancestor::subqueryFactoringClause)
                           and $i/@value and not($i/self::alias)
                           return <ret column="{ $i/@value }"
                           alias="{ $i/ancestor::selected[1]//alias/@value }"/>'
                    passing p_parse_tree
                    columns column_reference VARCHAR2(1000) path '@column',
                           alias_name VARCHAR2(30) path '@alias') x
      WHERE upper(column_reference) IN
             (SELECT upper(column_name) FROM TABLE(v_search_1))
            AND upper(alias_name) NOT IN
             (SELECT upper(column_name) FROM TABLE(v_search_1)))
    LOOP
      -- add internal column usage
      v_search_1.extend;
      v_search_1(v_search_1.count) :=
        tvd_coldep_typ(NULL, NULL, v_secondary.alias_name); ...
```

Analysis #4 – Package Body tvd_coldep_pkg (4)

```
... v_search_1(v_search_1.count) :=
    tvd_coldep_typ(NULL, NULL, v_secondary.alias_name);
END LOOP;
END LOOP;
-- analyze primary select list
-- TODO: handle table/view source and wildcard properly
FOR v_primary IN (
    SELECT DISTINCT x.column_id, atc.column_name
    FROM xmltable('for    $i in //selected/*
                  where  not($i/ancestor::fromList
                             or $i/ancestor::subqueryFactoringClause)
                             and $i/@value and not($i/self::alias)
                  return <ret column="{ $i/@value }"
                           id="{count($i/ancestor::selected/preceding-sibling::*)+1}"/>'
    passing p_parse_tree
    columns column_reference VARCHAR2(1000) path '@column',
            column_id NUMBER path '@id') x
    INNER JOIN all_tab_columns atc
    ON atc.owner = p_schema_name
    AND atc.table_name = p_view_name
    AND atc.column_id = x.column_id
    WHERE upper(x.column_reference) IN
        (SELECT upper(column_name) FROM TABLE(v_search_1))
    ORDER BY x.column_id)
LOOP ...
```

Analysis #4 – Package Body tvd_coldep_pkg (5)

```
... LOOP
    -- add external column usage
    v_coldep_1.extend;
    v_coldep_1(v_coldep_1.count) := tvd_coldep_typ(p_schema_name,
                                                    p_view_name,
                                                    v_primary.column_name);

    END LOOP;
    -- return column dependencies
    RETURN v_coldep_1;
END process_view;
END tvd_coldep_pkg;
```

Core Messages

PL/SQL
& SQL

Analysis

- Column based analysis are not yet supported by the Oracle Data Dictionary
- SQL is not yet covered by PL/Scope
- Extend the Oracle Dictionary by gathering and persisting parse trees in XML format
- Queries based on the Extended Oracle Dictionary allows fine grained static PL/SQL and SQL code analysis
- Wrap complex queries in Views or Pipelined Table Functions to simplify analysis

THANK YOU.
VISIT US TO
TRIVADIS-STAND:
Floor 3, No. 304

Trivadis AG

Philipp Salvisberg

Europastrasse 5
8152 Glattbrugg (Zürich)

Tel. +41-44-808 70 20

Fax +41-44-808 70 21

philipp.salvisberg@trivadis.com

www.trivadis.com

BASEL BERN LAUSANNE ZÜRICH DÜSSELDORF FRANKFURT A.M. FREIBURG I.BR. HAMBURG MÜNCHEN STUTTGART WIEN