Unidad 6

Automatización de tareas

Administración de sistemas gestores de bases de datos

Índice

Introducción

La automatización de tareas consiste en conseguir que se éstas se realicen de forma sistemática sin intervención directa del usuario durante el proceso de ejecución. El administrador podrá, si así lo configuró, controlar y analizar la ejecución de dichas tareas y sus resultados. La automatización de tareas reduce tiempo y costes de administración y también disminuye el error humano que pueda ocurrir en una ejecución manual.

Para poder automatizar una tarea que, normalmente, se compone de una o varias subrutinas que forman un conjunto, se requiere elaborar un programa que incluya estas sentencias (opcionalmente algo de control) que será ejecutado de forma programada. Para ello, se puede hacer mediante un programa externo al gestor que sea lanzado con el programador de tareas del sistema operativo o sea lanzado desde el propio gestor (la mayoría incluyen programadores de tareas).

La ventaja del gestor es que no se necesita tener privilegios de administrador del sistema.

Vamos a definir los términos más importantes con los que trabajaremos durante la unidad.

- **Rutina externa:** Programa que se ejecuta desde el SGBD pero se invoca de forma externa con algún lenguaje de programación.
- **Rutina interna:** Programa que se ejecuta y se define en el SGBD.
- **Tarea de administración:** Rutinas que manipulan los objetos de la base de datos para que funcionen correctamente. No se modifican los datos de forma lógica.
- **Tareas de base de datos:** Rutinas que manipulan los datos de la base de datos según las necesidades operativas de la organización.
- **Procedimiento:** Rutina interna que se ejecuta sin devolver ningún resultado.
- **Función:** Rutina interna que devuelve un valor.
- **Script:** Conjunto de instrucciones que se ejecutan de forma secuencial.
- **Agente:** Es el servicio que ejecuta las tareas programadas en el SGBD.
- **Trigger:** Es una rutina interna que se ejecuta bajo un evento previamente definido con una serie de condiciones sobre los objetos.

Al finalizar esta unidad

- Comprobaremos de forma práctica las ventajas de automatizar ciertas tareas de administración del SGBD.
- Sabremos qué tareas son las más automatizables.
- Aprenderemos como se programan las tareas de administración en el SGBD.
- Crearemos programas sencillos con tareas de administración.
- Manejaremos con más habilidad el diccionario de datos para acceder a objetos que necesiten ser manipulados por una tarea automática de administración.

6.1. Rutinas en Bases de datos

Las rutinas son secuencias de comandos que permiten procesar ciertas acciones dentro de la base de datos. La rutina es una forma de reutilización de código, de forma que una vez se ha programado, se puede llamar tantas veces como sea necesaria.

Las hay de dos tipos, externas e internas.

Externas

En las rutinas externas, los programas se definen de forma externa al servidor. Dependiendo del sistema gestor, se comunican con éste, a través de alguna interfaz o por el sistema operativo. En el caso de Oracle, puede ejecutar rutinas externas mediante la máquina virtual de java.

Internas

Se definen sobre el propio SGBD y pueden ser de tres tipos: procedimientos almacenados, funciones o disparadores (triggers). Las rutinas internas aportan varias ventajas sobre las externas:

- **Rendimiento:** El optimizador actúa almacenando en caché los planes de ejecución óptimos.
- **Reutilización:** Se evita tener que escribir repetidas veces la misma consulta.
- **Encapsulación de reglas de negocio:** Si cambia el funcionamiento de una rutina interna, no es necesario tocar las aplicaciones.
- **Seguridad:** los usuarios que conectan a apps externas no necesitarán permisos para manipular las tablas desde aplicaciones externas. Por ejemplo, si una aplicación externa necesita generar usuarios de la base de datos, podría llamar a un procedimiento almacenado que lo haga.

Como principal desventaja, hay determinadas tareas que consumen recursos del servidor del SGBD como son la manipulación o cálculos complejos sobre los datos que podrían realizarse sobre rutinas externas, interactuando con el SGBD solamente para insertar o recuperar datos, pues está pensado más para tareas administrativas y de acceso a datos. Una arquitectura recomendable para evitar esto, es incorporar a la arquitectura una capa de servicios web que se encargue de gestionar la lógica de negocio, liberando al servidor de base de datos.

Las rutinas internas de un SGBD deben documentarse debidamente. Debe indicarse, al menos, la tarea de la rutina, parámetros, autor, versión y fecha de la última modificación. Si la rutina es compleja, es recomendable comentar sobre el código adicionalmente.

