112 DBMS_JSON_DUALITY

The package `DBMS_JSON_DUALITY` simplifies the creation and access of JSON-relational duality views.

This package contains subprograms to:

1. Migrate or convert a set of JSON collections to JSON-relational duality views
2. Perform multi-operation non-interactive transactions on duality views

This chapter provides an overview of the package, outlines any security restrictions associated with it, and details the subprograms available within the package.

DBMS_JSON_DUALITY Overview

The `DBMS_JSON_DUALITY` package enables seamless integration between JSON documents and relational databases. The package provides subprograms that converts JSON collections into JSON relational duality views, imports data into duality views, and supports multi-operational transactions on the duality views.

The `DBMS_JSON_DUALITY` package contains a set of procedures and functions that allows you to find and create relational schema, import JSON documents to duality views and perform non-interactive transactions on the imported duality views.

1. **JSON-To-Duality Migrator :** Execute one of the appropriate JSON-To-Duality Migrator PL/SQL functions listed below to find and create the relational schema for the input collections (JSON tables).
   - `INFER_SCHEMA` : Infers the JSON schema that represents all of the input document sets.
   - `GENERATE_SCHEMA`: Produces the DDL code to create the required database objects for each duality view.
   - `INFER_AND_GENERATE_SCHEMA`: performs both operations of `INFER_SCHEMA` and `GENERATE_SCHEMA`.
2. **JSON-To-Duality Importer :**
   This feature is a PL/SQL procedure `IMPORT` to import application data from sets of JSON documents into duality views defined using the JSON-To-Duality Converter feature. Based on the relational schema created by the JSON-To-Duality Converter, input data gets decomposed and normalized into relational tables automatically. Any data that cannot be successfully imported is logged in an error log with the reason for the error and suggestions to fix the issue, if possible.

   > **Note:**
   >
   > Use the importer feature only when you are satisfied with the relational schema recommended by the JSON Relational Duality Converter feature.

3. **Non-Interactive Transactions**

This feature is a set of PL/SQL procedures that allows you to perform a multi-operational transaction on duality views. The following are the non-interactive procedures from package `DBMS_JSON_DUALITY`:

- `BEGIN_TRANSACTION` : Begins a multi-operational transaction.
- `REGISTER`: Check that the ETAG value of a document as last read matches that of the document in the database at the start of the transaction.
- `COMMIT_TRANSACTION`: Commits a multi-operational transaction.

DBMS_JSON_DUALITY Security

All subprograms in the `DBMS_JSON_DUALITY` package are executed with invoker rights. You must have `SELECT` privileges on tables and views referenced in a subprogram to use it.

In particular, JSON-To-Duality Migrator comprising the `INFER_SCHEMA`, `GENERATE_SCHEMA` and `INFER_AND_GENERATE_SCHEMA` subprograms, is subject to the following restrictions:

- They cannot be run in `SYS` schema.
- They require "select any table" privilege if `targetSchema` is not current schema. Otherwise, duality view creation fails while attempting to access the tables in `targetSchema`.

  Example:

  ```
  grant select any table to targetSchema;
  ```

Summary of DBMS_JSON_Duality Subprograms

This table lists the `DBMS_JSON_DUALITY` subprograms and briefly describes them.

DBMS_JSON_DUALITY Package Subprograms

Subprogram	Description
`INFER_SCHEMA` Function	Takes as input a set of JSON collections (tables with a single JSON column) and configuration parameters formatted as a JSON document. Returns the validated and normalized relational schema representing the collections as a JSON document
`GENERATE_SCHEMA` Function	Takes as input a normalized relational schema represented as a JSON document and returns the DDL script to generate the schema.
`INFER_AND_GENERATE_SCHEMA` Function	Takes as input a set of JSON collections (tables with a single JSON column) and configuration parameters formatted as a JSON document. Returns the DDL script to generate the validated and normalized relational schema representing the collections as a CLOB.

