Introduction to DBMS_METADATA

NYC Metro Area Oracle Users Group Meeting

September 21, 2006

Paul Baumgartel

Charles Island Consulting

DBMS_METADATA concepts

- API for retrieval of DDL and associated information
- First available in Oracle9i
- Enhanced in Oracle10g
- Underlies Oracle 10g Data Pump
- Powerful, but somewhat daunting interface
- Returns XML by default

...DBMS_METADATA concepts

- Calls can specify
  - Filters (to include or exclude certain objects or object types)
  - Transforms (for example, to return SQL DDL instead of XML)
  - Parse items (to return information about the SQL returned, e.g. the name of a table created in a CREATE TABLE statement)
- Provides ability to execute the retrieved DDL

DBMS_METADATA interfaces

- Simple “browse” interfaces
  - GET_XML | GET_DDL
  - GET_DEPENDENT_ XML | GET_DEPENDENT_DDL
  - GET_GRANTED_XML | GET_GRANTED_DDL
- These allow transforms
- These do not allow filters, parse items

...DBMS_METADATA interfaces

- Examples of “browse” interfaces
- Usable in SQL queries

GET_DDL

```
SELECT DBMS_METADATA.GET_DDL('TABLE',u.table_name)
FROM USER_ALL_TABLES u WHERE u.nested='NO' AND
(u.iot_type is null or u.iot_type='IOT');
```

Returns SQL to create tables in current schema, excluding nested and index-organized tables

...DBMS_METADATA browse interfaces

GET_DEPENDENT_DDL

```
SELECT
DBMS_METADATA.GET_DEPENDENT_DDL('OBJECT_GRANT',
'EMPLOYEES','HR') FROM DUAL;
```

Returns SQL to grant object privileges on HR.EMPLOYEES

...DBMS_METADATA browse interfaces

GET_GRANTED_DDL

```
SELECT
DBMS_METADATA.GET_GRANTED_DDL('SYSTEM_GRANT',
                              'SCOTT')
FROM DUAL;
```

Returns SQL to grant system privileges to user SCOTT

… DBMS_METADATA interfaces

- Most advanced capabilities provided by opening a context for an object type and then building customizations before retrieval
- Context is indicated by a handle returned by the OPEN call
- Handle is passed in subsequent calls

DBMS_METADATA data types

Code fragment that begins a function to return DDL for all tables in a schema

```
function table_ddl (pi_schema_name in varchar2,
                    pi_sqlterminator in
                    boolean default false)
return sys.ku$_ddls is
v_outputddls sys.ku$_ddls := sys.ku$_ddls();
  v_parsed_items parsed_items_t;
begin
  v_handle := dbms_metadata.open('TABLE');
```

System-defined type

DBMS_METADATA data types

- DBMS_METADATA package uses several OBJECT and TABLE types, defined in the SYS schema
- Public synonyms on these are defined at database creation time
- Understanding of these data types is critical to effective use of the interface

DBMS_METADATA data types continued

- Two most frequently used are KU$_PARSED ITEM and KU$_DDL
- Both also exist as array types, KU$_PARSED ITEMS and KU$_DDLS

```
CREATE TYPE
sys.ku$_parsed_item
AS OBJECT (
  item VARCHAR2(30),
  value VARCHAR2(4000),
  object_row NUMBER )
```

```
CREATE TYPE
sys.ku$_ddl
AS OBJECT (
  ddlText CLOB,
  parsedItem
    sys.ku$_parsed_items )
```

Practical application of DBMS_METADATA: example 1

- Project to partition a number of nonpartitioned tables
- Implement via a PL/SQL package
  - Dynamically create a CREATE TABLE AS SELECT statement with partitioning clauses based on data in source table; give new table a temporary name
  - Use DBMS_METADATA to retrieve all dependent DDL
  - Drop source table
  - Rename new table to original source table name
  - Apply dependent DDL (with exception of indexes)

Practical application of DBMS_METADATA: example 2

- Application service provider
- Maintained scripts both to upgrade a schema and to create a new schema from scratch
- The “create new schema” script was rarely used, and effort required to maintain it was high

Practical application of DBMS_METADATA: example 2 continued

- The “create new schema” script had to be available for certain projects
- Utility package built around DBMS_METADATA extracted all DDL required to re-create schema, and wrote it to a file in the proper order to allow entire schema to be built from scratch