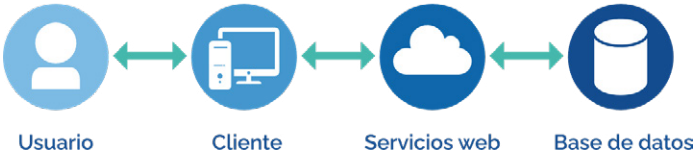

Imagen 1. Arquitectura en 3 capas

6.2. Rutinas internas en Oracle

El sistema gestor de base de datos de Oracle funciona con el lenguaje de programación PL/SQL y se usa para crear procedimientos, funciones y disparadores.

6.2.1. Permisos

Es necesario tener habilitados los permisos "CREATE PROCEDURE" y "CREATE TRIGGER" para que el usuario pueda crear procedimientos o disparadores respectivamente. Si el usuario tiene permisos para crear sus propios procedimientos, también podrá ejecutarlos.

Para dar permisos de ejecución sobre procedimientos, funciones o disparadores se utilizará "GRANT EXECUTE":

```
GRANT CREATE PROCEDURE, CREATE TRIGER TO
<usuario_nombre>;

GRANT EXECUTE ON <esquema_nombre>.<rutina> TO
<usuario_nombre> [WITH GRANT OPTION];
```

6.2.2. Procedimientos

Los procedimientos se ejecutan en bloque (como si fueran una instrucción) y no devuelven ningún valor.

Por defecto, los permisos del procedimiento son los que tiene el usuario que lo llamó (INVOKER), por lo que, si se necesita acceder a un objeto del que el usuario no tiene los permisos, el procedimiento dará un error. Aunque existe la opción de modificar este modo y ejecutar el procedimiento con los mismos permisos que el creador del procedimiento (DEFINER).

La sintaxis de un procedimiento es la siguiente:

```
CREATE OR REPLACE PROCEDURE <procedimiento_nombre> (
       <parámetro_1> [IN | OUT| IN OUT |] <tipo>,
       <parámetro_2> [IN | OUT| IN OUT |] <tipo>,
       …
)
[AUTH ID INVOKER | DEFINER]
IS | AS
BEGIN
-- Código PL/SQL
END;
```

Los parámetros pueden ser de entrada (por defecto: IN), de salida (OUT), o de entrada/salida (IN OUT). En los primeros, su valor no se modifica, los de salida se utilizan para generar un valor de salida, por ejemplo, un código de error, en los últimos, el parámetro recibido se utiliza en el procedimiento y se modifica el valor.

Para ejecutar los procedimientos almacenados, la sintaxis es:

```
EXECUTE <procedimiento_nombre>(<parámetro_1>,
<parámetro_2>, …>);
```

```
CREATE OR REPLACE PROCEDURE cambiar_departamento (
p_empleado IN NUMBER,
p_nuevo_dpto IN NUMBER
) IS
  BEGIN
    UPDATE EMPLEADOS
    SET DEPARTMENT_ID = p_nuevo_dpto
    WHERE EMPLOYEE_ID = p_empleado;
    COMMIT;
  END;

EXECUTE CAMBIAR_DEPARTAMENTO(100, 50);
```

Imagen 1. Ejemplo de procedimiento que cambia a un empleado de departamento

6.2.3. Funciones

Los parámetros de una función siempre son de entrada y devuelven un único valor, que puede ser de cualquier tipo compatible con Oracle.

La sintaxis para crear una función es la siguiente:

```
CREATE OR REPLACE FUNCTION <función_nombre> (
       <parámetro_1> <tipo>,
       <parámetro_2> <tipo>,
       …
)
RETURN <tipo>
[AUTH ID INVOKER | DEFINER]
IS | AS
<variables>d
BEGIN
-- Código PL/SQL
RETURN <valor_devuelto>
END;
```

```
CREATE OR REPLACE FUNCTION obtener_departamento(
  p_empleado NUMBER
  )
  RETURN NUMBER
  IS
  dpto NUMBER;
  BEGIN
    SELECT DEPARTMENT_ID INTO dpto FROM EMPLOYEES
    WHERE EMPLOYEE_ID = p_empleado;
    RETURN dpto;
  END;

SELECT OBTENER_DEPARTAMENTO(100) FROM DUAL;
```

Imagen 2. Ejemplo de función que devuelve el código de departamento de un empleado

```
CREATE OR REPLACE FUNCTION cuantos_empl_cobran_mas_que(
  p_empleado NUMBER
  )
  RETURN INT
  IS
  salario NUMBER;
  empleados_sup INT;
  BEGIN
    SELECT SALARY INTO salario FROM EMPLOYEES
    WHERE EMPLOYEE_ID = p_empleado;
    SELECT COUNT(*) INTO empleados_sup FROM EMPLOYEES e
    WHERE E.SALARY > salario;
    RETURN empleados_sup;
  END;
  SELECT cuantos_empl_cobran_mas_que(200) FROM DUAL;
```

1000

CUANTOS_EMPL_COBRAN_MAS_QUE(200)
NUMBER
60

Imagen 3. Ejemplo de función que devuelve el nº de empleados que cobran más que otro, que se le pasa como parámetro.