Subprogram	Description
`VALIDATE_SCHEMA_REPORT` Function	A table function which returns a report for schema validation. It validates the schema generated by `GENERATE_SCHEMA` or `INFER_AND_GENERATE_SCHEMA` functions and generates a report for documents that fail validation. **Note:** To validate a schema, it is assumed that you have created the schema by running an appropriate DDL script.
`IMPORT` Procedure	Imports data into the tables underlying the duality view(s). Logs any documents that fail import into an error log.
`IMPORT_ALL` Procedure	A procedure to import data from multiple tables into corresponding duality views.
`VALIDATE_IMPORT_REPORT` Function	A table function which return a report for import validation. It compares documents in the input JSON collection (tables with a single JSON column) with the documents of the corresponding duality view and generates a report for documents that fail validation. **Note:** To validate an import, it is assumed that you have imported the data using IMPORT Procedure or IMPORT_ALL procedure.
`BEGIN_TRANSACTION` Procedure	Starts a special multi-op transaction.
`COMMIT_TRANSACTION` Procedure	Commits the multi-op transaction.
`REGISTER` Procedure	Registers each document to be modified in the transaction, ensuring that the document has not been modified since the last read.

INFER_SCHEMA Function

This section describes the syntax, input and output formats of the `INFER_SCHEMA` function.

Overview

This function takes tables with a single JSON column and their configuration parameters formatted as a JSON document as input. The input fields of the JSON configuration document are described in the section Input Configuration Parameters. It returns the validated and normalized relational schema representing the collections as a JSON document.

Syntax

```
FUNCTION INFER_SCHEMA(config in JSON)
RETURN JSON;
```

Example 112-1   Example for INFER_SCHEMA Function

```
DECLARE
  schema_json json;
  schema_sql  clob;
BEGIN
  -- infer schema
  schema_json :=
    dbms_json_duality.infer_schema(
    json('{
           "tableNames"    : ["EMPLOYEE_TAB", "DEPARTMENT_TAB"],
           "viewNames"     : ["EMPLOYEES", "DEPARTMENTS"],
           "hints"         : json({"type" : "key",
                                 "table" : "EMP",
                                 "path" : "$",
                                 "value" : "empName"}),
           "useFlexFields" : true,
           "updatability"  : true,
           "sourceSchema"  : "HR",
           "targetSchema"  : "HR",
           "tablespace"    : "HR_TBS",
           "outputFormat"  : "executable"
          }'));
```

Input Configuration Fields

Note:

`INFER_SCHEMA`, `INFER_AND_GENERATE_SCHEMA` and `IMPORT_ALL` take a JSON configuration document as input. The parameter list for these functions are different but has some overlap. However, you can create a common configuration document that can be used for both functions and any parameters that do not apply to that function are ignored.

Field	Value
tableNames	JSON array of strings of the table names of JSON collection to be processed
viewNames	JSON array of strings of the output duality view names to be generated
useFlexFields	Boolean field which indicates whether flex fields should be used in the duality views. • When `true`, for each duality view, a flex column is added to each table that directly underlies the top-level fields of an object in the supported documents. • Default value for this field is `true`.
updatability	Boolean field which indicates whether the duality views created by the migrator should be updatable by default. • When `true`, the migrator will mark all duality views as updatable (i.e., annotations will be set for maximum updatability of the duality view). • When `false`, all duality views created will be read-only. • Default value for this field is `true`.
normalize	Boolean field which indicates whether the migrator will attempt to normalize the relational tables it infers for de-duplicating data. The default value for this parameter is 'true'
sourceSchema	This string field defines the schema name for the input tables. • Default value for this field is the current user schema.
targetSchema	This string field defines the schema name for the output tables and duality views. • There is no default value for this field. • If not provided then no database schema is specified in the output DDL; the names of the database objects to be created are unqualified. This means that the schema used is the one that's current at the time the DDL code is executed (not when it is generated).
tablespace	This string field defines the `tablespace` name to use for all tables created by the migrator. • This field has no default. • If this field is not specified, then the `tablespace` for tables created by the migrator will remain unspecified.
ingestLimit	This numeric field defines the maximum number of documents that will be analyzed in each JSON collection to infer the relational schema. • The default value of this parameter is 100000.

