New Oracle 12c Features for Developers

By: Chris Ostrowski

Table of Contents

Overview

In June of 2013, Oracle released the latest version of its flagship database product, Oracle 12c. Oracle 12c contains a plethora of new and amazing features, including the introduction of multi-tenant architecture, container and pluggable database instances, heat maps, Exadata enhancements – the list goes on and on. While many of the Oracle 12c enhancements are geared towards the administration and architecture of how Oracle is used within your organization, there are also new features aimed at making developer's lives easier and more productive. This paper discusses some of the most important enhancements that developers working with Oracle 12c will find useful.

THE "BIG 6"

The main developer enhancements in 12c

Certainly, there are more than six Oracle 12c enhancements that can be classified as "developer" enhancements, but there are six major ones that will be discussed in this paper. We will also cover a seventh enhancement that is not technically a developer enhancement and an eighth that makes working with the data in an Oracle database a lot easier. But let's start with the Big 6.

row_limiting_clause

In virtually every beginner SQL course, there is a problem where the student is expected to solve a "Top-N" problem, where the solution involves returning the top-N number of rows that satisfies some sort of arbitrary condition. For example, let's say I want to return the first 5 rows of employees alphabetically by first name. My employees table looks something like this:

```
SQL> select first_name from hr.employees order by 1;
FIRST_NAME
--------------------
Adam
Alana
Alberto
Alexander
Alexander
Alexis
Allan
....
```

Most students (having already been introduced to the ROWNUM keyword) usually try solving the problem like this:

```
SQL> select first_name from hr.employees
  2 where rownum < 6
  3 order by 1;
FIRST_NAME
--------------------
David
Ellen
Hermann
Mozhe
Sundar
```

But as you can see, this doesn't work as Oracle first selects the rows, THEN orders them. This type of problem can be surprisingly difficult to solve.

Oracle 12c, however, introduces new SELECT clauses: FETCH, OFFSET, and PERCENT. These new clauses allow you to easily find and solve Top-N type queries easily. Let's look at the FETCH keyword first.

FETCH allows you to define a query where only the first or last N number of rows are returned. Let's look at a simple query:

```
SQL> select first_name from hr.employees
  2 order by 1
  3 fetch first 5 rows only;
FIRST_NAME
--------------------
Adam
Alana
Alberto
Alexander
Alexander
```

As you can see, unlike the earlier query that used the ROWNUM clause, FETCH accurately returns the first five rows ordered by first_name. We can also use the LAST keyword to return the last N number of rows from a SELECT:

```
SQL> select first_name from hr.employees
  2 order by 1
  3 fetch last 5 rows only;
FIRST_NAME
--------------------
Valli
Vance
William
William
Winston
```

The OFFSET keyword can be used to start the returned rows "offset" by a number. In this example, we use the FIRST keyword to start at the beginning, but also use the OFFSET keyword to start at row 3:

```
SQL> select first_name from hr.employees
  2 order by 1
  3 offset 2 rows
  4 fetch first 3 rows only;
FIRST_NAME
--------------------
Alberto
Alexander
Alexander
```

New sizes for datatypes

Developers and database designers using the VARCHAR2 datatype in their Oracle databases have been limited to 4000 bytes. On top of the obvious limitation of not being able to use a free-form text field greater than 4000 bytes, there was often additional confusion with PL/SQL developers who could define VARCHAR2 variables in their code that had a maximum of 32k (32,767 bytes). In Oracle 12c, this confusion is eliminated by allowing VARCHAR2, NVARCHAR2 and RAW native Oracle datatypes a maximum length of 32,767 bytes.

