EBook Gratis

APRENDIZAJE Oracle Database

Free unaffiliated eBook created from
Stack Overflow contributors.

#oracle

Tabla de contenido

Acerca de

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: oracle-database

It is an unofficial and free Oracle Database ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official Oracle Database.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Capítulo 1: Comenzando con la base de datos Oracle

Observaciones

Oracle es un sistema de gestión de bases de datos relacionales (RDBMS) originalmente construido por Larry Ellison, Bob Miner y Ed Oates a finales de los 70. Fue pensado para ser compatible con el Sistema R de IBM.

Versiones

Versión	Fecha de lanzamiento
Versión 1 (inédito)	1978-01-01
Oracle V2	1979-01-01
Oracle Version 3	1983-01-01
Oracle Version 4	1984-01-01
Oracle Version 5	1985-01-01
Oracle Version 6	1988-01-01
Oracle7	1992-01-01
Oracle8	1997-07-01
Oracle8i	1999-02-01
Oracle9i	2001-06-01
Oracle 10g	2003-01-01
Oracle 11g	2007-01-01
Oracle 12c	2013-01-01

Examples

Hola Mundo

```
SELECT 'Hello world!' FROM dual;
```

En la versión de Oracle de SQL, "dual es solo una tabla de conveniencia" . Originalmente estaba pensado para duplicar filas a través de UNIR, pero ahora contiene una fila con un valor `DUMMY` de 'X'.

¡Hola Mundo! de la mesa

Crear una tabla simple

```
create table MY_table (
   what varchar2(10),
   who varchar2(10),
   mark varchar2(10)
);
```

Insertar valores (puede omitir columnas de destino si proporciona valores para todas las columnas)

```
insert into my_table (what, who, mark) values ('Hello', 'world', '!' );
insert into my_table values ('Bye bye', 'ponies', '?' );
insert into my_table (what) values('Hey');
```

Recuerda comprometerte, porque Oracle utiliza *transacciones.*

```
commit;
```

Seleccione sus datos:

```
select what, who, mark from my_table where what='Hello';
```

Consulta SQL

Enumere los empleados que ganan más de $ 50000 nacidos en este siglo. Escriba su nombre, fecha de nacimiento y salario, ordenados alfabéticamente por nombre.

```
SELECT employee_name, date_of_birth, salary
FROM   employees
WHERE  salary > 50000
   AND date_of_birth >= DATE '2000-01-01'
ORDER BY employee_name;
```

Muestra el número de empleados en cada departamento con al menos 5 empleados. Lista de los departamentos más grandes primero.

```
SELECT department_id, COUNT(*)
FROM   employees
GROUP BY department_id
HAVING COUNT(*) >= 5
ORDER BY COUNT(*) DESC;
```

Hola Mundo desde PL / SQL

```
/* PL/SQL is a core Oracle Database technology, allowing you to build clean, secure,
   optimized APIs to SQL and business logic. */

set serveroutput on

BEGIN
   DBMS_OUTPUT.PUT_LINE ('Hello World!');
END;
```

Lea Comenzando con la base de datos Oracle en línea:
https://riptutorial.com/es/oracle/topic/558/comenzando-con-la-base-de-datos-oracle

Capítulo 2: Actualizar con uniones

Introducción

Contrariamente a los malentendidos generalizados (incluso en SO), Oracle permite actualizaciones a través de uniones. Sin embargo, hay algunos requisitos (bastante lógicos). Ilustramos lo que no funciona y lo que hace a través de un ejemplo simple. Otra forma de lograr lo mismo es la declaración MERGE.

Examples

Ejemplos: lo que funciona y lo que no.

```
create table tgt ( id, val ) as
  select 1, 'a' from dual union all
  select 2, 'b' from dual
;

Table TGT created.

create table src ( id, val ) as
  select 1, 'x' from dual union all
  select 2, 'y' from dual
;

Table SRC created.

update
  ( select t.val as t_val, s.val as s_val
    from   tgt t inner join src s on t.id = s.id
  )
set t_val = s_val
;

SQL Error: ORA-01779: cannot modify a column which maps to a non key-preserved table
01779. 00000 -  "cannot modify a column which maps to a non key-preserved table"
*Cause:    An attempt was made to insert or update columns of a join view which
           map to a non-key-preserved table.
*Action:   Modify the underlying base tables directly.
```

Imagine lo que sucedería si tuviéramos el valor `1` en la columna `src.id` más de una vez, con valores diferentes para `src.val` . Obviamente, la actualización no tendría sentido (en CUALQUIER base de datos, eso es un problema lógico). Ahora, **sabemos** que no hay duplicados en `src.id` , pero el motor de Oracle no sabe que - por lo que se queja. Quizás esta es la razón por la que tantos profesionales creen que Oracle "no tiene ACTUALIZACIÓN con uniones".