Field	**Value**
outputFormat	This string field defines the format of the output DDL script. • The possible values for this field are "standalone" and "executable". • The "executable" format creates a DDL script output which can be directly executed using the "execute immediate" PL/SQL construct. • The "standalone" format creates a DDL script which needs to be copied into a SQL file and executed independently. It cannot be executed using the "execute immediate" PL/SQL construct. • The default value of this field is "executable". **Note:** For cases where the output DDL statements can be larger than 32k characters, you MUST use the "standalone" option.
minFieldFrequency	This numeric field defines the threshold in percentage for identifying high entropy fields. • The default value for this field is 5, meaning that fields which appear in less than 5% of documents in a collection will be pruned from the schema.
minTypeFrequency	This numeric field defines the threshold in percentage for identifying high entropy types of a given field. • The default value for this field is 5, meaning that types for a field which appear in less than 5% of documents in a collection will be pruned from the schema.
softnessThreshold	This numeric field defines the threshold in percentage for allowable 'softness' in identifying functional dependencies in the input data. • The default value of this field is 99, meaning 1% of data could have missing or incorrect information. – The value of this field can be set based on the 'dirtiness' of input data. The dirtier (containing more errors) the data, the smaller the value of this field should be.

Field	Value
hints	A JSON array with elements that are JSON objects whose fields specify overrides for the behavior of the converter in generating a relational schema. A hint object must have these fields (otherwise, an error is raised): `type`, `table`, `path`, and `value`. All parameters except `value` are string type. • `type` : A hint can be of type `datatype`, `key` or `normalize`, specifying the type of migrator-behavior override. • `value` : Information specific to the particular `type`, providing detail that defines the hint. (This is the only hint field that is not necessarily a string.) • `table` : A string naming an input table whose document-set data is used to define a duality view. • `path` : A SQL/JSON path expression string that targets data in input JSON documents. **Note:** Detailed information about `hints` field could be found here : Configuration Fields for JSON-To-Duality Migrator

GENERATE_SCHEMA Function

This section describes the syntax, input and output formats of the `GENERATE_SCHEMA` function.

Overview

This function takes a normalized relational schema represented as a JSON document as input and returns the DDL script to generate the schema.

Syntax

```
FUNCTION GENERATE_SCHEMA(er_schema in JSON)
  RETURN CLOB;
```

Example 112-2 Example for GENERATE_SCHEMA Function

```
DECLARE
  schema_json json;
  schema_sql  clob;
BEGIN
  -- infer schema
  schema_json :=
    dbms_json_duality.infer_schema(
    json('{
           "tableNames"    : ["EMPLOYEE_TAB", "DEPARTMENT_TAB"],
           "viewNames"     : ["EMPLOYEES", "DEPARTMENTS"],
           "useFlexFields" : true,
           "updatability"  : true,
           "sourceSchema"  : "HR",
           "targetSchema"  : "HR",
           "tablespace"    : "HR_TBS",
           "outputFormat"  : "executable"
          }'));

   -- modify schema as needed
   ...

   -- generate schema
   schema_sql = dbms_json_duality.generate_schema(schema_json);

  -- create schema
  execute immediate schema_sql;
END;
/
```

Note:

The `schema_sql` is the output from `INFER_SCHEMA` function. The details of `INFER_SCHEMA` function are available here.

INFER_AND_GENERATE_SCHEMA Function

This section describes the syntax, input and output formats of the `INFER_AND_GENERATE_SCHEMA` function.