Unfortunately, this functionality does not come directly out of the box; DBAs need to set an initialization parameter (MAX_STRING_SIZE) before these new datatypes can be used:

MAX_STRING_SIZE:

- **STANDARD** = size limits for releases prior to Oracle Database 12c apply: 4000 bytes for the VARCHAR2 and NVARCHAR2 data types, and 2000 bytes for the RAW data type
- **EXTENDED** = size limit is 32,767 bytes for the VARCHAR2, NVARCHAR2, and RAW data types

There are also these warnings from the Oracle documentation:

- Setting MAX_STRING_SIZE = EXTENDED may update database objects and possibly invalidate them
- A VARCHAR2 or NVARCHAR2 data type with a declared size of greater than 4000 bytes, or a RAW data type with a declared size of greater than 2000 bytes, is an extended data type. Extended data type columns are stored out-of-line, leveraging Oracle's LOB technology. The use of LOBs as a storage mechanism is internal only. Therefore, you cannot manipulate these LOBs using the DBMS_LOB package.

PL/SQL functions in the WITH clause

Sometimes, a developer needs a quick down-and-dirty procedure or function without the overhead of a package and all of the development that goes along with it. The WITH clause allows developers to quickly create a function that can be used within a SELECT statement, optimizing developer productivity and overall performance of the Oracle database. Here's a brief example:

```
WITH
FUNCTION quadruple(p_num IN NUMBER) IS
BEGIN
RETURN p_num*4;
END;
SELECT quadruple(7)
FROM DUAL;
DUAL
----------

28
```

The UTL_CALL_STACK package

Oracle 12c provides a new package that gives developers a rich set of tools for advanced debugging of their code. The UTL_CALL_STACK package provides information about currently executing subprograms. Functions in the package return subprogram names, unit names, owner names, edition names, line numbers for given dynamic depths and error stack information. In his blog, Tom Kyte provides an excellent introduction to some of the features in this package:

http://tkyte.blogspot.com.au/2013/06/12c-utlcallstack.html

The functions in this package can be summarized as follows:

Identity columns and new DEFAULT functionality

In most Oracle databases, a sequence is used to generate unique values for a primary key in a table. Oftentimes, a BEFORE INSERT trigger is used to select a sequence value and that value is used as the primary key value during an INSERT. Wouldn't it be nice to not have to go through the sequence and trigger definitions when defining a primary key?

Subprogram	Description
BACKTRACE_DEPTH Function	Returns the number of backtrace items in the backtrace
BACKTRACE_LINE Function	Returns the line number of the unit at the specified backtrace depth
BACKTRACE_UNIT Function	Returns the name of the unit at the specified backtrace depth
CURRENT_EDITION Function	Returns the current edition name of the unit of the subprogram at the specified dynamic depth
CONCATENATE_SUBPROGRAM Function	Returns a concatenated form of a unit-qualified name
DYNAMIC_DEPTH Function	Returns the number of subprograms on the call stack
ERROR_MSG Function	Returns the error message of the error at the specified error depth
ERROR_NUMBER Function	Returns the error number of the error at the specified error depth

Subprogram	Description
LEXICAL_DEPTH Function	Returns the lexical nesting level of the subprogram at the specified dynamic depth
OWNER Function	Returns the owner name of the unit of the subprogram at the specified dynamic depth
UNIT_LINE Function	Returns the line number of the unit of the subprogram at the specified dynamic depth
SUBPROGRAM Function	Returns the unit-qualified name of the subprogram at the specified dynamic depth

With Oracle 12c, this is now possible via the IDENTITY keyword. In the following example, a table is created with column C1 defined as an identity column:

```
CREATE TABLE t1 (c1 NUMBER GENERATED BY DEFAULT ON NULL AS IDENTITY,
        c2 VARCHAR2(10));
INSERT INTO t1 (c2) VALUES ('abc');
INSERT INTO t1 (c1, c2) VALUES (null, 'xyz');
```

Column C1 will be automatically populated with values as records are inserted into the table. You are not limited to starting at 1 and incrementing by 1. In this example, a table is created with the first INSERT starting at 10 and each subsequent INSERT incrementing by 10:

```
CREATE TABLE t1 (c1 NUMBER GENERATED BY DEFAULT ON NULL AS IDENTITY
(START WITH 10 increment by 10),
        c2 VARCHAR2(10));
```