¿SABÍAS QUÉ?

En Oracle, se utiliza una tabla del sistema llamada dual, que sirve para ejecutar consultas, por ejemplo, con llamadas a funciones que no requieren uso de tablas. Mientras que en otros SGBD sería correcto "SELECT OBTENER_DEPARTAMENTO(100);", en Oracle hay que terminar la sentencia con "FROM DUAL".

6.2.4. Disparadores (Triggers)

Son rutinas que se autoejecutan cuando se produce un determinado evento sobre su tabla asociada. El evento puede ser inserción, actualización o eliminación de datos. El flujo de creación y la sintaxis es bastante extenso y se puede consultar en la documentación oficial de Oracle.

Veamos algún ejemplo:

```
CREATE OR REPLACE TRIGGER disparador
  BEFORE
    INSERT OR
    UPDATE OF salary, department_id OR
    DELETE
  ON employees
BEGIN
  CASE
    WHEN INSERTING THEN
      DBMS_OUTPUT.PUT_LINE('Insertando');
    WHEN UPDATING('salary') THEN
      DBMS_OUTPUT.PUT_LINE('Actualizando salario');
    WHEN UPDATING('department_id') THEN
      DBMS_OUTPUT.PUT_LINE('Actualizando departamento');
    WHEN DELETING THEN
      DBMS_OUTPUT.PUT_LINE('Eliminando empleado/s');
  END CASE;
END;
```

Crea un disparador que se activa al insertar filas, al actualizar alguna de las columnas "salary" ó "department_id" o eliminar filas sobre la tabla employees. Dependiendo del evento que lo dispare, mostrará por consola con DBMS_OUTPUT.PUT_LINE un texto diferente. La variable serveroutput tiene que estar activada para que funcione.

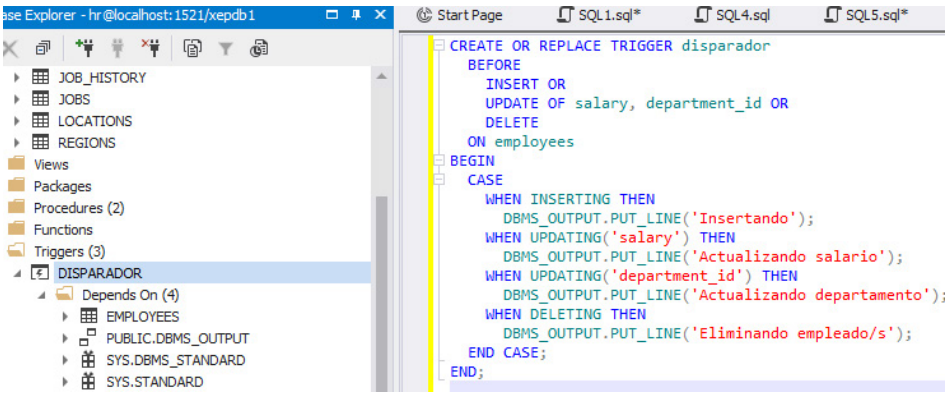

Imagen 4. Creando el disparador

A lo largo de esta unidad, las rutinas se centrarán en tareas orientadas a administrar el sistema gestor.

6.2.5. Estructuras de programación

Las estructuras del lenguaje son comunes para funciones, procedimientos y disparadores. Algunas de ellas también son comunes a otros tipos de lenguajes de programación.

Comentarios

Existen de dos tipos:

- **De una línea:** con el operador "--".
- **De varias líneas:** se abren con "**/***" y se cierran con **"*/".** Ej. /*comentario de más de una línea */

Variables

Se utilizan para almacenar valores dentro de un procedimiento de forma temporal. Se pueden usar en cualquier lugar dentro del procedimiento, función o disparador. Se tienen que declarar en la zona de código que la sintaxis PL/SQL define para ello. Para asignar un valor, se puede hacer de dos modos:

- `nombre_variable := valor:` operador asignación
- `select (…) into nombre_variable:` Cuando se asigna mediante una consulta

Cuando se declara una variable, es frecuente asignarle un valor inicial con el operador asignación.

Hay que tener en cuenta los tipos de las variables, que se deben especificar en su declaración. Las variables pueden ser del tipo de la columna, o de tipo fila (ROWTYPE), y sus campos tendrán los mismos tipos que los de la fila.