Overview

This function takes a set of JSON collections (tables with a single JSON column) and configuration parameters formatted as a JSON document as input. The input fields of the JSON configuration document are identical to the `INFER_SCHEMA` function and listed here. It returns the DDL script to generate the validated and normalized relational schema representing the collections as a CLOB.

Syntax

```
FUNCTION INFER_AND_GENERATE_SCHEMA(config in JSON)
  RETURN CLOB;
```

Example 112-3 Example for INFER_AND_GENERATE_SCHEMA Function

```
DECLARE
  schema_sql clob;
BEGIN
```

```
  -- infer and generate schema
  schema_sql :=
    dbms_json_duality.infer_and_generate_schema(
    json('{
           "tableNames"    : ["EMPLOYEE_TAB", "DEPARTMENT_TAB"],
           "viewNames"     : ["EMPLOYEES", "DEPARTMENTS"],
           "useFlexFields" : true,
           "updatability"  : true,
           "sourceSchema"  : "HR",
           "targetSchema"  : "HR",
           "tablespace"    : "HR_TBS",
           "outputFormat"  : "executable"
          }'));

   -- create schema
   execute immediate schema_sql;
END;
/
```

VALIDATE_SCHEMA_REPORT Function

This section describes the syntax, input and output formats of the `VALIDATE_SCHEMA_REPORT` function.

Overview

`VALIDATE_SCHEMA_REPORT` is a table function which return a report for schema validation. It validates the schema generated by `GENERATE_SCHEMA` or `INFER_AND_GENERATE_SCHEMA` functions and generates a report for documents that fail validation. Each row in the output corresponds to validation failure for a single JSON document. If no invalid document is found, the function does not return any rows. The output format for `VALIDATE_SCHEMA_REPORT` is a table with two CLOB columns, `data` and `errors`. The column `data` contains the erroneous JSON document, and the column `errors` contains an array of schema errors. The value of errors for `VALIDATE_SCHEMA_REPORT` is a JSON array containing JSON schema validation errors. The format of the array is identical to the format of the 'errors' field in the output of `DBMS_JSON_SCHEMA.VALIDATE_REPORT`.

Syntax

```
FUNCTION VALIDATE_SCHEMA_REPORT(
  table_owner_name in VARCHAR2 DEFAULT NULL,
  table_name       in VARCHAR2,
  view_owner_name  in VARCHAR2 DEFAULT NULL,
  view_name        in VARCHAR2
) return validate_report_tab_t PIPELINED;
```

Input Parameters

Table 112-1 Input Parameters for DBMS_JSON_DUALITY.VALIDATE_SCHEMA_REPORT

Parameter	Description
`table_owner_name`	The name of the owner of the input table with a single JSON column. The value of this field should be identical to the table owner name as present in the data dictionary. Otherwise, the schema of the current connected user is used.
`table_name`	The name of the input table with a single JSON column. The value of this field should be identical to the table name as present in the data dictionary.
`view_owner_name`	The name of the owner of the output duality view. The value of this field should be identical to the view owner name as present in the data dictionary. Otherwise, the schema of the current connected user is used.
`view_name`	The name of the output duality view. The value of this field should be identical to the view name as present in the data dictionary.

Example 112-4 Example for VALIDATE_SCHEMA_REPORT Function

Note:

To validate a schema, it is assumed that you have generated a schema using GENERATE_SCHEMA Function or INFER_AND_GENERATE_SCHEMA Function.

```
select * from dbms_json_duality.validate_schema_report('HR, 'EMPLOYEE_TAB',
'HR', 'EMPLOYEES');
select * from dbms_json_duality.validate_schema_report('HR, 'DEPARTMENT_TAB',
'HR', 'DEPARTMENTS');
```

IMPORT Procedure

This section describes the syntax, input and output formats of the `IMPORT` procedure.

Overview

This procedure imports data from a table with a single JSON column into a Duality View. It also logs any documents that fail import into an error log.

