Write Less (Code) with More (Oracle 12c New Features)

Oren Nakdimon
DB Oriented
Israel

Keywords:

Oracle12c; New Features; SQL; PL/SQL; Development; Productivity

Introduction

Oracle 12c introduced many new features that allow us developers to write less code than in previous releases, and become more efficient and productive.
Some features enhance the SQL language, so SQL statements can become much shorter and more readable than before.
Other features offer built-in functionality that previously had to be implemented by the application.
There are features that help write less configuration text, features that encourage writing less "inappropriately located" code, features that enable writing less administration scripts, and more.
In this presentation we look at several of these new features, and show them vs. the pre-12c solutions.
The features are presented under the following sub-categories:

- Write Less Configuration
- Write Less Application Code
- Write Less Code in SQL Statements
- Write Less "Inappropriately-Located" Code

Write Less Configuration

The following feature provides new functionality that reduces the need for writing configuration text.

SQL*Loader Express Mode

In Oracle 12c SQL*Loader can be executed with no control file and with many defaults, and it generates a log file for future use including control file, CREATE EXTERNAL TABLE statement and a corresponding INSERT statement.

In order to achieve the same functionality in pre-12c versions, we would write all of this manually.

Write Less Application Code

The following features provide new functionality that previously had to be implemented in the application level.

Identity Columns

In Oracle 12c a table column can be created as "identity". As a result, the column implicitly becomes mandatory, and a sequence is automatically created and associated with the table.

Then (depending on how exactly the identity is defined) the sequence is automatically used to produce values for the identity column during INSERT statements.

In order to achieve the same functionality in pre-12c versions, we would manually create a sequence and a BEFORE INSERT trigger. The trigger would get the next value of the sequence and assign it to the column.

For example, the following statement creates the table T in a way that will make INSERT statements populate the column C1 automatically with unique values, taken from an implicitly created sequence. Any attempt to manually set values to the C1 column will be rejected:

```
CREATE TABLE T (
  C1 NUMBER GENERATED ALWAYS AS IDENTITY,
  C2 VARCHAR2(100),
  C3 DATE
);
```

In-Database Archiving

In Oracle 12c tables can be defined as ROW ARCHIVAL. As a result, a hidden column is implicitly added to the table, holding an archiving ("logical deletion") state. Based on a session-level parameter, "Archived" rows are either visible or not.

In order to achieve the same functionality in pre-12c versions, we would manually add a column to the table, for holding the state of the row (whether it is "archived" or not). Then we would create a view to expose only the live (non-archived) rows of the table, add use this view rather than the table in every relevant SQL statement.

For example, the following statement creates the table T with four columns: the explicitly specified C1, C2 and C3, and the implicit ORA_ARCHIVE_STATE:

```
CREATE TABLE T (
  C1 NUMBER,
  C2 VARCHAR2(100),
  C3 DATE
)
ROW ARCHIVAL;
```

The following will return only the records of T in which ORA_ARCHIVE_STATE=0:

```
ALTER SESSION SET ROW ARCHIVAL VISIBILITY = ACTIVE;
-- Note:
-- ACTIVE is the default setting of ROW ARCHIVAL VISIBILITY
SELECT C1,C2,C3 FROM T;
```

The following will return all the records of T, regardless the value of ORA_ARCHIVE_STATE:

```
ALTER SESSION SET ROW ARCHIVAL VISIBILITY = ALL;
SELECT C1,C2,C3 FROM T;
```

Temporal Validity

In Oracle 12c a table can be defined with the PERIOD FOR clause, to associate it with one or more valid time dimensions. Each such dimension consists of two date/time columns (representing the period start time and the period end time). These columns can be created either explicitly or implicitly. As a result, the data can become visible or invisible automatically, based on specific valid times (or ranges), and a session-level configuration or the statement-level AS OF and VERSIONS BETWEEN clauses.

In order to achieve the same functionality in pre-12c versions, we would manually add pairs of columns to the table to represent the period boundaries. Then we would create a view (or several different views) to expose only the relevant rows for the required time points or ranges, add use this view rather than the table in every relevant SQL statement.