Expresiones condicionales

- `IF/ELSE/ELSIF:` Se ejecuta la sentencia o el bloque dentro del IF, si se cumple una condición.

```
IF condition THEN
  statements
END IF;
```

- `CASE/WHEN:` Se evalúa una variable o expresión. Se puede ejecutar en dos modos. Evaluando la variable o expresión y en función de su valor ejecutar una instrucción, en ese caso en el momento en el que se cumple una, no se evalúan las siguientes (case simple). También se puede utilizar con una condición por cada when, en este caso se evalúan todas.

```
CASE selector
WHEN selector_value_1 THEN statements_1
WHEN selector_value_2 THEN statements_2
...
WHEN selector_value_n THEN statements_n
[ ELSE
  else_statements ]
```

```
END CASE;]
CASE
WHEN condition_1 THEN statements_1
WHEN condition_2 THEN statements_2
...
WHEN condition_n THEN statements_n
[ ELSE
  else_statements ]
END CASE;]
```

Bucles

> LOOP: Bucle básico

```
DECLARE
  i PLS_INTEGER := 0;
  j PLS_INTEGER := 0;
BEGIN
  LOOP
    i := i + 1;
    DBMS_OUTPUT.PUT_LINE ('i = ' || i);
    LOOP
      j := j + 1;
      DBMS_OUTPUT.PUT_LINE ('j = ' || j);
      EXIT WHEN (j > 3);
    END LOOP;

    DBMS_OUTPUT.PUT_LINE ('Saliendo del bucle interno');

    EXIT WHEN (i > 2);
  END LOOP;
  DBMS_OUTPUT.PUT_LINE ('Saliendo del bucle externo');
END;
```

Text

Output

General | SQL Log | DBMS Output | Debug

```
i = 3
j = 6
Exited inner loop
Exited outer loop

i = 1
j = 1
j = 2
j = 3
j = 4
Saliendo del bucle interno
i = 2
j = 5
Saliendo del bucle interno
i = 3
j = 6
Saliendo del bucle interno
Saliendo del bucle externo
```

Imagen 5. Ejemplo de bucle básico

> FOR LOOP:

```
DECLARE
  i PLS_INTEGER :=0;
BEGIN
  FOR i IN 1..3 LOOP
    DBMS_OUTPUT.PUT_LINE ('Dentro del bucle, i vale ' || TO_CHAR(i));
  END LOOP;

  DBMS_OUTPUT.PUT_LINE ('Fuera del bucle, i vale ' || TO_CHAR(i));
END;
```

```
Dentro del bucle, i vale 1
Dentro del bucle, i vale 2
Dentro del bucle, i vale 3
Fuera del bucle, i vale 0
```

Imagen 6. Ejemplo de bucle FOR

> WHILE LOOP:

```
DECLARE
  done  BOOLEAN := FALSE;
BEGIN
  WHILE done LOOP
    DBMS_OUTPUT.PUT_LINE ('Esto no se imprime');
    done := TRUE;  -- This assignment is not made.
  END LOOP;

  WHILE NOT done LOOP
    DBMS_OUTPUT.PUT_LINE ('Hello, world!');
    done := TRUE;
  END LOOP;
END;
```

Imagen 7. Ejemplo de bucle WHILE. Cuando se cumple la condición, no realiza la primera vuelta.

Bloques de instrucciones (BEGIN-END)

Se usan para ejecutar bloques secuenciales como si fuese una sola instrucción. Resultan de utilidad en estructuras de control (bucles o expresiones condicionales).

Cursores

Los cursores son variables que almacenan los valores devueltos por una consulta y reutilizarlos en otra operación. Igual que las variables, se declaran en la zona de definición de variables. La sintaxis para su declaración es:

```
CURSOR <cursor_nombre>(<param1> <param1_tipo>,
<param2> <param2_tipo>, …) IS SELECT …;
```