Note:

In the process of normalization, data may be transformed, cast to different data types, and truncated to honor maximum size limits. In addition, any data not conforming to the resulting schema may be rejected during the import phase. Oracle recommends to verify that all data has been successfully imported by running verification tests and looking at the error log for resolving import errors.

Syntax

```
PROCEDURE IMPORT(table_owner_name   in VARCHAR2 DEFAULT NULL,
                 table_name         in VARCHAR2,
                 view_owner_name    in VARCHAR2 DEFAULT NULL,
                 view_name          in VARCHAR2,
                 err_log_owner_name in VARCHAR2 DEFAULT NULL,
                 err_log_name       in VARCHAR2 DEFAULT NULL,
                 reject_limit       in NUMBER   DEFAULT NULL);
```

Input Parameter Fields

Parameter	Value
`table_owner_name`	The name of the owner of the input table with a single JSON column. You can specify the owner in `table_name`. Otherwise, the schema of the current connected user is used.
`table_name`	The name of the input table with a single JSON column.
`view_owner_name`	The name of the owner of the output duality view. You can specify the owner in `veiw_name`. Otherwise, the schema of the current connected user is used
`view_name`	The name of the output duality view.
`err_log_owner_name`	The name of the owner of the error log to use for document that cannot be imported successfully. You can specify the owner in `err_log_name`. Otherwise, the schema of the current connected user is used.
`err_log_name`	The name of the error log to use.The default value is NULL, so no errors will be logged by default.
`reject_limit`	Specifies the maximum number of errors that can be logged before the operation is aborted. The default (NULL) is unlimited, so the

Parameter	Value
	operation will never be aborted by default when `error_log_name` is specified.

Example 112-5 Example for IMPORT Procedure

```
BEGIN
  dbms_errlog.create_error_log(dml_table_name => 'EMPLOYEES',
err_log_table_name => 'ERR$_EMP', skip_unsupported => TRUE);
  dbms_errlog.create_error_log(dml_table_name => 'DEPARTMENTS',
err_log_table_name => 'ERR$_DEPT', skip_unsupported => TRUE);
  dbms_json_duality.import('HR', 'EMPLOYEE_TAB', 'HR', 'EMPLOYEES', 'HR',
'ERR$_EMP', 100);
  dbms_json_duality.import('HR', 'DEPARTMENT_TAB', 'HR' 'DEPARTMENTS', 'HR',
'ERR$_DEPT', 100);
END;
/
```

IMPORT_ALL Procedure

This section describes the syntax, input and output formats of the `IMPORT_ALL` procedure.

Overview

`IMPORT_ALL` is a procedure to import data from multiple tables into corresponding duality views. When multiple imports are needed, then this procedure should be used instead of the `IMPORT` procedure to avoid risk of foreign key constraint violation. The input for the `IMPORT_ALL` procedure is set of configuration options provided in the form of a JSON document. `IMPORT_ALL` has no output, since it is a procedure.

```
PROCEDURE IMPORT_ALL(config in JSON);
```

Table 112-2 Input Fields of the JSON Configuration Document

Field	Description
`tableNames`	Array of the names of the tables with a single JSON column to be imported. The value of this field should be identical to the table name as present in the data dictionary.
`viewNames`	Array of the names of the duality view. The order must be maintained between `tableNames` and `viewNames`. If not specified, the `viewNames` are derived from `tableNames` by post-fixing them with **'_DUALITY'**. The value of this field should be identical to the view name as present in the data dictionary.
`sourceSchema`	This string parameter defines the quoted schema name for the input tables. The value of this field should be identical to the source schema as present in the data dictionary. Default value for this parameter is the current user schema.