Lo que Oracle espera es que `src.id` sea único y que, Oracle, lo sepa de antemano. ¡Fácilmente arreglado! Tenga en cuenta que lo mismo funciona con claves compuestas (en más de una columna), si la coincidencia para la actualización necesita usar más de una columna. En la práctica, `src.id` puede ser PK y `tgt.id` puede ser FK apuntando a esta PK, pero eso no es

relevante para las actualizaciones con join; lo que *es* relevante es la restricción única.

```
alter table src add constraint src_uc unique (id);

Table SRC altered.

update
  ( select t.val as t_val, s.val as s_val
    from   tgt t inner join src s on t.id = s.id
  )
set t_val = s_val
;

2 rows updated.

select * from tgt;

ID  VAL
--  ---
 1  x
 2  y
```

El mismo resultado se podría lograr con una declaración MERGE (que merece su propio artículo de Documentación), y personalmente prefiero MERGE en estos casos, pero la razón no es que "Oracle no hace actualizaciones con las combinaciones". Como muestra este ejemplo, Oracle *hace* actualizaciones con uniones.

Lea Actualizar con uniones en línea: https://riptutorial.com/es/oracle/topic/8061/actualizar-con-uniones

Capítulo 3: Bloque PL / SQL anónimo

Observaciones

Dado que no tienen nombre, los bloques anónimos no pueden ser referenciados por otras unidades de programa.

Examples

Un ejemplo de un bloque anónimo.

```
DECLARE
    -- declare a variable
    message varchar2(20);
BEGIN
  -- assign value to variable
  message := 'HELLO WORLD';

  -- print message to screen
  DBMS_OUTPUT.PUT_LINE(message);
END;
/
```

Lea Bloque PL / SQL anónimo en línea: https://riptutorial.com/es/oracle/topic/6451/bloque-pl---sql-anonimo

Capítulo 4: Bomba de datos

Introducción

Los siguientes son los pasos para crear una importación / exportación de datos de la bomba:

Examples

Monitorear trabajos de Datapump

Los trabajos de Datapump pueden ser monitoreados usando

1. Vistas del diccionario de datos:

```
   select * from dba_datapump_jobs;
   SELECT * FROM DBA_DATAPUMP_SESSIONS;
   select username,opname,target_desc,sofar,totalwork,message from V$SESSION_LONGOPS where
username = 'bkpadmin';
```

2. Estado del volcado de datos:

- Anote el nombre del trabajo del registro de importación / exportación o el nombre del diccionario de datos y
- Ejecutar comando de **adjuntar** :
- escriba el estado en Importar / Exportar mensaje

```
impdp <bkpadmin>/<bkp123> attach=<SYS_IMPORT_SCHEMA_01>
Import> status
```

Presione Presione **CTRL + C** para salir del indicador Importar / Exportar

Paso 3/6: Crear directorio

```
create or replace directory DATAPUMP_REMOTE_DIR as '/oracle/scripts/expimp';
```

Paso 7: Comandos de exportación

Comandos:

```
expdp <bkpadmin>/<bkp123>  parfile=<exp.par>
```

* Reemplace los datos en <> con los valores adecuados según su entorno. Puede agregar / modificar parámetros según sus requerimientos. En el ejemplo anterior, todos los parámetros restantes se agregan en los archivos de parámetros como se indica a continuación: *

- Tipo de exportación: **Exportación de usuario**
- Exportar esquema completo
- Detalles del archivo de parámetros [decir exp.par]:

```
schemas=<schema>
directory= DATAPUMP_REMOTE_DIR
dumpfile=<dbname>_<schema>.dmp
logfile=exp_<dbname>_<schema>.log
```

- Tipo de exportación: **Exportación de usuario para esquema grande**
- Exportar todo el esquema para grandes conjuntos de datos: aquí los archivos de volcado de exportación se desglosarán y comprimirán. Aquí se usa el paralelismo *(Nota: Agregar paralelismo aumentará la carga de la CPU en el servidor)*
- Detalles del archivo de parámetros [decir exp.par]:

```
schemas=<schema>
directory= DATAPUMP_REMOTE_DIR
dumpfile=<dbname>_<schema>_%U.dmp
logfile=exp_<dbname>_<schema>.log
compression = all
parallel=5
```

- Tipo de exportación: **Exportar tabla** [Exportar conjunto de tablas]
- Detalles del archivo de parámetros [decir exp.par]:

```
tables= tname1, tname2, tname3
directory= DATAPUMP_REMOTE_DIR
dumpfile=<dbname>_<schema>.dmp
logfile=exp_<dbname>_<schema>.log
```

Paso 9: Comandos de importación

Requisito previo:

- Antes de importar usuarios, es una buena práctica eliminar el esquema o la tabla importada.

