Servicio Andaluz de Salud
CONSEJERÍA DE SALUD

Oficina Técnica para la Gestión y Supervisión de Servicios TIC
Subdirección de Tecnologías de la Información

Funcionalidades de PL/SQL de Oracle RDBMS 11g y 12c

Referencia documento: InfV5_JASAS_PLSQL_FeaturesPerVersion_V920.doc
Fecha: Fecha: 16 de noviembre de 2018
Versión: 9.2.0

Registro de Cambios

Fecha	Autor	Versión	Notas
12 de Enero de 2012	Oracle ACS	3.1.	Versión inicial
14 de marzo de 2013	Oracle ACS	4.1	Revisión de marzo de 2013, contrato 2012-2014
13 de junio de 2013	Oracle ACS	4.2	Revisión de junio de 2013, contrato 2012-2014
17 de octubre de 2013	Oracle ACS	4.3	Revisión de octubre de 2013, contrato 2012-2014
16 de Julio de 2015	Paola Juárez	6.1	Revisión de Julio de 2015, contrato 2014-2016
16 de diciembre de 2015	Paola Juárez	6.2	Revisión de diciembre de 2015, contrato 2014-2016
16 de junio de 2016	Paola Juárez	7.1	Revisión de junio de 2016, contrato 2014-2016
16 de junio de 2017	Paola Juárez	8.1	Revisión de junio de 2016, contrato 2016-2018
16 de noviembre de 2017	Paola Juárez	8.2	Revisión de noviembre de 2017, contrato 2016-2018
16 de junio de 2018	Paola Juárez	9.1	Revisión de junio de 2018, contrato 2016-2018
16 de noviembre de 2018	Paola Juárez	9.2	Revisión de noviembre de 2018, contrato 2016-2018

Revisiones

Nombre	Role
Gregorio Adame	Oracle ACS Service Engineer
Jonathan Ortiz	Oracle ACS Service Engineer
José María Gómez	Oracle Technical Account Manager

Distribución

Copia	Nombre	Empresa
1	Subdirección de Tecnologías de la Información	Consejería de Salud, Junta de Andalucía
2	Servicio de Coordinación de Informática de la Consejería de Hacienda y Administración Pública	Consejería de Hacienda y Administración Pública, Junta de Andalucía

Índice de Contenidos

CONTROL DE CAMBIOS 5
INTRODUCCIÓN 6
CARACTERÍSTICAS INTRODUCIDAS EN 11GR1 7
Mejoras en expresiones regulares construídas con funciones SQL 7
Tipos de datos SIMPLE_INTEGER, SIMPLE_FLOAT y SIMPLE_DOUBLE 7
Sentencia CONTINUE 8
Secuencias en Expresiones PL/SQL 9
Mejoras en SQL Dinámico 10
Notación por nombre y mixta en invocacaciones a subprogramas PL/SQL 10
Función PL/SQL Result Cache 11
Triggers Compuestos 12
Mayor control sobre Triggers 14
Database Resident Connection Pool 14
Inlining automático en subprogramas 15
PL/Scope 15
PL/SQL hierarchical profiler 16
Generación diracta de código nativo 16
CARACTERÍSTICAS INTRODUCIDAS EN 11GR2 17
Tratamiento de paquetes como "sin estado" 17
Paquete DBMS_PARALLEL_EXECUTE 17
CREATE TYPE con la opción FORCE 18
Triggers crossedition 18
Restricciones de la sentencia ALTER TYPE para ADTs con versiones 19
Opción RESET para la sentencia ALTER TYPE 19
Detección Automática de datos Result-Cache 19
Los resultados cacheados en entornos RAC 20
CARACTERÍSTICAS INTRODUCIDAS EN 12CR1 21
Las funciones de derechos del invocador se pueden almacenar en Result-Cached 21
Más tipos de datos PL/SQL-only pueden cruzar de la interfaz PL/SQL al interfaz SQL 21
Nueva Clausula ACCESSIBLE BY 21
Clausula FETCH FIRST 22
Grant Roles a paquetes PL/SQL Packages y programas Standalone 22
Mismo tamaño máximo en SQL y PL/SQL en más tipos de datos 23
Objetos LIBRARY 23
Resultados de declaraciones implícitas 24
Vistas BEQUEATH CURRENT_USER 24
Privilegios INHERIT PRIVILEGES y Privilegios INHERIT ANY PRIVILEGES 25
Columnas Invisibles 25
Objetos, No types, son Editioned o Noneditioned 25
Las Funciones PL/SQL de ejecución más rápida en SQL 26
Directivas de consulta predefinidas $$PLSQL_UNIT_OWNER y $$PLSQL_UNIT_TYPE 26
Obsolescencia del parámetro PLSQL_DEBUG (Deprecated) 26

CARACTERÍSTICAS INTRODUCIDAS EN 12CR2 27
Mejoras de la cláusula ACCESSIBLE BY *27*
Intercalación de datos limitados (Data-Bound Collation) *27*
Controlar los privilegios de los derechos del definidor para procedimientos remotos *28*
Mejoras de expresiones PL / SQL *28*
Soporte para operadores SQL JSON en PL / SQL *28*
Soporte para identificadores más largos *29*
Compartición de objetos comunes de aplicaciones vinculadas a metadatos *29*
Funcionalidades "Deprecated" *29*
Funcionalidades Desoportadas *30*

Control de cambios

Cambio	Descripción	Página
	No se realizan cambios en esta versión del documento.	N/A

Introducción

Este documento recoge una lista de funcionalidades *PL/SQL* por versión de *Oracle RDBMS*, concretamente de las versiones *11gR1 y 11gR2 y 12c.*

El objetivo es que sirva de base para que los procesos de actualización de base de datos que incorporen código *PL/SQL* actualicen de igual forma dicho código adoptando estas nuevas funcionalidades.

En ningún momento este documento es un documento de buenas practicas ni de recomendaciones, y por tanto debe tomarse exclusivamente como una guía de funcionalidades.

Características introducidas en 11gR1

Estas son las nuevas características para Oracle Database 11g Release 1 (11.1).

Mejoras en expresiones regulares construídas con funciones *SQL*

Las expresiones regulares construídas con *REGEXP_INSTR* y *REGEXP_SUBSTR* han incrementado su funcionalidad. Una expresión regular construída con la función *REGEXP_COUNT,* devuelve el número de veces que un patrón aparece en una cadena. Estas funciones actúan del mismo modo en *SQL* que en *PL/SQL*.

```
SELECT REGEXP_COUNT('123123123123123', '(12)3', 1, 'i') REGEXP_COUNT
   FROM DUAL;

REGEXP_COUNT
------------
           5
```

Más información en REGEXP_INSTR, REGEXP_SUBSTR, y REGEXP_COUNT.