Table 112-2 (Cont.) Input Fields of the JSON Configuration Document

Field	Description
`targetSchema`	This string parameter defines the quoted schema name for duality views. The value of this field should be identical to the target schema as present in the data dictionary. Default value for this parameter is the current user schema.
`errorLog`	• String name of an error log to be used, OR • A string array of names of error logs to be used, one for each duality view. The value of this field should be identical to the error log name as present in the data dictionary.
`errorLogSchema`	The name of the schema of the error log to use for document that cannot be imported successfully. You can specify the owner in `errorLog`. Otherwise, the schema of the current connected user is used. The value of this field should be identical to the error log schema as present in the data dictionary.
`rejectLimit`	Specifies the maximum number of errors that can be logged before the operation is aborted. The default (NULL) is unlimited, so the operation will never be aborted by default when `errorLog` is specified.

Note:

All parameters except `tableNames` are optional. `INFER_SCHEMA`, `INFER_AND_GENERATE_SCHEMA` and `IMPORT_ALL` take a JSON configuration document as input. The parameter list for these functions are different but has some overlap. However, you can create a common configuration document that can be used for both functions and any parameters that do not apply to that function are ignored.

Example 112-6 Example for IMPORT_ALL Procedure

```
BEGIN
  dbms_errlog.create_error_log(dml_table_name => 'EMPLOYEES',
err_log_table_name => 'ERR$_EMP', skip_unsupported => TRUE);
  dbms_errlog.create_error_log(dml_table_name => 'DEPARTMENTS',
err_log_table_name => 'ERR$_DEPT', skip_unsupported => TRUE);
  dbms_json_duality.import_all(json('{"tableNames" :
["EMPLOYEE_TAB","DEPARTMENT_TAB"],
                                      "viewNames" :
["EMPLOYEES","DEPARTMENTS"],
                                      "sourceSchema" : "HR",
                                      "targetSchema" : "HR",
                                      "errorLog" : ["ERR$_EMP","ERR$_DEPT"]
                                     }'));
END;
/
```

VALIDATE_IMPORT_REPORT Function

This section describes the syntax, input and output formats of the `VALIDATE_IMPORT_REPORT` function.

Overview

`VALIDATE_IMPORT_REPORT` is a table function which return a report for import validation. It compares the documents passed as an input to `IMPORT` and IMPORT_ALL functions with the documents of the corresponding duality view. and generates a report for documents that fail validation. Each row in the output corresponds to validation failure for a single JSON document. If no invalid document is found, the function does not return any rows. The output format for `VALIDATE_IMPORT_REPORT` is a table with two CLOB columns, `data` and `errors`. The column `data` contains the erroneous JSON document, and the column `errors` contains an array of import errors. The value of errors for `VALIDATE_IMPORT_REPORT` is a JSON diff between the input JSON document in the table and the output JSON document in the duality view. The format of the diff is the same as the format of the JSON patch.

Syntax

```
FUNCTION VALIDATE_IMPORT_REPORT(
  table_owner_name  in VARCHAR2 DEFAULT NULL,
  table_name        in VARCHAR2,
  view_owner_name   in VARCHAR2 DEFAULT NULL,
  view_name         in VARCHAR2
) return validate_report_tab_t PIPELINED;
```

Input Parameters

Table 112-3   Input Parameters for DBMS_JSON_DUALITY.VALIDATE_IMPORT_REPORT

Parameter	Description
`table_owner_name`	The name of the owner of the input table with a single JSON column. The value of this field should be identical to the table owner name as present in the data dictionary. Otherwise, the schema of the current connected user is used.
`table_name`	The name of the input table with a single JSON column. The value of this field should be identical to the table name as present in the data dictionary.
`view_owner_name`	The name of the owner of the output duality view. The value of this field should be identical to the view owner name as present in the data dictionary. Otherwise, the schema of the current connected user is used.
`view_name`	The name of the output duality view. The value of this field should be identical to the view name as present in the data dictionary.