Comandos:

```
impdp <bkpadmin>/<bkp123>  parfile=<imp.par>
```

* Reemplace los datos en <> con los valores adecuados según su entorno. Puede agregar / modificar parámetros según sus requerimientos. En el ejemplo anterior, todos los parámetros restantes se agregan en los archivos de parámetros como se indica a continuación: *

- Tipo de importación: **importación de usuario**
- Importar esquema completo
- Detalles del archivo de parámetros [decir imp.par]:

```
schemas=<schema>
directory= DATAPUMP_REMOTE_DIR
dumpfile=<dbname>_<schema>.dmp
logfile=imp_<dbname>_<schema>.log
```

- Tipo de importación: **importación de usuario para esquema grande**
- Importe el esquema completo para grandes conjuntos de datos: aquí se usa el paralelismo *(Nota: agregar paralelismo aumentará la carga de la CPU en el servidor)*
- Detalles del archivo de parámetros [decir imp.par]:

```
schemas=<schema>
directory= DATAPUMP_REMOTE_DIR
dumpfile=<dbname>_<schema>_%U.dmp
logfile=imp_<dbname>_<schema>.log
parallel=5
```

- Tipo de importación: **Importar tabla** [Importar conjunto de tablas]
- Detalles del archivo de parámetros [decir imp.par]:

```
tables= tname1, tname2, tname3
directory= DATAPUMP_REMOTE_DIR
dumpfile=<dbname>_<schema>.dmp
logfile=exp_<dbname>_<schema>.log
TABLE_EXISTS_ACTION= <APPEND /SKIP /TRUNCATE /REPLACE>
```

---

1. pasos del datapump

Servidor de origen [Exportar datos]	Servidor de destino [Importar datos]
1. Cree una carpeta datapump que contendrá los archivos de volcado de exportación	4. Cree una carpeta de datapump que contendrá los archivos de volcado de importación
2. Inicie sesión en el esquema de base de datos que realizará la exportación.	5. Inicie sesión en el esquema de base de datos que realizará la importación.
3. Crear directorio apuntando al paso 1.	6. Crear directorio apuntando al paso 4.
7. Ejecutar declaraciones de exportación.	
8. Copie / SCP los archivos de volcado al servidor de destino.	
	9. Ejecutar declaraciones de importación
	10. Verificar datos, compilar objetos inválidos y proporcionar concesiones relacionadas.

Copia tablas entre diferentes esquemas y espacios de tabla

```
 expdp <bkpadmin>/<bkp123> directory=DATAPUMP_REMOTE_DIR dumpfile=<customer.dmp>


 impdp <bkpadmin>/<bkp123> directory=DATAPUMP_REMOTE_DIR dumpfile=<customer.dmp>
remap_schema=<source schema>:<target schema> remap_tablespace=<source tablespace>:<target
tablespace>
```

Lea Bomba de datos en línea: https://riptutorial.com/es/oracle/topic/9391/bomba-de-datos

Capítulo 5: Consejos

Parámetros

Parámetros	Detalles
Grado de Paralelismo (DOP)	Es el número de procesos / conexión en paralelo que desea que se abra su consulta. Por lo general es 2, 4, 8, 16, etc.
Nombre de la tabla	El nombre de la tabla a la que se aplicará la sugerencia paralela.

Examples

Pista paralela

Las sugerencias paralelas a nivel de declaración son las más fáciles:

```
SELECT /*+ PARALLEL(8) */ first_name, last_name FROM employee emp;
```

Las sugerencias paralelas a nivel de objeto dan más control pero son más propensas a errores; los desarrolladores a menudo olvidan usar el alias en lugar del nombre del objeto, u olvidan incluir algunos objetos.

```
SELECT /*+ PARALLEL(emp,8) */ first_name, last_name FROM employee emp;
```

```
SELECT /*+ PARALLEL(table_alias,Degree of Parallelism) */ FROM table_name table_alias;
```

Digamos que una consulta tarda 100 segundos en ejecutarse sin utilizar una sugerencia paralela. Si cambiamos DOP a 2 para la misma consulta, lo *ideal sería que* la misma consulta con una sugerencia paralela tomara 50 segundos. De manera similar, usar DOP como 4 tomará 25 segundos.

En la práctica, la ejecución paralela depende de muchos otros factores y no se escala linealmente. Esto es especialmente cierto para tiempos de ejecución pequeños en los que la sobrecarga paralela puede ser mayor que las ganancias de la ejecución en múltiples servidores paralelos.

USE_NL

Usa bucles anidados.

Uso: `use_nl(AB)`

Esta sugerencia le pedirá al motor que use el método de bucle anidado para unir las tablas A y B. Eso es comparación fila por fila. La sugerencia no fuerza el orden de la unión, solo solicita NL.

```
SELECT /*+use_nl(e d)*/ *
FROM Employees E
JOIN Departments D on E.DepartmentID = D.ID
```

APÉNDATE A LA PISTA

"Usar el método DIRECT PATH para insertar nuevas filas".