Tipos de datos SIMPLE_INTEGER, *SIMPLE_FLOAT* y *SIMPLE_DOUBLE*

Los tipos de datos *SIMPLE_INTEGER, SIMPLE_FLOAT*, y *SIMPLE_DOUBLE* son subtipos predefinidos de *PLS_INTEGER*, *BINARY_FLOAT*, y BINARY_*DOUBLE,* respectivamente. Cada subtipo tiene el mismo rango que su tipo base y una restricción *NOT NULL*:

SIMPLE_INTEGER y *PLS_INTEGER* difieren significativamente en sus semánticas de desbordamiento, pero *SIMPLE_FLOAT* y *SIMPLE_DOUBLE* son idénticos a sus tipos base, excepto por su restricción *NOT NULL*.

SIMPLE_INTEGER es útil cuando el valor es distinto de *NULL* y la comprobación de desbordamiento es innecesaria. *SIMPLE_FLOAT* y *SIMPLE_DOUBLE* son útiles cuando el valor es distinto de *NULL*. Sin el coste de comprobación de no nulo y desbordamiento, estos subtipos proporcionan mejor rendimiento que sus tipos base cuando *PLSQL _CODE_TYPE='NATIVE'*, ya que las operaciones aritméticas sobre valores *SIMPLE_INTEGER* se hacen directamente en hardware. Cuando *PLSQL_CODE_TYPE='INTERPRETED'*, la mejora de rendimiento es menor.

```
SQL> DECLARE
  2    n SIMPLE_INTEGER := 2147483645;
  3  BEGIN
  4    FOR j IN 1..4 LOOP
  5      n := n + 1;
  6      DBMS_OUTPUT.PUT_LINE(TO_CHAR(n, 'S9999999999'));
  7    END LOOP;
  8    FOR j IN 1..4 LOOP
```

```
  9      n := n - 1;
 10      DBMS_OUTPUT.PUT_LINE(TO_CHAR(n, 'S9999999999'));
 11    END LOOP;
 12  END;
 13  /

+2147483646
+2147483647
-2147483648
-2147483647
-2147483648
+2147483647
+2147483646
+2147483645

PL/SQL procedure successfully completed.
```

Más información sobre esto accede a la siguiente documentación: SIMPLE_INTEGER Subtype of PLS_INTEGER, BINARY_FLOAT and BINARY_DOUBLE Data Types, Use PLS_INTEGER or SIMPLE_INTEGER for Integer Arithmetic, Use BINARY_FLOAT, BINARY_DOUBLE, SIMPLE_FLOAT, and SIMPLE_DOUBLE for Floating-Point Arithmetic.

Sentencia CONTINUE

La sentencia *CONTINUE* sale de la iteración actual de un blucle y transfiere el control a la siguiente iteración (a diferencia de la sentencia *EXIT*, que sale del buble y transfiere el control al final del bucle). La sentencia *CONTINUE* toma dos formas: la incondicional con *CONTINUE* y la condicional con *CONTINUE WHEN*.

```
SQL> DECLARE
  2    x NUMBER := 0;
  3  BEGIN
  4    LOOP -- After CONTINUE statement, control resumes here
  5      DBMS_OUTPUT.PUT_LINE ('Inside loop:  x = ' || TO_CHAR(x));
  6      x := x + 1;
  7
  8      IF x < 3 THEN
  9        CONTINUE;
 10      END IF;
 11
 12      DBMS_OUTPUT.PUT_LINE
 13        ('Inside loop, after CONTINUE:  x = ' || TO_CHAR(x));
 14
 15      EXIT WHEN x = 5;
 16    END LOOP;
 17
 18    DBMS_OUTPUT.PUT_LINE (' After loop:  x = ' || TO_CHAR(x));
 19  END;
 20  /
Inside loop:  x = 0
Inside loop:  x = 1
Inside loop:  x = 2
Inside loop, after CONTINUE:  x = 3
Inside loop:  x = 3
Inside loop, after CONTINUE:  x = 4
Inside loop:  x = 4
Inside loop, after CONTINUE:  x = 5
```

```
After loop:  x = 5

PL/SQL procedure successfully completed.
```

Para más información, ver:

- Controlling Loop Iterations (LOOP, EXIT, and CONTINUE Statements)
- CONTINUE Statement

Secuencias en Expresiones *PL/SQL*

Las pseudocolumnas *CURRVAL* y *NEXTVAL* hacen que escribir código *PL/SQL* sea más fácil y que se mejore el rendimiento y la escalabilidad en tiempo de ejecución. Pueden usarse *sequence_name*. *CURRVAL* y *sequence_name*. *NEXTVAL* donde se esté haciendo uso de una expresión *NUMBER*.

```
CREATE TABLE employees_temp
  AS SELECT employee_id, first_name, last_name
  FROM employees;

CREATE TABLE employees_temp2
  AS SELECT employee_id, first_name, last_name
  FROM employees;

DECLARE
  seq_value NUMBER;
BEGIN
  -- Generate initial sequence NUMBER
  seq_value := employees_seq.NEXTVAL;

  -- Print initial sequence NUMBER:
  DBMS_OUTPUT.PUT_LINE
    ('Initial sequence value: ' || TO_CHAR(seq_value));

  -- Use NEXTVAL to CREATE unique NUMBER when inserting data:
     INSERT INTO employees_temp VALUES (employees_seq.NEXTVAL,
                                        'Lynette', 'Smith');

  -- Use CURRVAL to store same value somewhere else:
     INSERT INTO employees_temp2 VALUES (employees_seq.CURRVAL,
                                         'Morgan', 'Smith');

  -- Because NEXTVAL values might be referenced
  -- by different users and applications,
  -- and some NEXTVAL values might not be stored in the database,
  -- there might be gaps in the sequence.

  -- Use CURRVAL to specify the record to delete:
     seq_value := employees_seq.CURRVAL;
     DELETE FROM employees_temp2 WHERE employee_id = seq_value;

  -- Udpate employee_id with NEXTVAL for specified record:
     UPDATE employees_temp SET employee_id = employees_seq.NEXTVAL
       WHERE first_name = 'Lynette' AND last_name = 'Smith';

  -- Display final value of CURRVAL:
     seq_value := employees_seq.CURRVAL;
```

```
     DBMS_OUTPUT.PUT_LINE
       ('Ending sequence value: ' || TO_CHAR(seq_value));
END;
/
```

Para más información, ver CURRVAL and NEXTVAL.

Mejoras en *SQL* Dinámico

Se ha mejorado tanto *SQL* dinámico nativo como el paquete DBMS_*SQL*. Ahora el *SQL* dinámico nativo soporta sentencias mayores de 32KB permitiendo que éstas sean *CLOB*. Ver EXECUTE IMMEDIATE Statement y OPEN-FOR Statement.