Example 112-7 Example for VALIDATE_IMPORT_REPORT Function

> **Note:**
>
> To validate an import, it is assumed that you have imported the data using IMPORT Procedure or IMPORT_ALL procedure.

```
select * from dbms_json_duality.validate_import_report('HR, 'EMPLOYEE_TAB',
'HR', 'EMPLOYEES');
select * from dbms_json_duality.validate_import_report('HR, 'DEPARTMENT_TAB',
'HR', 'DEPARTMENTS');
```

BEGIN_TRANSACTION Procedure

This section describes the syntax of the non-interactive procedure named `BEGIN_TRANSACTION`

Overview

The typical workflow for a noninteractive transaction involves the following procedures in the below-mentioned sequence:

- Begin the transaction
- Register the documents
- Perform queries or DMLs on the duality views
- Commit the transaction

The `BEGIN_TRANSACTION` procedure starts a special multi-operation transaction with the following properties:

- The multi-operation transaction provides repeatable reads, i.e., all reads run as of the snapshot when the transaction began.
- Locks are only taken for rows that are modified, unmodified rows remain unlocked throughout the transaction.

Syntax

```
 procedure begin_transaction;
```

Example 112-8 Example for BEGIN_TRANSACTION Procedure

```
dbms_json_duality.begin_transaction()
```

> **Note:**
>
> The `BEGIN_TRANSACTION` procedure is normally followed by `REGISTER` procedure. The detailed information about `REGISTER` procedure can be found here.

COMMIT_TRANSACTION Procedure

This section describes the syntax of the non-interactive procedure named `COMMIT_TRANSACTION`

Overview

This procedure commits the special multi-operation transaction. It validates all modified documents in the transaction, i.e., ensures that no other session has modified these documents.

Syntax

```
procedure commit_transaction;
```

Example 112-9 Example for COMMIT_TRANSACTION Procedure

`COMMIT_TRANSACTION` procedure is normally the last step in the non-interactive transaction after queries are performed on the duality views.

```
dbms_json_duality.commit_transaction();
```

REGISTER Procedure

This section describes the syntax of the non-interactive procedure named `REGISTER`

Overview

This procedure verifies that the current `etag` of the object is unchanged from a prior read. If the `etag` doesn't match the expected `etag`, and error is thrown from this function. The register procedure relies on object ids (oid) to identify objects/documents to register and validate. The object id can be obtained by querying the `resid` hidden column from the duality view. Its value is simply a concatenated binary encoding of all the primary key columns of the root table in the duality view.

Syntax

```
 procedure register(view_name in VARCHAR2,  /* duality view name
*/
                    oid       in RAW,       /* duality view obj identifier
*/
                    etag      in RAW);      /* document etag */
```

Example 112-10 Example for REGISTER Procedure

This example assumes that a duality view `team_dv` is already created. The detailed information about creating the duality views are explained under Creating Duality Views in Oracle Database JSON-Relational Duality Developer's Guide. Obtaining the `RESID` and `ETAG` is mandatory step prior to calling the `REGISTER` procedure. The following shows an example code to obtain the `RESID` and `ETAG` of the duality view `team_dv`:

```
SELECT RESID, DATA FROM team_dv dv
  WHERE dv.DATA.name LIKE 'Mercedes%';
```

This produces an output that includes the `oid` (`RESID`) and the `ETAG` information as shown below:

```
RESID
-----
DATA
----
FB03C2040400
{"_id" : 303,
 "_metadata":
  {"etag" : "039A7874ACEE6B6709E06E42E4DC6355",
   "asof" : "00000000001BE239"},
 "name" : "Mercedes",
 ...}
```

Note:

`REGISTER` procedure is normally performed after executing the `BEGIN_TRANSACTION` procedure. The details of `BEGIN_TRANSACTION` procedure are provided here.

```
BEGIN
  DBMS_JSON_DUALITY.begin_transaction();
  DBMS_JSON_DUALITY.register('team_dv',
                             hextoraw('FB03C2040400'),
                             hextoraw('039A7874ACEE6B6709E06E42E4DC6355'));
  .............................................
  ****Perform the updating (DML) operations****
  .............................................

  DBMS_JSON_DUALITY.commit_transaction();
END
```