La sugerencia `APPEND` indica al motor que use la carga de ruta directa . Esto significa que el motor no utilizará una inserción convencional que use estructuras de memoria y bloqueos estándar, sino que escribirá directamente en el espacio de tablas los datos. Siempre crea nuevos bloques que se agregan al segmento de la tabla. Esto será más rápido, pero tiene algunas limitaciones:

- No puede leer de la tabla que adjuntó en la misma sesión hasta que cometa o restituya la transacción.
- Si hay activadores definidos en la tabla, Oracle no usará la ruta directa (es una historia diferente para las cargas de sqlldr).
- otros

Ejemplo.

```
INSERT /*+append*/ INTO Employees
SELECT *
FROM Employees;
```

USE_HASH

Indica al motor que use el método hash para unir tablas en el argumento.

Uso: `use_hash(TableA [TableB] ... [TableN])`

Como se explica en muchos lugares , "en una combinación HASH, Oracle accede a una tabla (generalmente el más pequeño de los resultados combinados) y crea una tabla hash en la clave de combinación en la memoria. Luego escanea la otra tabla en la combinación (generalmente la más grande). uno) y sondea la tabla hash para coincidencias con ella ".

Se prefiere el método de bucles anidados cuando las tablas son grandes, no hay índices disponibles, etc.

Nota : la sugerencia no fuerza el orden de la unión, solo solicita el método HASH JOIN.

Ejemplo de uso:

```
SELECT /*+use_hash(e d)*/ *
FROM Employees E
JOIN Departments D on E.DepartmentID = D.ID
```

COMPLETO

La sugerencia COMPLETA le dice a Oracle que realice una exploración completa de la tabla en una tabla específica, sin importar si se puede usar un índice.

```
create table fullTable(id) as select level from dual connect by level < 100000;
create index idx on fullTable(id);
```

Sin sugerencias, se utiliza el índice:

```
select count(1) from fullTable f where id between 10 and 100;
-----------------------------------------------------------------------------
| Id  | Operation         | Name | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |      |     1 |    13 |     3   (0)| 00:00:01 |
|   1 |  SORT AGGREGATE   |      |     1 |    13 |            |          |
|*  2 |   INDEX RANGE SCAN| IDX  |     2 |    26 |     3   (0)| 00:00:01 |
-----------------------------------------------------------------------------
```

La sugerencia completa obliga a una exploración completa:

```
select /*+ full(f) */ count(1) from fullTable f where id between 10 and 100;
-----------------------------------------------------------------------------------
| Id  | Operation          | Name      | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |           |     1 |    13 |    47   (3)| 00:00:01 |
|   1 |  SORT AGGREGATE    |           |     1 |    13 |            |          |
|*  2 |   TABLE ACCESS FULL| FULLTABLE |     2 |    26 |    47   (3)| 00:00:01 |
-----------------------------------------------------------------------------------
```

Caché de resultados

Oracle (*11g y superior*) permite que las consultas SQL se almacenen en caché en el SGA y se reutilicen para mejorar el rendimiento. Consulta los datos de la caché en lugar de la base de datos. La ejecución posterior de la misma consulta es más rápida porque ahora los datos se extraen de la memoria caché.

```
SELECT /*+ result_cache */ number FROM main_table;
```

Salida -

```
Number
------
   1
   2
   3
   4
   5
   6
   7
   8
   9
   10

Elapsed: 00:00:02.20
```

Si vuelvo a ejecutar la misma consulta ahora, el tiempo de ejecución se reducirá ya que los datos ahora se recuperan del caché que se estableció durante la primera ejecución.

Salida -

```
Number
------
   1
   2
   3
   4
   5
   6
   7
   8
   9
   10

Elapsed: 00:00:00.10
```

Observe cómo el tiempo transcurrido se redujo de **2,20 segundos** a **0,10 segundos** .

> La memoria caché de resultados mantiene la memoria caché hasta que los datos en la base de datos se actualicen / modifiquen / eliminen. Cualquier cambio liberará el caché.

Lea Consejos en línea: https://riptutorial.com/es/oracle/topic/1490/consejos

Capítulo 6: consulta de nivel

Observaciones

La cláusula de nivel es responsable de generar N número de registros ficticios basados en alguna condición específica.

Examples

Generar N Número de registros

```
SELECT ROWNUM NO FROM DUAL CONNECT BY LEVEL <= 10
```

Pocos usos de Level Query.