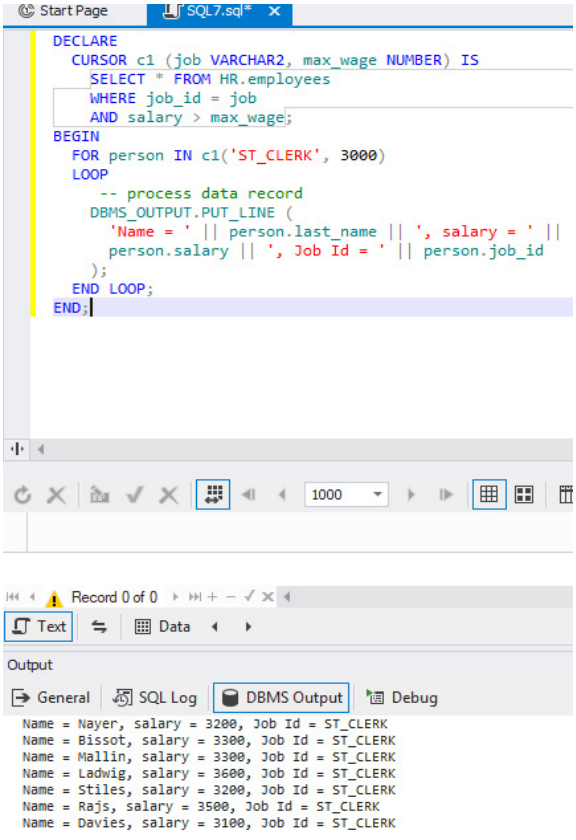

Imagen 8. Ejemplo de bucle y cursor parametrizado.

Transacciones

Oracle trata cualquier sentencia que no sea una consulta, es decir, que modifique los datos, como el inicio de una transacción. Las rutinas por defecto no realizan un commit de cada transacción, por lo que es importante tanto incluirlo como hacerlo de forma apropiada.

PARA TENER EN CUENTA...

En un proceso de movimiento de datos masivos, se puede producir un error por saturación del tablespace UNDO debido a la gran cantidad de datos que contiene la transacción sin confirmar. Es importante tener en cuenta y gestionarlo de modo que se realicen commit cada ciertas operaciones.

Manejo de excepciones

Como la mayoría de lenguajes de programación, en PL/SQL se incluye el tratamiento de las excepciones, de forma que la rutina pueda tener una alternativa de escape cuando se produzca el error, sin tener que parar o cerrar el programa bruscamente. Además de los mensajes de error, es habitual guardar logs y si se da el caso, suele ser el lugar para hacer rollback.

```
CREATE OR REPLACE PROCEDURE obtener_estado_tarea (
  fecha_max DATE,
  nombre_tarea VARCHAR2
) IS
 hoy DATE;
  vencida  EXCEPTION;  -- declare exception
BEGIN
hoy := CURRENT_DATE;
  IF fecha_max < hoy THEN
    RAISE vencida;  -- explicitly raise exception
  ELSE DBMS_OUTPUT.PUT_LINE('estás a tiempo de realizar la tarea ' ||
  COMMIT;
  END IF;
EXCEPTION
  WHEN vencida THEN  -- handle exception
    DBMS_OUTPUT.PUT_LINE (NOMBRE_TAREA || ' ha vencido');
    ROLLBACK;
END;

BEGIN
  obtener_estado_tarea (TO_DATE('12-AGO-2022'), 'Unidad 6');
  obtener_estado_tarea (TO_DATE('10-AGO-2022'), 'Unidad 5');
END;
```

Imagen 9. Ejemplo de sintaxis de excepciones sobre un procedimiento

```
BEGIN
  obtener_estado_tarea (TO_DATE('12-AGO-2022'), 'Unidad 6');
  obtener_estado_tarea (TO_DATE('10-AGO-2022'), 'Unidad 5');
END;
```

```
Output
General | SQL Log | DBMS Output | Debug
  estás a tiempo de realizar la tarea Unidad 6
  Unidad 5 ha vencido
```

Imagen 10. Ejecutando los procedimientos, el segundo lanzaría la excepción.

6.3. Automatización de tareas

La automatización de tareas tiene como objetivos reducir carga de trabajo y evitar errores, especialmente de aquellas tareas que son repetitivas o cíclicas.

Las tareas que podemos automatizar en nuestro SGBD se pueden clasificar en dos categorías: de base de datos o de administración.

Tareas de base de datos

Son tareas que tienen que ver con las reglas de negocio de la organización, modificando o creando datos sobre los objetos. Normalmente las crea el diseñador de base de datos, en coordinación con el equipo desarrollador de software. Algunos ejemplos podrían ser un procedimiento que gestione el alta de usuarios de una aplicación, asociada a usuarios del sistema gestor o un disparador que guarde un histórico de datos eliminados mediante un disparador que se active con el evento "delete".

Tareas de administración

Son tareas que define el DBA y están orientadas al mantenimiento del SGBD. No suelen programarse por disparadores, sino a través de un programador de tareas en el que se especifica la tarea y cuándo debe ejecutarse. El agente o programador de tareas puede estar integrado en el SGBD o ser externo (gestor de tareas del sistema operativo).

6.3.1. Tareas de administración automatizables

Con un sistema gestor en entorno de producción, aparecerán problemas que pueden ser resueltos mediante programación de rutinas. La aparición de problemas más graves (falta de espacio, consultas que se vuelven lentas, etc.), incluso se puede prevenir.