In a similar vein, the DEFAULT keyword on a table can now include the sequence pseudocolumns CURRVAL and NEXTVAL, as long as the sequence exists and you have the privileges necessary to access it. Users who perform subsequent inserts that use the DEFAULT expression must have the INSERT privilege on the table and the SELECT privilege on the sequence. Here's an example:

```
CREATE TABLE t1 (
  id          NUMBER DEFAULT t1_seq.NEXTVAL,
  description VARCHAR2(30)
);
```

The ACCESSIBLE clause

For small PL/SQL-based applications, the security model built into PL/SQL works pretty well. But what happens when a project gets really large? With a large amount of packages, procedures, and functions, it can get pretty challenging to keep track of what program unit is calling what package, procedure, or

function. Oracle 12c introduces another mechanism by which developers can control and restrict which PL/SQL program unit calls another – the ACCESSIBLE clause. It works like this:

The developer can create a 'white listing' of other PL/SQL program units that can call a particular procedure, function, or package by specifying the list:

```
CREATE PROCEDURE ADD_EMPLOYEE ACCESSIBLE BY (ONBOARDING_PKG);
```

In this way, even though a particular user may have access to a package (or schema), a restriction can be put in place that stops unauthorized code from accessing the PL/SQL program unit in question.

Those are the Big 6.

TWO MORE IMPORTANT 12C ENHANCEMENTS

There are two more important Oracle 12c enhancements that you should be aware of as a developer:

Enhanced DDL Online

While this is not technically a developer feature, it is important for developers to know: many schema-level DDL maintenance commands no longer have blocking locks. This means you can now perform the following functions on-line:

```
DROP INDEX ONLINE
DROP CONSTRAINT ONLINE
SET UNUSED COLUMN ONLINE
ALTER INDEX VISIBLE
ALTER INDEX INVISIBLE
SET UNUSED COLUMN ONLINE
```

This is just a partial list – there are even more DDL operations you can now perform on-line in Oracle 12c.

TRUNCATE TABLE CASCADE

This is a personal favorite. You can now specify CASCADE as part of your TRUNCATE TABLE statements. As you can probably guess, Oracle truncates all child tables that reference a table with an enabled ON DELETE CASCADE referential constraint. This is a recursive operation that will truncate all child tables, grandchild tables, and so on, using the specified options. Be exceptionally careful when using this – TRUNCATE statements cannot be undone (without a lot of work, at least)!

Conclusion

Oracle has included a tremendous amount of new features for Oracle 12c. The enhancements that get the most attention are those related to DBA and architecture design, but Oracle hasn't forgotten about developers! The development enhancements make it easy for developers to focus on delivering world-class applications and not spending their efforts on arcane and difficult-to-maintain SQL.

About the Author

Chris Ostrowski is the Oracle Solution Architect Director at Avout. He has worked with Oracle technologies for over 20 years as a Developer, DBA, Project Manager, and Enterprise Architect. He is a certified Oracle SOA Implementation Champion and an Oracle ACE, well-versed in assembling complex, end-to-end solutions spanning multiple competencies and platforms.

Recently, Chris has focused his efforts on service oriented architecture (SOA) technologies including Oracle JDeveloper and the Oracle SOA Suite, and enterprise technologies including Oracle Fusion Applications and Oracle's Application Integration Architecture. He is the author of three books from Oracle Press: Oracle Application Server 10g Web Development, Oracle Application Server Portal Handbook, and the soon-to-be-released Migrating to Fusion Applications. He has written articles for technology publications including Update, Select, Oracle Magazine, and the Oracle blog, and frequently presents at Oracle OpenWorld and various Oracle User Groups. Prior to joining Avout, Chris worked as a Director and Oracle Solution Architect for Fujitsu Consulting.

He holds a Bachelor of Arts from Rutgers University. In his spare time he is an avid hockey fan and plays the guitar.

If you have any questions about Oracle 12c, please contact Chris.Ostrowski@avout.com.