En el paquete DBMS_*SQL*:

- Están soportados todos los tipos de datos soportados por *SQL* nativo dinámico.
- La función DBMS_*SQL.PARSE* acepta un argumento *CLOB*, permitiendo sentencias *SQL* dinámicas mayores de 32 KB.
- La nueva función DBMS_SQL.TO_REFCURSOR permite cambiar de paquete DBMS_*SQL* a *SQL* nativo dinámico.
- La nueva función DBMS_SQL.TO_CURSOR_NUMBER permite cambiar de *SQL* nativo dinámico a paquete *DBMS_SQL*.

Notación por nombre y mixta en invocacaciones a subprogramas *PL/SQL*

Antes de la versión 11.1, una sentencia *SQL* que invocara un subprograma *PL/SQL* tenía que especificar los parámetros reales en notación posicional. A partir de la versión 11.1 también están permitidas la notación por nombre y notación mixta. Esto hace que se mejore la usabilidad cuando una sentencia *SQL* invoca a un subprograma *PL/SQL* que tiene muchos parámetros por defecto, y sólo unos pocos parámetros reales difieren de sus valores por defecto.

```
SQL> DECLARE
  2    emp_num NUMBER(6) := 120;
  3    bonus   NUMBER(6) := 50;
  4    PROCEDURE raise_salary (emp_id NUMBER, amount NUMBER) IS
  5      BEGIN
  6        UPDATE employees SET salary =
  7          salary + amount WHERE employee_id = emp_id;
  8    END raise_salary;
  9  BEGIN
 10    -- Positional notation:
 11    raise_salary(emp_num, bonus);
 12    -- Named notation (parameter order is insignificant):
 13    raise_salary(amount => bonus, emp_id => emp_num);
 14    raise_salary(emp_id => emp_num, amount => bonus);
 15    -- Mixed notation:
 16    raise_salary(emp_num, amount => bonus);
 17  END;
```

```
 18  /

PL/SQL procedure successfully completed.

SQL> REM Clean up
SQL> ROLLBACK;

Rollback complete.

SQL>
SQL> CREATE OR REPLACE FUNCTION compute_bonus (emp_id NUMBER, bonus
NUMBER)
  2    RETURN NUMBER
  3  IS
  4    emp_sal NUMBER;
  5  BEGIN
  6    SELECT salary INTO emp_sal
  7      FROM employees
  8        WHERE employee_id = emp_id;
  9    RETURN emp_sal + bonus;
 10  END compute_bonus;
 11  /

Function created.

SQL> SELECT compute_bonus(120, 50) FROM DUAL;                    --
positional
  2  SELECT compute_bonus(bonus => 50, emp_id => 120) FROM DUAL; --
named
  3  SELECT compute_bonus(120, bonus => 50) FROM DUAL;           --
mixed
  4
SQL>
```

Función PL/SQL Result Cache

Antes de la versión 11.1, si se deseaba que una aplicación *PL/SQL* cacheara los resultados de una función, tenían que diseñarse y codificarse la caché y los subprogramas de gestión de caché. Si varias sesiones ejecutaban una aplicación, cada sesión necesitaba tener su propia copia de caché y de los subprogramas de gestión de caché. En algunas ocasiones cada sesión tenía que realizar de nuevo los mismos cálculos con un alto coste.

Una función result cache puede ahorrar tiempo y espacio significativamente. Cada vez que se invoca una función result-cache con parámetros diferentes, esos parámetros y su resultado se guardan en caché. Posteriormente, cuando la misma función es invocada con los mismos valores de parámetros, el resultado se extrae de la caché en lugar de calcularse de nuevo.

A partir de la versión 11.1, *PL/SQL* incorpora la función result cache. Para usarla, hay que usar la calúsula *RESULT_CACHE* en cada función *PL/SQL* cuyo resultado se desee cachear. Como la función result cache se almacena en la shared global area (*SGA*), estará disponible para cualquier sesión que ejecute la aplicación.

Si se incorpora a una aplicación la función *PL/SQL* result cache, usará más *SGA*, pero reducirá significativamente el uso de memoria total de sistema.

```
-- Package specification
CREATE OR REPLACE PACKAGE department_pks IS
  TYPE dept_info_record IS RECORD (average_salary      NUMBER,
                                   NUMBER_of_employees NUMBER);
  -- Function declaration
  FUNCTION get_dept_info (dept_id NUMBER) RETURN dept_info_record
  RESULT_CACHE;
END department_pks;
/
CREATE OR REPLACE PACKAGE BODY department_pks AS
  -- Function definition
  FUNCTION get_dept_info (dept_id NUMBER) RETURN dept_info_record
    RESULT_CACHE RELIES_ON (EMPLOYEES)
  IS
    rec dept_info_record;
  BEGIN
    SELECT AVG(SALARY), COUNT(*) INTO rec
      FROM EMPLOYEES
        WHERE DEPARTMENT_ID = dept_id;
    RETURN rec;
  END get_dept_info;
END department_pks;
/
DECLARE
  dept_id   NUMBER := 50;
  avg_sal   NUMBER;
  no_of_emp NUMBER;
BEGIN
  avg_sal   := department_pks.get_dept_info(50).average_salary;
  no_of_emp := department_pks.get_dept_info(50).NUMBER_of_employees;
  DBMS_OUTPUT.PUT_LINE('dept_id = ' ||dept_id);
  DBMS_OUTPUT.PUT_LINE('average_salary = '|| avg_sal);
  DBMS_OUTPUT.PUT_LINE('NUMBER_of_employees = ' ||no_of_emp);
END;
/
```

Para más información, ver:

- Using the *PL/SQL* Function Result Cache
- Table 13-0, "Function Declaration and Definition"

Triggers Compuestos

Un *trigger* compuesto es un *trigger DML* que puede lanzarse en más de un punto del tiempo. El cuerpo de un *trigger* compuesto es compatible con el estado normal de *PL/SQL*, en el que puede accederse a todas las secciones del código. El estado normal se establece cuando la sentencia del *trigger* se lanza y termina cuando la sentencia *trigger* acaba, incluso cuando la sentencia *trigger* provaca un error.