En ocasiones, es el propio sistema gestor el que dispone de mecanismos para ir regulando y actuando de forma preventiva sobre determinados objetos, lanzando rutinas o avisos mediante el registro de errores o monitor de rendimiento. Sin embargo, es difícil mantener un control sobre este tipo de procesos en segundo plano, que, por otro lado, podrían sobrecargar el sistema si no se ejecutan en momentos de baja carga. Por ese motivo, en ocasiones es conveniente deshabilitarlos y crearlos manualmente.

La estructura habitual de un procedimiento administrativo suele ser crear un cursos sobre una consulta que busca en el diccionario de datos una lista de objetos con posibles problemas y luego un bucle irá recorriendo el cursos y ejecutando las tareas que resuelvan el problema.

PARA TENER EN CUENTA...

La información que almacena Oracle en el diccionario de datos está en mayúsculas. Para consultar cualquier información sobre un objeto, es conveniente utilizar la función upper:

```
CREATE TABLE quesos(
variedad VARCHAR2(30),
origen VARCHAR2(30),
calidad VARCHAR2(30)
);

SELECT *
FROM user_tab_columns
WHERE table_name = upper('quesos');
```

Imagen 11. Si no indicamos upper, aunque la tabla exista, no la va a encontrar porque en la tabla del diccionario de datos, tanto los nombres de las tablas como las columnas se guardan en mayúsculas.

Algunas de las principales tareas a automatizar por el DBA son: Copias de seguridad, particionamiento, actualización de estadísticas, desfragmentación, balanceo de índices, purgado, etc.

Otras tareas se pueden automatizar desde el sistema operativo, como lectura de logs, comprobación de espacio en disco, etc.

6.3.2. Copias de seguridad

Aunque hay sistemas gestores que permiten hacerlo con su propio agente, normalmente se suelen realizar la automatización por aplicaciones externas (aunque se instalen con el sistema gestor) que son invocadas por el sistema operativo.

6.3.3. Particionamiento

Consiste en dividir una tabla de gran tamaño en tablas particionadas o subtablas que se tratan como si fuesen tablas separadas, lo cual aporta las siguientes ventajas:

- **Acceso optimizado a tabla:** Cuando una tabla ha crecido mucho verticalmente, como norma general, se suelen consultar los datos más recientes (por ejemplo, ventas). Si la tabla ocupa millones de registros, de meses o años anteriores, el acceso será más lento.
- **Optimización de tareas de administración:** Las operaciones de actualización sobre ciertos datos consolidados ya no serán necesarias, por lo que facilita su archivado (compresión, mover a discos más lentos, etc.).
- **Facilidad para la purga de datos:** Las operaciones de purga se ejecutan más rápido si se hacen a nivel de partición, pudiendo mover la partición en lugar de la tabla completa.
- **Reducción de uso de tablespaces:** Las operaciones de modificación de datos masivas, ocupan un gran espacio temporal hasta que se complete la transacción, por lo que, si una tabla está particionada, no lo colapsará.
- **Intercambio de particiones:** Este tipo de operaciones permiten mover rápidamente la información de un sistema gestor a otro de un solo bloque y con un tamaño "tratable".

El particionamiento de una tabla se realiza por un campo de la tabla, a la que se le puede indicar el rango de valores que pertenecen a dicha partición. El campo debe ser de tipo fecha o numérico no puede ser nulo.

Las particiones pueden ser de dos tipos: fijas o por fechas.

- **Fijas:** se especifican al crear la tabla y no se producen cambios posteriores. Por ejemplo, si una tienda online vende productos en varios países, la tabla ventas podría particionarse por el id del país.
- **Variables:** Se suelen dar cuando el campo por el que se particiona es de tipo fecha. A medida que va pasando el tiempo y se agregan registros a la tabla, se irán creando nuevas particiones.

Las operaciones relativas al particionamiento que se deben automatizar son:

- **Creación:** las tablas con particiones variables tienen definidos unos rangos que inicialmente contienen todos los datos. Cuando los registros nuevos que se crean no pertenecen a ninguno de los rangos especificados, se mueven a una partición final (P_FINAL). Cuando se hace necesario crear nuevas particiones, los registros que estaban en P_FINAL y que ahora encajan con la nueva partición creada se mueven a dicha partición.
- **Compresión:** en tablas que ocupan gran espacio, puede ser conveniente reducir el tamaño que ocupan las particiones antiguas a las que prácticamente no se accede. Esta operación, mejora el rendimiento de las consultas sobre la tabla, pero penaliza las operaciones de modificación de los datos comprimidos, por lo que es conveniente comprimir solamente datos consolidados. También hay que tener en cuenta que comprimir una tabla o partición anula la funcionalidad de los índices. Una automatización habitual es comprimir la última partición cada vez que se crea una nueva.
- **Reconstrucción de índices:** Cuando se comprime una partición, se deben reconstruir los índices que quedaron inutilizables.
- **Mover particiones:** para optimizar los recursos de hardware, puede ser conveniente mover las particiones comprimidas a dispositivos de almacenamiento más lento (y barato), liberando espacio de almacenamiento rápido (ssd) para aquellos datos con más volatilidad.