For example, the following statement creates the table T with two hidden timestamp columns - **ACTIVE_PERIOD**_START and **ACTIVE_PERIOD**_END - which can (and should) be set explicitly when records of T are inserted and updated:

```
CREATE TABLE T (
  C1 NUMBER,
  C2 VARCHAR2(100),
  C3 DATE,
  PERIOD FOR ACTIVE_PERIOD
);
```

The following query returns only records of T that are currently "active", i.e. records in which **ACTIVE_PERIOD**_START ≤ SYSTIMESTAMP ≤ **ACTIVE_PERIOD**_END (a NULL value in ACTIVE_PERIOD_START/END means there is no limit):

```
SELECT * FROM T AS OF PERIOD FOR ACTIVE_PERIOD SYSTIMESTAMP;
```

While this last example shows a statement-level control, the next one shows a session-level control. The call to DBMS_FLASHBACK_ARCHIVE.ENABLE_AT_VALID_TIME affects all the subsequent queries in the session - in this case showing only records of T that are currently "active":

```
BEGIN
    DBMS_FLASHBACK_ARCHIVE.ENABLE_AT_VALID_TIME('CURRENT');
END;
/

SELECT * FROM T;
```

Session-Level Sequences

In Oracle 12c a non-persistent sequence can be defined, for getting a range of numbers which is unique within a session.

In order to achieve the same functionality in pre-12c versions, we would create a package with a global variable and a function that increments the global variable and returns its value.

For example, the following statement creates a session-level sequence called SEQ:

```
CREATE SEQUENCE SEQ SESSION;
```

Write Less Code in SQL Statements

The following features add new capabilities to the SQL language by extending its syntax.

Row Limiting

In Oracle 12c the new Row Limiting clause can be added to the end of SELECT statements, to allow fetching a specific "page" of records. The size of the page is defined by either an absolute number of records or a specific percent of records out of the complete result set. It can be defined whether the page starts from the first record of the original result set or from some offset. It can also be defined how to treat "ties" (i.e., when several records with the same value are on the borders of the page).

In order to achieve the same functionality in pre-12c versions, a less clear syntax - and often quite cumbersome - should be used.

For example, the following queries return the third page of records from the table T, where each page contains 5 records (i.e., records 11-15), after sorting all the records in the table by the C1 column.

This query uses the new Oracle 12c Row Limiting clause:

```
SELECT C1,C2,C3 FROM T ORDER BY C1
OFFSET 10 ROWS
FETCH NEXT 5 ROWS ONLY;
```

And this query is a pre-12c alternative:

```
SELECT C1,C2,C3 FROM (
  SELECT C1,C2,C3,ROWNUM RN FROM (
    SELECT C1,C2,C3 FROM T ORDER BY C1)
  WHERE ROWNUM <= 15)
WHERE RN > 10;
```

Lateral Inline Views

In Oracle 12c an inline view in a query may be defined as LATERAL. This allows the inline view to refer to other tables that appear to its left in the FROM clause - something that cannot be done in a "regular" inline view, and (when appropriate) allows writing complex queries in a relatively concise way.

In order to achieve the same functionality in pre-12c versions, we could use Collection Unnesting, which requires:

- Creating dedicated schema-level user-defined types
- Converting the content of the inline view to a collection, using the MULTISET operator
- Unnesting the collection to "regular" records, using the TABLE operator

For a detailed example from a real-life use case, including the implementations in versions 11g and 12c, see: http://www.db-oriented.com/2013/08/10/tip004.

Write Less "Inappropriately-Located" Code

The following features provide new functionality that requires less "supporting" code than before, and allows for placing ad-hoc code in a more appropriate scope than in the past.

SELECT FROM Package-Level Collections

In Oracle 12c it is possible to select from a package-level collection, using the TABLE operator. This allows the narrowing of the written code to the appropriate scope (e.g., the scope of a package rather than the scope of the schema).

In order to achieve the same functionality in pre-12c versions, we had to create a schema-level collection type and use it, even if this type is solely used by this package.

PL/SQL in the WITH Clause

In Oracle 12c the WITH clause can include not only subquery factoring but also PL/SQL declarations of functions that can be used in the query (and procedures that can be used in those functions). This allows for embedding ad-hoc functions, that are relevant only for the context of a specific SQL statement, in the statement itself.

In order to achieve the same functionality in pre-12c versions, we would create a stored function (either in a package or standalone), whose scope is the whole schema, and call the function from within the SQL statement.

Conclusion

In this presentation we've seen several new features that can make our life as developers easier. There are many additional features, that due to time constraints could not be covered in the presentation, including, for example:

- Pattern Matching
- Extended Strings
- The ON NULL clause
- Oracle syntax outer join with multiple tables
- Multiple partitions in a single DDL
- Transaction Guard
- And more...

To get more information about the features - the ones that were presented as well as the others - please don't hesitate to contact me (see contact details below).

Contact address:

Oren Nakdimon

DB Oriented
POB 145
Zurit 2010400
Israel

Phone: +972(0)54-4393763
Email oren@db-oriented.com
Internet: www.db-oriented.com
Twitter: @DBoriented