Antes de la versión 11.1, los desarrolladores de aplicaciones modelaban el estado normal con un paquete auxiliar. Este enfoque era complicado de programar y estaba sujeto a pérdidas de memoria cuando la sentencia *trigger* causaba un error y la sentencia siguiente al *trigger* no se lanzaba. Los triggers compuestos hacen que sea

más fácil programar un enfoque donde se desea que las acciones inplementadas para puntos de tiempo diferentes compartan datos comunes.

```
CREATE TABLE employee_salaries (
  employee_id NUMBER NOT NULL,
  change_date DATE   NOT NULL,
  salary NUMBER(8,2) NOT NULL,
  CONSTRAINT pk_employee_salaries PRIMARY KEY (employee_id,
change_date),
  CONSTRAINT fk_employee_salaries FOREIGN KEY (employee_id)
    REFERENCES employees (employee_id)
      ON DELETE CASCADE)
/
CREATE OR REPLACE TRIGGER maintain_employee_salaries
  FOR UPDATE OF salary ON employees
    COMPOUND TRIGGER

-- Declarative Part:
-- Choose small threshhold value to show how example works:
  threshhold CONSTANT SIMPLE_INTEGER := 7;

  TYPE salaries_t IS TABLE OF employee_salaries%ROWTYPE INDEX BY
SIMPLE_INTEGER;
  salaries  salaries_t;
  idx       SIMPLE_INTEGER := 0;

  PROCEDURE flush_array IS
    n CONSTANT SIMPLE_INTEGER := salaries.count();
  BEGIN
    FORALL j IN 1..n
      INSERT INTO employee_salaries VALUES salaries(j);
    salaries.delete();
    idx := 0;
    DBMS_OUTPUT.PUT_LINE('Flushed ' || n || ' rows');
  END flush_array;

  -- AFTER EACH ROW Section:

  AFTER EACH ROW IS
  BEGIN
    idx := idx + 1;
    salaries(idx).employee_id := :NEW.employee_id;
    salaries(idx).change_date := SYSDATE();
    salaries(idx).salary := :NEW.salary;
    IF idx >= threshhold THEN
      flush_array();
    END IF;
  END AFTER EACH ROW;

  -- AFTER STATEMENT Section:

  AFTER STATEMENT IS
  BEGIN
    flush_array();
  END AFTER STATEMENT;
END maintain_employee_salaries;
/
/* Increase salary of every employee in department 50 by 10%: */

UPDATE employees
  SET salary = salary * 1.1
  WHERE department_id = 50
```

```
/

/* Wait two seconds: */

BEGIN
  DBMS_LOCK.SLEEP(2);
END;
/

/* Increase salary of every employee in department 50 by 5%: */

UPDATE employees
  SET salary = salary * 1.05
  WHERE department_id = 50
/
```

Para más información, ver Compound Triggers.

Mayor control sobre Triggers

A partir de ahora, la sentencia SQL CREATE TRIGGER admite las claúsulas ENABLE, DISABLE, y FOLLOWS que proporcionan un mayor control sobre los triggers.

La claúsula DISABLE permite crear un trigger en estado desabilitado, de esta forma se garantiza que el código compila satisfactoriamente antes de habilitar el trigger.

La claúsula ENABLE especifica el estado por defecto. La claúsula FOLLOWS permite controlar el orden en que se lanzan los triggers que están definidos en la misma tabla y en el mismo punto del tiempo.

Para más información, ver:

- Ordering of Triggers
- Enabling Triggers
- Disabling Triggers

Ver también: *CREATE TRIGGER* Statement

Database Resident Connection Pool

El paquete DBMS_CONNECTION_POOL está destinado a la gestión del Database Resident Connection Pool, que es compartido por varios procesos middle-tier. El administrador de base de datos utiliza procedimientos de DBMS_CONNECTION_POOL para arrancar y parar el Database Resident Connection Pool y para configurar parámetros del pool como el tamaño y el límite de tiempo.

```
DBMS_CONNECTION_POOL.CONFIGURE_POOL (
   pool_name                 IN VARCHAR2 DEFAULT
'SYS_DEFAULT_CONNECTION_POOL',
   minsize                   IN NUMBER   DEFAULT 4,
   maxsize                   IN NUMBER   DEFAULT 40,
   incrsize                  IN NUMBER   DEFAULT 2,
   session_cached_cursors    IN NUMBER   DEFAULT 20,
   inactivity_timeout        IN NUMBER   DEFAULT 300,
   max_think_time            IN NUMBER   DEFAULT 120,
   max_use_session           IN NUMBER   DEFAULT 500000,
   max_lifetime_session      IN NUMBER   DEFAULT 86400);
```

Para más información, ver *DBMS_CONNECTION_POOL* Package.

Inlining automático en subprogramas

Un subprograma inline substituye la llamada al subprograma (a un subprograma en la misma unidad *PL/SQL*) por una copia del subprograma llamado, lo que hace que casi siempre se mejore el rendimiento.

Puede usarse *PRAGMA INLINE* para especificar que una llamada a un subprograma sea o no sea inline. También puede activarse inlining automático—que es, solicitar al compilador que busque llamadas donde sea posible hacer inline—estableciendo el parámetro de compilación *PLSQL_OPTIMIZE_LEVEL* a 3 (por defecto es 2).

En el caso poco frecuente de que con inlining automático no se mejore el rendimiento, puede usarse el *hierarchical profiler* de *PL/SQL* para identificar los subprogramas para los que se podría desactivar inlining.

```
PROCEDURE p1 (x PLS_INTEGER) IS ...
...
PRAGMA INLINE (p1, 'YES');
x:= p1(1) + p1(2) + 17;    -- These 2 calls to p1 will be inlined
...
x:= p1(3) + p1(4) + 17;    -- These 2 calls to p1 will not be inlined
...
```

Para más información, ver:

- How PL/SQL Optimizes Your Programs
- INLINE Pragma

PL/Scope

PL/Scope es una herramienta de optimización de código que recopila y organiza información sobre los identificadores definidos por el usuario en el código *PL/SQL*. Como es una herramienta de óptimización de código, se utiliza a través de entornos de desarrollo interactivo (como *SQL* Developer y JDeveloper) en lugar de directamente.

PL/Scope da la posibilidad de desarrollar potentes navegadores de código y así incrementar la productividad del desarrollador *PL/SQL* minimizando el tiempo invertido en la navegación y el entendimiento del código.

Para más información, ver Collecting Data About User-Defined Identifiers.

PL/SQL hierarchical profiler

El *PL/SQL hierarchical profiler* informa del perfil de ejecución dinámica del programa *PL/SQL*, organizando la información en llamadas a subprogramas. Considera los tiempos de ejecución para *SQL* y *PL/SQL* por separado. Cada informe a nivel de subprograma incluye información como el número de llamada, tiempo invertido en el subprograma en sí, tiempo invertido en el subárbol del subprograma (es decir, en sus subprogramas descendientes), e información detallada sobre padres-hijos.

Puede generarse un informe *HTML* compatible con cualquier navegador. Las capacidades de navegación del navegador combinado con enlaces bien elegidos proporcionan una forma eficiente de analizar el rendimiento de aplicaciones grandes, mejorar el rendimiento de aplicaciones, y menores costes de desarrollo.

Para más información, ver Profiling and Tracing *PL/SQL* Programs.