/ * Esta es una consulta simple que puede generar una secuencia de números. El siguiente ejemplo genera una secuencia de números de 1..100 * /

```
select level from dual connect by level <= 100;
```

/ * La consulta anterior es útil en varios escenarios, como generar una secuencia de fechas a partir de una fecha determinada. La siguiente consulta genera 10 fechas consecutivas * /

```
select to_date('01-01-2017','mm-dd-yyyy')+level-1 as  dates from dual connect by level <= 10;
```

01-ene-17
02-ENE-17
03-ene-17
04-ene-17
05-ene-17
06-ene-17
07-ene-17
08-ene-17
09-ene-17
10-ene-17

Lea consulta de nivel en línea: https://riptutorial.com/es/oracle/topic/6548/consulta-de-nivel

Capítulo 7: Creando un contexto

Sintaxis

- CREAR [O REEMPLAZAR] el espacio de nombres CONTEXTO USANDO el paquete [schema.];
- CREAR [O REEMPLAZAR] el espacio de nombres CONTEXTO UTILIZANDO el paquete [esquema.] INICIALIZADO EXTERNAMENTE;
- CREAR [O REEMPLAZAR] el espacio de nombres CONTEXTO UTILIZANDO el paquete [esquema.] INICIALIZADO GLOBALMENTE;
- CREAR [O REEMPLAZAR] el espacio de nombres CONTEXTO UTILIZANDO el paquete [schema.] ACCESSED GLOBALLY;

Parámetros

Parámetro	Detalles
`OR REPLACE`	Redefinir un espacio de nombres de contexto existente
espacio de nombres	Nombre del contexto: este es el espacio de nombres para las llamadas a `SYS_CONTEXT`
esquema	Propietario del paquete
paquete	Paquete de base de datos que establece o restablece los atributos de contexto. Nota: el paquete de base de datos no tiene que existir para crear el contexto.
`INITIALIZED`	Especifique una entidad que no sea Oracle Database que pueda establecer el contexto.
`EXTERNALLY`	Permitir que la interfaz OCI inicialice el contexto.
`GLOBALLY`	Permitir que el directorio LDAP inicialice el contexto al establecer la sesión.
`ACCESSED GLOBALLY`	Permita que el contexto sea accesible en toda la instancia: varias sesiones pueden compartir los valores de los atributos siempre que tengan la misma ID de cliente.

Observaciones

Documentación de Oracle (12cR1):
http://docs.oracle.com/database/121/SQLRF/statements_5003.htm

Examples

Crear un contexto

```
CREATE CONTEXT my_ctx USING my_pkg;
```

Esto crea un contexto que solo se puede configurar mediante rutinas en el paquete de base de datos `my_pkg` , por ejemplo:

```
CREATE PACKAGE my_pkg AS
  PROCEDURE set_ctx;
END my_pkg;

CREATE PACKAGE BODY my_pkg AS
  PROCEDURE set_ctx IS
  BEGIN
    DBMS_SESSION.set_context('MY_CTX','THE KEY','Value');
    DBMS_SESSION.set_context('MY_CTX','ANOTHER','Bla');
  END set_ctx;
END my_pkg;
```

Ahora, si una sesión hace esto:

```
my_pkg.set_ctx;
```

Ahora puede recuperar el valor de la clave de esta manera:

```
SELECT SYS_CONTEXT('MY_CTX','THE KEY') FROM dual;

Value
```

Lea Creando un contexto en línea: https://riptutorial.com/es/oracle/topic/2088/creando-un-contexto

Capítulo 8: Delimitando palabras clave o caracteres especiales

Examples

Delimite el nombre de la tabla o columna con caracteres especiales

Seleccione * de firm's_address;

Seleccione * de "firm's_address";

Delimitando el nombre de la tabla o columna, que también es una palabra reservada

Supongamos que tiene una tabla con el nombre de tabla o que desea crear una tabla con nombre que también sea una palabra clave. Debe incluir la tabla de nombres en un par de comillas dobles "tabla"

Seleccione * de la tabla; La consulta anterior fallará con un error de sintaxis, donde la consulta siguiente funcionará bien.

Seleccione * de "tabla";

Lea Delimitando palabras clave o caracteres especiales en línea:
https://riptutorial.com/es/oracle/topic/6553/delimitando-palabras-clave-o-caracteres-especiales

Capítulo 9: Diccionario de datos

Observaciones

Las vistas del diccionario de datos, también conocidas como vistas de catálogo, le permiten monitorear el estado de la base de datos en tiempo real:

Las vistas con el prefijo `USER_` , `ALL_` y `DBA_` muestran información sobre los objetos de esquema que le pertenecen (`USER_`), a los que puede acceder (`ALL_`) o a los que puede acceder un usuario con privilegio de SYSDBA (`DBA_`). Por ejemplo, la vista `ALL_TABLES` muestra todas las tablas en las que tiene privilegios.

Las vistas `V$` muestran información relacionada con el rendimiento.

Las vistas `_PRIVS` muestran información de privilegios para diferentes combinaciones de usuarios, roles y objetos.

Documentación de Oracle: Vistas de catálogo / Vistas de diccionario de datos

Examples

Origen del texto de los objetos almacenados.