6.3.4. Estadísticas

Conviene mantener actualizadas las estadísticas de aquellas tablas que tienen bastante movimiento de datos. De este modo, los planes de ejecución se invalidan y se crean nuevos planes.

Los sistemas gestores suelen hacer la actualización de forma autónoma, sin embargo, durante la operación se produce un bloqueo de las tablas, lo cual puede ser un problema en tablas grandes. Por ello se recomienda tener controlados este tipo de procesos y realizar la actualización de forma controlada a través de programación de rutinas que se ejecuten en horarios de baja carga.

6.3.5. Desfragmentación

Las operaciones que se van realizando en entornos operativos de producción, sobre todo modificación y borrado van dejando espacios vacíos en los bloques. Esto, además de ocupar espacio innecesariamente, provoca que cuando se consulta una tabla, la consulta cuando accede al disco, lea todos los registros incluso los vacíos, con la consecuente penalización en rendimiento.

Para solucionar este problema, se puede utilizar un procedimiento que busque en el diccionario de datos las tablas con más espacios vacíos y los compacten.

Los dos problemas principales son:

- **Fragmentación de tablas:** Provocada por el borrado de registros de tablas.
- **Segmentación de filas en varios bloques:** Ocurre al modificar filas de una tabla que requieren una ampliación de tipos. Esto provoca una segmentación horizontal de la fila, guardando cada segmento en una zona no contigua del disco, dejando huecos en la ubicación inicial del disco y necesitando dos operaciones de acceso a disco.

Con siguiente sentencia SQL, se solucionan ambos tipos de problema:

```
ALTER TABLE <tabla_nombre> MOVE;
```

6.3.6. Reconstrucción de índices

Cuando se anulan los índices, por ejemplo, al comprimir una tabla o cambiar su tablespace, este estado puede provocar errores en la base de datos, por ejemplo, al lanzar las estadísticas o al insertar o modificar datos. Este tipo de errores son difíciles de controlar, por lo que se debe prestar especial atención a su mantenimiento.

Los procedimientos habituales para gestionar estas tareas consisten en declarar un cursor que busque en el diccionario de datos aquellos índices que están en estado inválido y se recorra lanzando las sentencias para reconstruirlo.

```
ALTER INDEX <índice_nombre> REBUILD;
```

6.3.7. Purgado y paso a histórico

Cuando una tabla ha alcanzado un gran tamaño vertical, las operaciones se degrada el rendimiento de las operaciones que se realizan. Esto suele ocurrir en tablas críticas, como compras, ventas, facturas, etc. que se suelen almacenar en discos RAID de alta seguridad, que normalmente son bastante caros y es mejor reservarlos para información con la que se trabaje con más frecuencia.

Para aquella información que se va quedando obsoleta y apenas se va a acceder, se suele almacenar en tablas de histórico comprimidas y en discos más lentos y económicos e incluso la información muy antigua e innecesaria, a la que rara vez vaya a acceder, se puede llegar a archivar en discos externos fuera del SGBD.

Las operaciones de purgado y paso a histórico, pueden ser programadas para que se ejecuten regularmente, por ejemplo, a principios de año, volcando la información del año anterior al histórico y/o purgando la información más antigua.

6.4. Automatización de tareas en Oracle

Oracle incorpora su propio agente para gestionar la planificación de tareas automáticas, aunque también se pueden programar utilizando las herramientas del sistema operativo.

6.4.1. Herramientas del SO

Las tareas programadas se pueden ejecutar desde scripts invocando al programa SQLPLUS. El inconveniente de esta operación es que se necesita indicar las credenciales de usuario, lo cual puede suponer un problema de seguridad.

```
Sqlplus usuario/contraseña@servicio @script.sql > log.txt
```

Con esta línea, por ejemplo, en un cron (Linux), podemos programar con qué frecuencia o a qué hora se ejecuta la tarea.