DBMS_METADATA example

- Package from which examples are excerpted provides two top-level capabilities
  - Get all dependent DDL (except grants) for an object
  - Get all DDL (except grants) to re-create a schema
  - Stores retrieved DDL in a table and can write to a file using UTL_FILE
- Ability to get DDL programmatically and use it in other utility applications

Function specification: uty_metadata.dependent_ddl

What kind of dependent DDL—constraint, index, grant?

```
-- Name                Description
-- ----                -----------
-- pi_schema_name      Object schema
-- pi_object_type      Type of dependent DDL to retrieve
-- pi_table_name       Table for which to get
-- pi_sqlterminator    “Add SQL terminator” flag
```

Dependent DDL (constraints, indexes, grants, for example) retrieved for this table)

How to use the interface: step by step

- The following examples demonstrate how to retrieve DDL
- Show the use of filters, transforms, and parse items

How to use the interface: initialization and filtering

- **Open a context handle**

```
v_handle := dbms_metadata.open(pi_object_type);
```

- **Set filter to get objects only for specified schema**

```
dbms_metadata.set_filter(v_handle,
                         cv_schema,
                         pi_schema_name);
```

How to use the interface: filtering

- Some other filters available
  - TABLESPACE returns objects residing in the specified tablespace
  - SPECIFICATION returns specification of a package or type if set to TRUE
  - GRANTEE selects objects that are granted to the specified user or role

How to use the interface: advanced filtering

- Advanced filtering uses SQL predicate (WHERE clause) fragments to create an INCLUDE or EXCLUDE name expression
- Predicate is appended to a DBMS_METADATA-generated query against the data dictionary
- Query is viewable via DBMS_METADATA.GET_QUERY

How to use the interface: advanced filtering, continued

- Use of EXCLUDE_NAME_EXPR

```
if pi_object_type = cv_index then
    dbms_metadata.set_filter(v_handle,
                  'EXCLUDE_NAME_EXPR',
'IN (SELECT CONSTRAINT_NAME FROM
ALL_CONSTRAINTS
WHERE CONSTRAINT_TYPE IN (''U'',''P'')
UNION SELECT INDEX_NAME FROM ALL_LOBS)');
end if;
```

How to use the interface: parse items

- In this example, a subroutine was used to request a fixed set of parse items for each DDL retrieved.
- Parse items are useful in identifying, categorizing, and storing DDL for later use

How to use the interface: parse items, continued

The name of the object in the DDL

```
dbms_metadata.set_parse_item(pi_handle, cv_name);
dbms_metadata.set_parse_item(pi_handle,
                             cv_object_type);
dbms_metadata.set_parse_item(pi_handle,
                            cv_base_object_name);
dbms_metadata.set_parse_item(pi_handle,
                            cv_base_object_type);
dbms_metadata.set_parse_item(pi_handle, cv_verb);
dbms_metadata.set_parse_item(pi_handle,
                             cv_schema);
```

The SQL verb, indicating the type of operation: CREATE, ALTER,...

How to use the interface: transforms

- Using transforms requires that a *transform handle* be opened with ADD_TRANSFORM
- The transform handle is opened with reference to an existing *context handle*

```
v_transform_handle :=
  dbms_metadata.add_transform(v_handle, cv_ddl);
```

How to use the interface: transforms, continued

- SET_TRANSFORM_PARAM specifies text changes

```
 if pi_sqlterminator = true then
dbms_metadata.set_transform_param
     (v_transform_handle,
      cv_sqlterminator, TRUE);
  end if;
```

How to use the interface: transforms, continued

- Additional transforms available with SET_TRANSFORM_PARAM
  - SEGMENT_ATTRIBUTES
  - STORAGE
  - TABLESPACE
- SET_REMAP_PARAM uses same transform handle to, for example
  - Remap tablespace
  - Remap schema

How to use the interface: fetch

- Once all filters and transforms are specified, fetch one DDL at a time
- A single DDL statement is returned in the ddlText member of the sys.ku$ddl type
- Parsed items, if requested, are returned in the parsedItem member (which is a nested type)

```
sys.ku$_parsed_item
AS OBJECT (
  item VARCHAR2(30),
  value VARCHAR2(4000),
  object_row NUMBER )
```

```
sys.ku$_ddl
AS OBJECT (
  ddlText CLOB,
  parsedItem
    sys.ku$_parsed_items )
```