`USER_SOURCE` describe la fuente de texto de los objetos almacenados que son propiedad del usuario actual. Esta vista no muestra la columna `OWNER` .

```
select * from user_source where type='TRIGGER' and lower(text) like '%order%'
```

`ALL_SOURCE` describe la fuente de texto de los objetos almacenados accesibles para el usuario actual.

```
select * from all_source where owner=:owner
```

`DBA_SOURCE` describe la fuente de texto de todos los objetos almacenados en la base de datos.

```
select * from dba_source
```

Obtener lista de todas las tablas en Oracle

```
select owner, table_name
from all_tables
```

`ALL_TAB_COLUMNS` describe las columnas de las tablas, vistas y clusters accesibles para el usuario actual. `COLS` es un sinónimo de `USER_TAB_COLUMNS` .

```
select *
from all_tab_columns
where table_name = :tname
```

Información de privilegio

Todos los roles otorgados al usuario.

```
select *
from dba_role_privs
where grantee= :username
```

Privilegios otorgados al usuario:

1. privilegios del sistema

```
select *
from dba_sys_privs
where grantee = :username
```

2. concesión de objeto

```
select *
from dba_tab_privs
where grantee = :username
```

Permisos otorgados a roles.

Roles otorgados a otros roles.

```
select *
from role_role_privs
where role in (select granted_role from dba_role_privs where grantee= :username)
```

1. privilegios del sistema

```
select *
from role_sys_privs
where role in (select granted_role from dba_role_privs where grantee= :username)
```

2. concesión de objeto

```
select *
from role_tab_privs
where role in (select granted_role from dba_role_privs where grantee= :username)
```

Versión de Oracle

```
select *
from v$version
```

Describe todos los objetos en la base de datos.

```
select *
from dba_objects
```

Para ver todas las vistas del diccionario de datos a las que tiene acceso.

```
select * from dict
```

Lea Diccionario de datos en línea: https://riptutorial.com/es/oracle/topic/7347/diccionario-de-datos

Capítulo 10: Diferentes formas de actualizar registros.

Sintaxis

- ACTUALIZAR nombre-tabla [[AS] nombre-correlación] SET columna-Nombre = Valor [, columna-Nombre = Valor}] * [cláusula WHERE]
- ACTUALIZAR nombre-tabla AJUSTE columna-Nombre = Valor [, columna-Nombre = Valor] * DÓNDE ACTUALMENTE

Examples

Actualice la sintaxis con el ejemplo

Actualización normal

```
UPDATE
      TESTTABLE
SET
      TEST_COLUMN= 'Testvalue',TEST_COLUMN2= 123
WHERE
      EXISTS
          (SELECT MASTERTABLE.TESTTABLE_ID
           FROM MASTERTABLE
           WHERE ID_NUMBER=11);
```

Actualizar usando la vista en línea

Uso de la vista en línea (si Oracle lo considera actualizable)

Nota : Si se enfrenta a un error de fila no conservada por clave, agregue un índice para resolver el mismo y así poder actualizarlo.

```
UPDATE
      (SELECT
            TESTTABLE.TEST_COLUMN AS OLD,
            'Testvalue' AS NEW
       FROM
                TESTTABLE
            INNER JOIN
                MASTERTABLE
            ON TESTTABLE.TESTTABLE_ID = MASTERTABLE.TESTTABLE_ID
       WHERE ID_NUMBER=11) T
SET
      T.OLD        = T.NEW;
```

Actualizar utilizando Merge

Utilizando Merge

```
MERGE INTO
      TESTTABLE
USING
      (SELECT
            T1.ROWID AS RID,
            T2.TESTTABLE_ID
       FROM
                TESTTABLE T1
            INNER JOIN
                MASTERTABLE T2
            ON TESTTABLE.TESTTABLE_ID = MASTERTABLE.TESTTABLE_ID
       WHERE ID_NUMBER=11)
ON
      ( ROWID = RID )
WHEN MATCHED
THEN
    UPDATE SET TEST_COLUMN= 'Testvalue';
```

Combinar con datos de muestra

```
drop table table01;
drop table table02;

create table table01 (
       code int,
       name varchar(50),
       old int
);

create table table02 (
       code int,
       name varchar(50),
       old int
);

truncate table table01;
insert into table01 values (1, 'A', 10);
insert into table01 values (9, 'B', 12);
insert into table01 values (3, 'C', 14);
insert into table01 values (4, 'D', 16);
insert into table01 values (5, 'E', 18);

truncate table table02;
insert into table02 values (1, 'AA', null);
insert into table02 values (2, 'BB', 123);
insert into table02 values (3, 'CC', null);
insert into table02 values (4, 'DD', null);
insert into table02 values (5, 'EE', null);

select * from table01 a order by 2;
select * from table02 a order by 2;

--

merge into table02 a using (
      select b.code, b.old from table01 b
) c on (
```

```
  a.code = c.code
)
when matched then update set a.old = c.old
;

--

select a.*, b.* from table01 a
inner join table02 b on a.code = b.codetable01;

select * from table01 a
where
       exists
       (
         select 'x' from table02 b where a.code = b.codetable01
       );

select * from table01 a where a.code in (select b.codetable01 from table02 b);

--

select * from table01 a
where
       not exists
       (
         select 'x' from table02 b where a.code = b.codetable01
       );

select * from table01 a where a.code not in (select b.codetable01 from table02 b);
```