Ver también:
Oracle Database Advanced Application Developer's Guide

Generación diracta de código nativo

A partir de ahora, el compilador nativo de *PL/SQL* genera directamente código nativo, en lugar de tener que traducir a C. Esto permite que el proceso se acelere en tiempo, así como que los desarrolladores pueden compilar nativamente sin necesidad de la intervención de los DBA.

Para más información, ver Compiling PL/SQL Units for Native Execution.

Características introducidas en 11gR2

Estas son las nuevas características para Oracle Database 11*g* Release 2 (11.2).

Tratamiento de paquetes como "sin estado"

Hasta la Release 11.2.0.2, si una sesión recompilaba el cuerpo de un paquete con estado, y luego una sesión que había instanciado ese paquete lo referenciaba, la última sesión obtenía el error ORA-04068 ("existing state of packages… has been discarded"). Por tanto, los paquetes "hot patching" podrían perjudicar a los usuarios.

A partir de la Release 11.2.0.2, Oracle Database trata los paquetes como paquetes "sin estado" si su estado es constante durante el tiempo de sesión (o durante un tiempo mayor). Este es el caso de un paquete cuyos elementos permanecen constantes en tiempo de compilación. Por tanto, los paquetes "hot patching" (especialmente los paquetes noneditioned) hacen que sea mucho menos probable que se alteren las sesiones que los están utilizando.

Para más información, ver "Package State".

Paquete DBMS_PARALLEL_EXECUTE

El paquete DBMS_PARALLEL_EXECUTE permite actualizar en paralelo y de forma incremental las tablas con una gran cantidad de datos en dos pasos:

1. Agrupar conjuntos de filas de la tabla en chunks más pequeños.
2. Aplicar la sentencia UPDATE deseada a los chunks en paralelo, haciendo commit cada vez que se procese un *chunk*.

Oracle recomienda esta técnica siempre y cuando se vaya a actualizar una gran cantidad de datos. De esta forma se mejora el rendimiento y se reduce tanto del consumo de espacio de *rollback* como el número de filas bloqueadas.

```
DECLARE
  l_SQL_stmt VARCHAR2(1000);
  l_try NUMBER;
  l_status NUMBER;
BEGIN

  -- CREATE the TASK
  DBMS_PARALLEL_EXECUTE.CREATE_TASK ('mytask');

  -- Chunk the table by ROWID
  DBMS_PARALLEL_EXECUTE.CREATE_CHUNKS_BY_ROWID('mytask', 'HR',
'EMPLOYEES', true, 100);

  -- Execute the DML in parallel
  l_SQL_stmt := 'update /*+ ROWID (dda) */ EMPLOYEES e
      SET e.salary = e.salary + 10
      WHERE rowid BETWEEN :start_id AND :end_id';
  DBMS_PARALLEL_EXECUTE.RUN_TASK('mytask', l_SQL_stmt,
DBMS_SQL.NATIVE,
```

```
                                    parallel_level => 10);

  -- If there is an error, RESUME it for at most 2 times.
  L_try := 0;
  L_status := DBMS_PARALLEL_EXECUTE.TASK_STATUS('mytask');
  WHILE(l_try < 2 and L_status != DBMS_PARALLEL_EXECUTE.FINISHED)
  LOOP
    L_try := l_try + 1;
    DBMS_PARALLEL_EXECUTE.RESUME_TASK('mytask');
    L_status := DBMS_PARALLEL_EXECUTE.TASK_STATUS('mytask');
  END LOOP;

  -- Done with processing; drop the task
  DBMS_PARALLEL_EXECUTE.DROP_TASK('mytask');

END;
/
```

Para más información, ver "Updating Large Tables in Parallel".

CREATE TYPE con la opción FORCE

Hasta la Release 11.2, si una sentencia *CREATE OR REPLACE TYPE* especificaba un tipo existente que tenía dependencia de tipo o dependencia a tabla, la sentencia fallaba con un error ORA-02303. A partir de la Release 11.2, si se especifica *FORCE* en esta situación, la sentencia falla si el tipo existente tiene dependencias de tabla, pero no si tiene dependencias de tipo.

Para más información, ver "CREATE TYPE Statement".

Triggers crossedition

Los triggers crossedition deben lanzarse cuando sentencias *DML* hacen cambios en tablas de la base de datos mientras una aplicación online que usa esa tabla se está parcheando o actualizando con edition-based redefinition.

El cuerpo de un *trigger* crossedition se diseña para manejar esos cambios a fin de que puedan aplicarse adecuadamente después de que se hayan completado los cambios en el código de la aplicación.

Para más información, ver "CREATE TRIGGER Statement".

Ver también:
Oracle Database Advanced Application Developer's Guide para información sobre edition-based redefinition en general y triggers crossedition en particular, incluyendo las relaciones entre triggers crossedition y las versiones.

Restricciones de la sentencia ALTER TYPE para ADTs con versiones

Si se usa edition-based redefinition para parchear o actualizar una aplicación, pueden usarse objetos versionados. Si alguno de los objetos versionados son tipos abstractos de datos (ADTs), ver "Restriction on type".

Ver también:
Oracle Database Advanced Application Developer's Guide para información sobre edition-based redefinition en general y objetos editados en particular

Opción RESET para la sentencia ALTER TYPE

La opción *RESET* de la sentencia *ALTER TYPE* resetea la versión de un tipo a valor 1, por lo que ya no se tiene en cuenta para una posible evoluación.

RESET está destinado a ADTs que impiden a sus propietarios editarlos. Para más información, ver "ALTER TYPE Statement".

Ver también:
Oracle Database Advanced Application Developer's Guide para información sobre habilitar edicion para usuarios

Detección Automática de datos Result-Cache

Antes de la Release 11.2, había que especificar los datos que dependían de una función result-cache. A partir de la Release 11.2, Oracle Database detecta automáticamente los datos requeridos mientras se está ejecutando una función result-cache.

```
CREATE OR REPLACE PACKAGE department_pkg IS

  TYPE dept_info_record IS RECORD (
    dept_name  departments.department_name%TYPE,
    mgr_name   employees.last_name%TYPE,
    dept_size  PLS_INTEGER
  );

  -- Function declaration

  FUNCTION get_dept_info (dept_id PLS_INTEGER)
    RETURN dept_info_record
    RESULT_CACHE;

END department_pkg;
/
CREATE OR REPLACE PACKAGE BODY department_pkg IS
  -- Function definition
  FUNCTION get_dept_info (dept_id PLS_INTEGER)
    RETURN dept_info_record
    RESULT_CACHE RELIES_ON (DEPARTMENTS, EMPLOYEES)
  IS
```

```
    rec  dept_info_record;
  BEGIN
    SELECT department_name INTO rec.dept_name
    FROM departments
    WHERE department_id = dept_id;

    SELECT e.last_name INTO rec.mgr_name
    FROM departments d, employees e
    WHERE d.department_id = dept_id
    AND d.manager_id = e.employee_id;

    SELECT COUNT(*) INTO rec.dept_size
    FROM EMPLOYEES
    WHERE department_id = dept_id;

    RETURN rec;
  END get_dept_info;
END department_pkg;
/
```

Para más información, ver "PL/SQL Function Result Cache".