How to use the interface: fetch, continued

- Multiple DDL statements may be returned:
  - When you call SET_COUNT to specify a count greater than 1
  - When an object is transformed into multiple DDL statements. For example, A TYPE object that has a DDL transform applied to it can be transformed into both CREATE TYPE and CREATE TYPE BODY statements. A TABLE object can be transformed into a CREATE TABLE, and one or more ALTER TABLE statements

```
v_localddls := dbms_metadata.fetch_ddl(v_handle);
```

Storing the retrieved DDL and parsed items

- Storage in the database can provide a repository
- In the example, parsed items are requested and fetched so that they can be stored in a table along with DDL
- Scalar variables are populated from the sys.ku$_ddls structure
- First assign the DDL text itself

```
v_ddl := pi_fetched_ddls(j).ddlText;
```

Storing the retrieved DDL and parsed items, continued

Because order of parsed items is indeterminate, must search for each one

```
for k in v_pi.first..v_pi.last loop
  if v_pi(k).item = cv_name then
  pi_parsed_items(cv_name) := v_pi(k).value;
  elsif v_pi(k).item = cv_object_type then
pi_parsed_items(cv_object_type):=v_pi(k).value;
  …
;
  end if;
end loop;
```

Associative array

Storing the retrieved DDL and parsed items, continued

```
insert into schema_metadata(schema_name, id,
                verb, object_type, object_name,
        base_object_type, base_object_name, ddl)
values (upper(pi_parsed_items(cv_schema)),
              metadata_id.nextval,
              pi_parsed_items(cv_verb),
              pi_parsed_items(cv_object_type),
              pi_parsed_items(cv_name),
              pi_parsed_items(cv_base_object_type),
              pi_parsed_items(cv_base_object_name),
              v_ddl);
```

metadata_id is a sequence

Using the stored DDL

- DBMS_METADATA offers the PUT function to allow submission of retrieved XML to the database to create objects
- An alternative is to
  - Store DDL and parse information in a table (as shown here)
  - Retrieve objects (proper order is important)
  - Write them to a file using UTL_FILE

Heterogeneous objects

- The Oracle10g version of DBMS_METADATA adds heterogeneous object types
- When a context is opened, specification of a heterogeneous object type such as SCHEMA_EXPORT initiates retrieval of a collection of objects that form a logical unit

Heterogeneous objects continued

- The interface can be used to achieve a table export, schema export, or database export
- Filters can also be applied to include only certain object types or to exclude certain object types
- This interface provides the underpinning for Oracle10g Data Pump export and import

Heterogeneous object type example

- Note similarity to scalar object type example
- Behavior and applicability of filters depends on object type specified in OPEN call

```
v_handle := dbms_metadata.open(cv_schema_export);
--Add a transform handle
v_transform_handle :=
dbms_metadata.add_transform(v_handle,cv_ddl);
--Set filter to retrieve objects for the specified
--schema only
dbms_metadata.set_filter(v_handle, cv_schema,
pi_schema_name);
```

Heterogeneous object type example continued

- Filter out certain object types that (in this example code) are retrieved separately
- Set transforms to add terminator, provide indented, “pretty” output

```
dbms_metadata.set_filter(v_handle,
                         cv_exclude_path_expr,
    'IN (''OBJECT_GRANT'',''TABLE'',''TYPE'')');
dbms_metadata.set_transform_param(v_xform_handle,
                                  cv_pretty,
                                  TRUE);
dbms_metadata.set_transform_param(v_xform_handle,
                         cv_sqlterminator, TRUE);
```

Summary

- No more convoluted SQL scripts to retrieve DDL
- DDL can be retrieved and applied programmatically for various purposes
  - Store source code for a schema revision
  - Create a schema copy with modifications
  - Drop, recreate, modify objects and apply dependent DDL

Questions?

- Contact me at
  - 917 549-4717
  - paul.baumgartel@aya.yale.edu
- I offer Oracle consulting and training
- Also see my Ask The Experts page at www.searchoracle.com