Lea Diferentes formas de actualizar registros. en línea:
https://riptutorial.com/es/oracle/topic/4193/diferentes-formas-de-actualizar-registros-

Capítulo 11: División de cadenas delimitadas

Examples

División de cadenas mediante una cláusula de factorización de subconsulta recursiva

Datos de muestra :

```
CREATE TABLE table_name ( id, list ) AS
SELECT 1, 'a,b,c,d' FROM DUAL UNION ALL -- Multiple items in the list
SELECT 2, 'e'       FROM DUAL UNION ALL -- Single item in the list
SELECT 3, NULL      FROM DUAL UNION ALL -- NULL list
SELECT 4, 'f,,g'    FROM DUAL;          -- NULL item in the list
```

Consulta :

```
WITH bounds ( id, list, start_pos, end_pos, lvl ) AS (
  SELECT id, list, 1, INSTR( list, ',' ), 1 FROM table_name
UNION ALL
  SELECT id,
         list,
         end_pos + 1,
         INSTR( list, ',', end_pos + 1 ),
         lvl + 1
  FROM   bounds
  WHERE  end_pos > 0
)
SELECT id,
       SUBSTR(
         list,
         start_pos,
         CASE end_pos
           WHEN 0
           THEN LENGTH( list ) + 1
           ELSE end_pos
         END - start_pos
       ) AS item,
      lvl
FROM   bounds
ORDER BY id, lvl;
```

Salida :

```
        ID ITEM           LVL
---------- ------- ----------
         1 a                1
         1 b                2
         1 c                3
         1 d                4
         2 e                1
         3 (NULL)           1
         4 f                1
```

```
         4 (NULL)           2
         4 g                3
```

División de cadenas utilizando una función PL / SQL

Función PL / SQL :

```
CREATE OR REPLACE FUNCTION split_String(
  i_str    IN  VARCHAR2,
  i_delim  IN  VARCHAR2 DEFAULT ','
) RETURN SYS.ODCIVARCHAR2LIST DETERMINISTIC
AS
  p_result       SYS.ODCIVARCHAR2LIST := SYS.ODCIVARCHAR2LIST();
  p_start        NUMBER(5) := 1;
  p_end          NUMBER(5);
  c_len CONSTANT NUMBER(5) := LENGTH( i_str );
  c_ld  CONSTANT NUMBER(5) := LENGTH( i_delim );
BEGIN
  IF c_len > 0 THEN
    p_end := INSTR( i_str, i_delim, p_start );
    WHILE p_end > 0 LOOP
      p_result.EXTEND;
      p_result( p_result.COUNT ) := SUBSTR( i_str, p_start, p_end - p_start );
      p_start := p_end + c_ld;
      p_end := INSTR( i_str, i_delim, p_start );
    END LOOP;
    IF p_start <= c_len + 1 THEN
      p_result.EXTEND;
      p_result( p_result.COUNT ) := SUBSTR( i_str, p_start, c_len - p_start + 1 );
    END IF;
  END IF;
  RETURN p_result;
END;
/
```

Datos de muestra :

```
CREATE TABLE table_name ( id, list ) AS
SELECT 1, 'a,b,c,d' FROM DUAL UNION ALL -- Multiple items in the list
SELECT 2, 'e'       FROM DUAL UNION ALL -- Single item in the list
SELECT 3, NULL      FROM DUAL UNION ALL -- NULL list
SELECT 4, 'f,,g'    FROM DUAL;          -- NULL item in the list
```

Consulta :

```
SELECT t.id,
       v.column_value AS value,
       ROW_NUMBER() OVER ( PARTITION BY id ORDER BY ROWNUM ) AS lvl
FROM   table_name t,
       TABLE( split_String( t.list ) ) (+) v
```

Salida :

```
        ID ITEM           LVL
---------- ------- ----------
```

```
         1 a                1
         1 b                2
         1 c                3
         1 d                4
         2 e                1
         3 (NULL)           1
         4 f                1
         4 (NULL)           2
         4 g                3
```

División de cadenas utilizando una expresión de tabla correlacionada

Datos de muestra :

```
CREATE TABLE table_name ( id, list ) AS
SELECT 1, 'a,b,c,d' FROM DUAL UNION ALL -- Multiple items in the list
SELECT 2, 'e'       FROM DUAL UNION ALL -- Single item in the list
SELECT 3, NULL      FROM DUAL UNION ALL -- NULL list
SELECT 4, 'f,,g'    FROM DUAL;          -- NULL item in the list
```