Los resultados cacheados en entornos RAC

Para Oracle Database 11*g* Release 1 (11.1), cada instancia de base de datos en un entorno *Oracle RAC* tenía una función result cache privada disponible sólo para las sesiones de esa instancia. Si un resultado no se encontraba en la caché privada de la instancia local, el cuerpo de la función se ejecutaba para calcular el resultado y después era añadido al cache local. El resultado no se extraía de la caché privada de otra instancia.

En la Release 11.2, cada instancia de base de datos maneja su propia result cache local, pero ahora el cache local no es privado – las sesiones asociadas a instancias remotas pueden acceder a su contenido. Si un resultado no se encuentra en la result cache de la instancia local, el resultado podría extraerse del cache local de otra instancia, en lugar ejecutarse en local.

Para más información, ver "Result Caches in Oracle RAC Environment".

Características introducidas en 12CR1

Estas son las nuevas características para Oracle Database 12c Release 1 (12.1). La documentación oficial se encuentra publicada en el siguiente enlace:

https://docs.oracle.com/database/121/LNPLS/release_changes.htm#LNPLS105

Las funciones de derechos del invocador se pueden almacenar en Result-Cached

Antes de Oracle Database 12c, una función de derechos del invocador no se podía almacenar en caché. A partir de Oracle Database 12c, esta restricción se ha ido.

Para obtener información sobre las funciones de los derechos del invocador, consulte "Invoker's Rights and Definer's Rights (AUTHID Property)". Para obtener información sobre el almacenamiento en caché de resultados, consulte "PL/SQL Function Result Cache".

Más tipos de datos PL/SQL-only pueden cruzar de la interfaz PL/SQL al interfaz SQL

En las versiones anteriores a 12c, valores de tipo PL/SQL-only (por ejemplo, BOOLEAN, associative array o record) no se podían limitar por los programas del cliente (OCI or JDBC) o desde SQL dinámico estático y nativo emitido desde PL / SQL en el servidor.

Desde Oracle 12c es posible vincular valores con tipos de datos PL/SQL-only a bloques anónimos (que son sentencias SQL), llamadas a funciones PL / SQL en consultas SQL y CALL statements, y el operador TABLE en consultas SQL.

Existen algunas restricciones, para más información consultar la documentación oficial .

Nueva Clausula ACCESSIBLE BY

Es posible implementar una aplicación de base de datos mediante varios paquetes PL / SQL, se necesita un paquete que proporciona la interfaz de programación de aplicaciones (API) y el resto de paquetes de ayuda para hacer el trabajo. Idealmente, solo la API es accesible por los clientes. Además, se puede crear un paquete de

utilidad para proporcionar servicios solo a otras unidades PL / SQL en el mismo esquema.

Antes de Oracle Database 12c, PL / SQL no podía evitar que los clientes utilizaran elementos expuestos en los paquetes auxiliares. Para aislar estos elementos, se debía usar las características de seguridad del sistema de administración de bases de datos relacionales (RDBMS), con la dificultad que esto conlleva.

A partir de Oracle Database 12c, cada una de los siguientes "statemens" tiene una cláusula opcional ACCESSIBLE BY que le permite especificar una lista blanca de unidades PL / SQL que pueden acceder a la unidad PL / SQL que está creando o alterando suplementando los mecanismos de seguridad estándar de Oracle.

"CREATE FUNCTION Statement"

"CREATE PACKAGE Statement"

"CREATE PROCEDURE Statement"

"CREATE TYPE Statement"

"ALTER TYPE Statement"

Clausula FETCH FIRST

La cláusula opcional FETCH FIRST limita el número de filas que devuelve una consulta, lo que reduce significativamente la complejidad de SQL de las consultas comunes en el "Top-N".

FETCH FIRST permite principalmente simplificar la migración de bases de datos de terceros a Oracle Database. Sin embargo, también puede mejorar el rendimiento de algunas sentencias SELECT BULK COLLECT INTO. Para obtener más información, consulte "Row Limits for SELECT BULK COLLECT INTO Statements".

Grant Roles a paquetes PL/SQL Packages y programas Standalone

Antes de Oracle Database 12c, la unidad de derechos de un definidor (DR) siempre se ejecutaba con los privilegios del definidor y una unidad de derechos de invocación (IR) siempre se ejecutaba con los privilegios del invocador. Si quisiera crear una unidad PL / SQL que todos los usuarios pudieran invocar, incluso si sus privilegios fueran más bajos que los suyos, entonces tenía que ser una unidad DR. La unidad de DR siempre se ejecuta con todos sus privilegios, independientemente de qué usuario lo invocó.

A partir de Oracle Database 12c, puede otorgar roles a paquetes PL / SQL individuales y subprogramas independientes. En lugar de una unidad de DR, puede crear una unidad de IR y luego otorgarle roles. La unidad de IR se ejecuta con los privilegios tanto del invocador como de los roles, pero sin ningún privilegio adicional que tenga.

Para obtener más información, consulte Granting Roles to PL/SQL Packages and Standalone Subprograms".

Mismo tamaño máximo en SQL y PL/SQL en más tipos de datos

Antes de Oracle Database 12c, los tipos de datos VARCHAR2, NVARCHAR2 y RAW tenían diferentes tamaños máximos en SQL y PL / SQL. En SQL, el tamaño máximo de VARCHAR2 y NVARCHAR2 era de 4.000 bytes y el tamaño máximo de RAW era de 2.000 bytes. En PL / SQL, el tamaño máximo de cada uno de estos tipos de datos era 32.767 bytes.

A partir de Oracle Database 12c, el tamaño máximo de cada uno de estos tipos de datos es 32.767 bytes en SQL y PL / SQL. Sin embargo, SQL tiene estos tamaños máximos solo si el parámetro de inicialización MAX_STRING_SIZE está establecido en EXTENDED.

Para obtener información acerca de los tipos de datos extendidos, consulte la documentación Oracle Database SQL Language Reference.

Objetos LIBRARY

Antes de Oracle Database 12c:

• Se podía definir un objeto LIBRARY sólo mediante el uso de una ruta explícita, incluso en las versiones de Oracle Database donde el objeto DIRECTORIO estaba destinado a ser el único punto de mantenimiento para las rutas del sistema de archivos.

• Cuando se ejecutaba un subprograma almacenado en una biblioteca, el agente extproc siempre suplantaba al usuario propietario de la instalación de Oracle Database.

A partir de Oracle Database 12c:

• Se puede definir un objeto LIBRARY utilizando una ruta explícita o un objeto DIRECTORIO. El uso de un objeto DIRECTORIO mejora la seguridad y la portabilidad de una aplicación que utiliza procedimientos externos.