6.4.2. DBMS Scheduler

Es el nuevo planificador de tareas de Oracle (a partir de la versión 10g). Se puede invocar mediante el paquete DMBS_SCHEDULER (en versiones anteriores a 10g DBMS_JOB). Este paquete permite realizar la programación de tareas (Jobs) o grupos de tareas (chains). Los Jobs pueden ser desde sentencias simples SQL, bloques de código PL/SQL o rutinas, o incluso la ejecución de un programa externo a la base de datos. Los Jobs o chains se crean con los procedimientos DBMS_SCHEDULER.CREATE_JOB y DBMS_SCHEDULER.CREATE_CHAIN respectivamente. Se recomienda revisar la documentación del paquete, especialmente sobre los dos procedimientos mencionados:

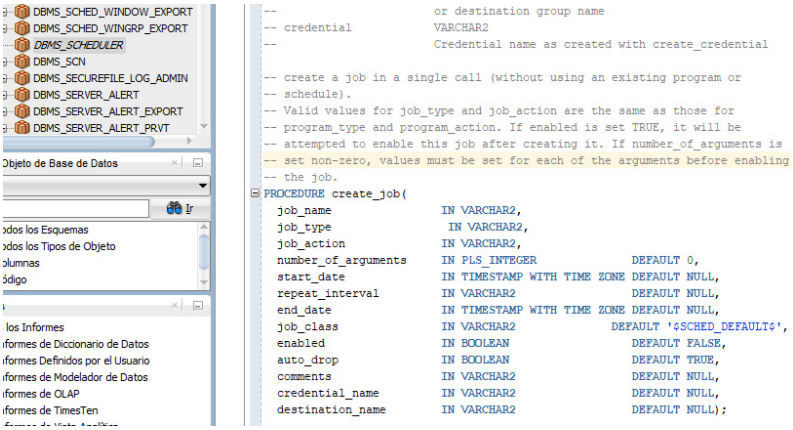

Imagen 12. DBMS_SCHEDULER (CREATE_JOB)

Si el job que queremos crear ejecuta un procedimiento que no tiene parámetros, su programación es más sencilla. Si el procedimiento requiere parámetros, se debe crear en tres pasos:

1. Ejecutar DBMS_SCHEDULER.CREATE_JOB, especificando el número de argumentos y asignándole el estado "disabled".
2. Por cada parámetro del procedimiento, hay que ejecutar DBMS_SCHEDULER.SET_JOB_ARGUMENT_VALUE, indicando el job, su posición y su valor.
3. Finalmente, modificar la programación de la tarea, cambiando el estado del job a "enable", con DBMS_SCHEDULER.ENABLE

```
BEGIN
  DBMS_SCHEDULER.CREATE_JOB(
  job_name => 'Purga_ventas_mes',
  job_type => 'STORED_PROCEDURE',
  job_action => 'pr_purga_datos', /* Recibe dos parámetros: modo (1: normal, 2: rapido)
                                  y particionado (TRUE: con particionado, FALSE: sin) */
  number_of_arguments => 2,
  start_date => '01-JAN-2022 00:00:00',
  repeat_interval => 'FREQ=MONTHLY; BYDAY=LASTDAY',
  end_date => '31-DEC-2022 23:59:59',
  enabled => FALSE,
  comments => 'Este trabajo purga los datos históricos de ventas cada mes.');
  DBMS_SCHEDULER.SET_JOB_ARGUMENT_VALUE('Purga_ventas_mes', 1, 1);
  DBMS_SCHEDULER.SET_JOB_ARGUMENT_VALUE('Purga_ventas_mes', 2, FALLSE);
  DBMS_SCHEDULER.ENAGLE('Purga_ventas_mes');
END;
```

Imagen 13. Ejemplo de rutina programada. Se ejecuta cada mes y llama al procedimiento almacenado "pr_purga_datos', pasándole como parámetros 1 (purga normal) y FALSE (no particionamiento)

La ejecución de esta tarea la lleva a cabo el agente de Oracle, que es un servicio que se encarga de la ejecución de tareas programadas.

PARA TENER EN CUENTA...

Para que un usuario pueda programar tareas, es necesario habilitar los permisos de creación de Jobs y permisos de ejecución sobre el paquete DBMS_SCHEDULER:

```
GRANT CREATE JOB TO <usuario_nombre>;
GRANT EXCECUTE ON DBMS_SCHEDULER TO <usuario_nombre>;
```

Si hay varios Jobs que requieren el mismo periodo de ejecución y siempre se ejecutan secuencialmente, es conveniente agruparlos en una cadena. Al conjunto de Jobs, cadenas que ejecutan tareas de mantenimiento periódicas se le conoce como plan de mantenimiento.

La forma más fácil de crear las cadenas es utilizando la herramienta SQL Developer, que permite programar tanto Jobs como cadenas.

www.universae.com