• Cuando se define un objeto LIBRARY, puede usar la cláusula CREDENTIAL para especificar el usuario del sistema operativo que el agente extproc usará al ejecutar un subprograma almacenado en la biblioteca. (El valor predeterminado es el propietario de la instalación de la base de datos Oracle).

Para más información acudir a la documentación, "CREATE LIBRARY Statement".

Resultados de declaraciones implícitas

Antes de Oracle Database 12c, un subprograma PL / SQL almacenado devolvía conjuntos de resultados desde consultas SQL explícitamente, a través de parámetros OUT REFCURSOR, y el programa cliente que invocaba el subprograma tenía que vincularse explícitamente a esos parámetros para recibir los conjuntos de resultados.

A partir de Oracle Database 12c, un subprograma almacenado PL / SQL puede devolver los resultados de la consulta a su cliente implícitamente, utilizando el paquete PLMS_SQL en lugar de los parámetros OUT REF CURSOR. Esta técnica facilita la migración de las aplicaciones que dependen del retorno implícito de los resultados de las consultas desde subprogramas almacenados desde bases de datos de terceros a Oracle Database.

Para más información ver "DBMS_SQL.RETURN_RESULT Procedure" y "DBMS_SQL.GET_NEXT_RESULT Procedure".

Vistas BEQUEATH CURRENT_USER

Antes de Oracle Database 12c, una vista siempre se comportaba como la unidad de derechos de un definidor (DR).

A partir de Oracle Database 12c, una vista puede ser BEQUEATH DEFINER (valor predeterminado), que se comporta como una unidad DR, o BEQUEATHCURRENT_USER, que se comporta como una unidad de derechos de invocación (IR)

Para obtener más detalles, consulte Oracle Database Security Guide. Para obtener información general sobre unidades DR e IR, consulte Invoker's Rights and Definer's Rights (AUTHID Property).

Privilegios INHERIT PRIVILEGES y Privilegios INHERIT ANY PRIVILEGES

Antes de Oracle Database 12c, una unidad de IR siempre se ejecutaba con los privilegios de su invocador. Si su invocador tenía privilegios más altos que su propietario, entonces la unidad de IR podría realizar operaciones involuntarias o prohibidas a su propietario.

A partir de Oracle Database 12c, una unidad IR puede ejecutarse con los privilegios de su invocador solo si su propietario tiene el privilegio INHERITPRIVILEGES en el invocador o el privilegio INHERIT ANY PRIVILEGES. Para obtener más información, consulte "Invoker's Rights and Definer's Rights (AUTHID Property)".

Columnas Invisibles

Una columna invisible es una columna oculta especificada por el usuario, que difiere de una columna oculta generada por el sistema de las siguientes maneras:

• Puede especificar explícitamente el nombre de una columna invisible siempre que pueda especificar explícitamente el nombre de una columna visible.

Para mostrar o asignar un valor a una columna invisible, debe especificar su nombre explícitamente, no implícitamente, como en el comando SQL * Plus DESCRIBE, SELECT *, Oracle Call Interface (OCI) describe y PL / SQL% ROWTYPE attribute.

• Puede hacer que una columna invisible sea visible.

Hacer visible una columna invisible cambia la estructura de algunos registros definidos con el atributo% ROWTYPE.

Objetos, No types, son Editioned o Noneditioned

Antes de Oracle Database 12c, un objeto de esquema era editable si su tipo era editable en la base de datos y su propietario tenía permiso para editar. Un usuario habilitado para editar no puede tener un objeto no editado de tipo editable.

A partir de Oracle Database 12c, un objeto de esquema es editable si su tipo es editable en el esquema que lo posee y tiene la propiedad EDITIONABLE. Un usuario habilitado para ediciones posee un objeto no editado de un tipo que es editable en la base de datos si el tipo no es editable en el esquema o si el objeto tiene la propiedad NONEDITIONABLE. Por lo tanto, las instrucciones "CREATE [O REPLACE]" y "ALTER Statements" le permiten especificar EDITIONABLE or NONEDITIONABLE.

Para ver más información al respecto, ir a la documentación Oracle Database Development Guide

Las Funciones PL/SQL de ejecución más rápida en SQL

A partir de Oracle Database 12c, dos tipos de funciones PL / SQL pueden ejecutarse más rápido en SQL:

• Funciones PL / SQL que se declaran y definen en las cláusulas WITH de las sentencias SELECT de SQL, descritas en Oracle Database SQL Language Reference

• Funciones PL / SQL que se definen con el "UDF Pragma"

Directivas de consulta predefinidas $$PLSQL_UNIT_OWNER y $$PLSQL_UNIT_TYPE

Antes de Oracle Database 12c, el código de diagnóstico podía identificar solo el nombre de la unidad PL / SQL actual (con la directiva de consulta predefinida $$ PLSQL_UNIT) y el número de la línea fuente en la que apareció la directiva de consulta predefinida $$ PLSQL_LINE en esa unidad.

A partir de Oracle Database 12c, las directivas de consulta predefinidas adicionales $$ PLSQL_UNIT_OWNER y $$ PLSQL_UNIT_TYPE permiten que el código de diagnóstico identifique el propietario y el tipo de la unidad PL / SQL actual. Para obtener más información, consulte "Predefined Inquiry Directives".

Obsolescencia del parámetro PLSQL_DEBUG (Deprecated)

El parámetro de compilación PLSQL_DEBUG, que especifica si compilar unidades PL / SQL para la depuración, está en desuso. Para compilar unidades PL / SQL para depuración, especifique PLSQL_OPTIMIZE_LEVEL = 1.

Para obtener información sobre los parámetros de compilación, consulte "PL/SQL Units and Compilation Parameters".

Características introducidas en 12CR2

Estas son las nuevas características para Oracle Database 12c Release 1 (12.1). La documentación oficial se encuentra publicada en el siguiente enlace:

https://docs.oracle.com/database/122/LNPLS/release-changes.htm#LNPLS105

Mejoras de la cláusula ACCESSIBLE BY

La cláusula ACCESSIBLE BY especifica una lista de unidades PL / SQL que se consideran seguras para invocar el subprograma y bloquea todas las demás.

A partir de Oracle Database 12c versión 2 (12.2), la lista de acceso se puede definir en subprogramas individuales en un paquete. Esta lista se verifica además de la lista de acceso definida en el paquete (si corresponde). Esta lista solo puede restringir el acceso al subprograma; no puede expandir el acceso.

Esta función de administración de código es útil para evitar el uso inadvertido de subprogramas internos. Por ejemplo, puede que no sea conveniente o factible reorganizar un paquete en dos paquetes: uno para un pequeño número de procedimientos que requieren acceso restringido y otro para las unidades restantes que requieren acceso público.

Intercalación de datos limitados (Data-Bound Collation)

La intercalación (también llamada clasificación) es un conjunto de reglas que determina si una cadena de caracteres es igual a, precede o sigue a otra cadena cuando las dos cadenas se comparan y ordenan.

Diferentes intercalaciones corresponden a reglas de diferentes idiomas hablados. Las operaciones sensibles a la intercalación son operaciones que comparan el texto y necesitan una intercalación para controlar las reglas de comparación. El operador de igualdad y la función incorporada INSTR son ejemplos de operaciones sensibles a la intercalación.

Oracle Database 12c versión 2 (12.2) agrega una nueva arquitectura para controlar la intercalación que se aplicará a las operaciones sobre datos de caracteres. En la nueva arquitectura, la intercalación se convierte en un atributo de datos de caracteres, análogo a un tipo de datos. Ahora puede declarar la intercalación para una columna y esta intercalación se aplica automáticamente por todas las operaciones SQL sensibles a la intercalación que hacen referencia a la columna. La función de intercalación de datos vinculados utiliza sintaxis y semántica compatibles con el estándar ISO / IEC SQL.

La versión 2 de Oracle Database 12c agrega una nueva propiedad llamada colación predeterminada a tablas, vistas, vistas materializadas, paquetes, procedimientos almacenados, funciones almacenadas, disparadores y tipos. La intercalación predeterminada de una unidad determina la intercalación de los contenedores de datos, como columnas, variables, parámetros, literales y valores de retorno, que no tienen su propia declaración de intercalación explícita en esa unidad. En esta versión, la clasificación predeterminada para paquetes, procedimientos

almacenados, funciones almacenadas, desencadenadores y tipos debe ser USING_NLS_COMP.

Para revisar la sintaxis de la cláusula acudir a DEFAULT COLLATION Clause

Controlar los privilegios de los derechos del definidor para procedimientos remotos

Si tus aplicaciones utilizan enlaces de base de datos y procedimientos de derechos de definidor, puede controlar cómo se otorgan los privilegios cuando los usuarios ejecutan el procedimiento de derechos del definidor.

Un nuevo privilegio INHERIT REMOTE PRIVILEGES permite a un usuario actual usar un enlace de base de datos de usuario conectado desde un procedimiento de derechos de definidor (DR). Sin este privilegio, el procedimiento DR no podrá conectarse utilizando el enlace de la base de datos de usuarios conectados.

Para obtener más información, consulte Connected User Database Links in DR Units

Mejoras de expresiones PL / SQL

A partir de Oracle Database 12c versión 2 (12.2), las expresiones se pueden usar en declaraciones donde anteriormente solo se permitían las constantes literales. Las expresiones estáticas ahora se pueden usar en declaraciones de subtipo.

La definición de expresiones estáticas se amplía para incluir todos los tipos escalares PL / SQL y una gama mucho más amplia de operadores. Los operandos de caracteres están restringidos a un subconjunto seguro del juego de caracteres ASCII. Los operadores cuyos resultados dependen de cualquier parámetro NLS implícito no están permitidos.

Las expresiones ampliadas y generalizadas tienen dos beneficios principales para los desarrolladores de PL / SQL:

- Los programas son mucho más adaptables a los cambios en su entorno.
- Los programas son más compactos, claros y sustancialmente más fáciles de entender y mantener.

Soporte para operadores SQL JSON en PL / SQL

Esta característica facilita trabajar con documentos JSON almacenados en una base de datos Oracle y generar documentos JSON a partir de datos relacionales.

El soporte de Oracle Database para almacenar y consultar documentos JSON en la base de datos se amplía mediante la adición de nuevas capacidades, incluida la capacidad de generar de forma declarativa documentos JSON a partir de datos relacionales mediante SQL y la capacidad de manipular documentos JSON como objetos PL / SQL. Los operadores SQL JSON son compatibles con PL / SQL con algunas excepciones. Ver SQL Functions in PL/SQL Expressions para conocer la lista de excepciones.

Soporte para identificadores más largos

La longitud máxima de todos los identificadores utilizados y definidos por PL / SQL se incrementa a 128 bytes, frente a los 30 bytes de versiones anteriores.

Si el parámetro COMPATIBLE se establece en un valor de 12.2.0 o superior, la representación del identificador en el conjunto de caracteres de la base de datos no puede exceder los 128 bytes. Si el parámetro COMPATIBLE se establece en un valor de 12.1.0 o inferior, el límite es de 30 bytes.

Se ha introducido una nueva función ORA_MAX_NAME_LEN_SUPPORTED para verificar este límite.

```
EXEC DBMS_OUTPUT.PUT_LINE (ORA_MAX_NAME_LEN_SUPPORTED);
128
```

Una nueva constante ORA_MAX_NAME_LEN define el máximo de longitud del nombre. Los nuevos subtipos DBMS_ID y DBMS_QUOTED_ID definen la longitud de identificadores en objetos para SQL, PL / SQL y usuarios.

Compartición de objetos comunes de aplicaciones vinculadas a metadatos

Un enlace de metadatos permite que los objetos de la base de datos en una base de datos enchufable de aplicación (PDB) compartan metadatos con objetos en la raíz de la aplicación.

Se introduce una nueva cláusula SHARING para especificar cómo se puede compartir una unidad PL / SQL almacenada entre un PDB y una raíz de aplicación. Los enlaces de metadatos son útiles para reducir los requisitos de memoria y disco porque almacenan solo una copia de los metadatos de un objeto (como el código fuente de un paquete PL / SQL) para objetos definidos de forma idéntica. Esto mejora el rendimiento de las operaciones de actualización porque los cambios en estos metadatos se realizarán en un lugar, la raíz de la aplicación. Ver Cláusula SHARING Clause para la sintaxis y la semántica.

Funcionalidades “Deprecated”

Las siguientes funciones están en desuso en esta versión y pueden ser desoportadas en una versión futura.

- El comando ALTER TYPE ... INVALIDATE está en desuso. Se debe usar la cláusula CASCADE en su lugar.
- La cláusula REPLACE de ALTER TYPE está en desuso. Se debe usar la cláusula alter_method_spec en su lugar. Alternativamente, puede volver a crear el tipo utilizando la instrucción CREATE O REPLACE TYPE.

Para la sintaxis y la semántica, vea ALTER TYPE Statement

Funcionalidades Desoportadas

El uso de SQLJ del lado del servidor pasa a estar fuera de soporte en la versión Oracle Database 12c versión 2 (12.2).

Oracle admite el uso de SQLJ del lado del cliente. Sin embargo, Oracle no admite el uso de SQLJ del lado del servidor, lo que incluye ejecutar procedimientos almacenados, tipos, funciones y disparadores en el entorno de la base de datos.