Consulta :

```
SELECT t.id,
       v.COLUMN_VALUE AS value,
       ROW_NUMBER() OVER ( PARTITION BY id ORDER BY ROWNUM ) AS lvl
FROM   table_name t,
       TABLE(
         CAST(
           MULTISET(
             SELECT REGEXP_SUBSTR( t.list, '([^,]*)(,|$)', 1, LEVEL, NULL, 1 )
             FROM   DUAL
             CONNECT BY LEVEL < REGEXP_COUNT( t.list, '[^,]*(,|$)' )
           )
           AS SYS.ODCIVARCHAR2LIST
         )
       ) v;
```

Salida :

```
        ID ITEM           LVL
---------- ------- ----------
         1 a                1
         1 b                2
         1 c                3
         1 d                4
         2 e                1
         3 (NULL)           1
         4 f                1
         4 (NULL)           2
         4 g                3
```

División de cadenas utilizando una consulta jerárquica

Datos de muestra :

```
CREATE TABLE table_name ( id, list ) AS
SELECT 1, 'a,b,c,d' FROM DUAL UNION ALL -- Multiple items in the list
SELECT 2, 'e'       FROM DUAL UNION ALL -- Single item in the list
SELECT 3, NULL      FROM DUAL UNION ALL -- NULL list
SELECT 4, 'f,,g'    FROM DUAL;          -- NULL item in the list
```

Consulta :

```
SELECT t.id,
       REGEXP_SUBSTR( list, '([^,]*)(,|$)', 1, LEVEL, NULL, 1 ) AS value,
       LEVEL AS lvl
FROM   table_name t
CONNECT BY
       id = PRIOR id
AND    PRIOR SYS_GUID() IS NOT NULL
AND    LEVEL < REGEXP_COUNT( list, '([^,]*)(,|$)' )
```

Salida :

```
        ID ITEM           LVL
---------- ------- ----------
         1 a                1
         1 b                2
         1 c                3
         1 d                4
         2 e                1
         3 (NULL)           1
         4 f                1
         4 (NULL)           2
         4 g                3
```

División de cadenas utilizando expresiones XMLTable y FLWOR

Esta solución utiliza la función `ora:tokenize` XQuery que está disponible en Oracle 11.

Datos de muestra :

```
CREATE TABLE table_name ( id, list ) AS
SELECT 1, 'a,b,c,d' FROM DUAL UNION ALL -- Multiple items in the list
SELECT 2, 'e'       FROM DUAL UNION ALL -- Single item in the list
SELECT 3, NULL      FROM DUAL UNION ALL -- NULL list
SELECT 4, 'f,,g'    FROM DUAL;          -- NULL item in the list
```

Consulta :

```
SELECT t.id,
       x.item,
       x.lvl
FROM   table_name t,
       XMLTABLE(
         'let $list := ora:tokenize(.,","),
              $cnt := count($list)
          for $val at $r in $list
          where $r < $cnt
          return $val'
```

```
       PASSING list||','
       COLUMNS
         item VARCHAR2(100) PATH '.',
         lvl FOR ORDINALITY
       ) (+) x;
```

Salida :

```
        ID ITEM           LVL
---------- ------- ----------
         1 a                1
         1 b                2
         1 c                3
         1 d                4
         2 e                1
         3 (NULL)       (NULL)
         4 f                1
         4 (NULL)           2
         4 g                3
```

División de cadenas utilizando CROSS APPLY (Oracle 12c)

Datos de muestra :

```
CREATE TABLE table_name ( id, list ) AS
SELECT 1, 'a,b,c,d' FROM DUAL UNION ALL -- Multiple items in the list
SELECT 2, 'e'       FROM DUAL UNION ALL -- Single item in the list
SELECT 3, NULL      FROM DUAL UNION ALL -- NULL list
SELECT 4, 'f,,g'    FROM DUAL;          -- NULL item in the list
```

Consulta :

```
SELECT t.id,
       REGEXP_SUBSTR( t.list, '([^,]*)($|,)', 1, l.lvl, NULL, 1 ) AS item,
       l.lvl
FROM   table_name t
       CROSS APPLY
       (
         SELECT LEVEL AS lvl
         FROM   DUAL
         CONNECT BY LEVEL <= REGEXP_COUNT( t.list, ',' ) + 1
       ) l;
```

Salida :

```
        ID ITEM           LVL
---------- ------- ----------
         1 a                1
         1 b                2
         1 c                3
         1 d                4
         2 e                1
         3 (NULL)           1
         4 f                1
         4 (NULL)           2
```

```
         4 g                3
```

División de cadenas delimitadas usando XMLTable

Datos de muestra :

```
CREATE TABLE table_name ( id, list ) AS
SELECT 1, 'a,b,c,d' FROM DUAL UNION ALL -- Multiple items in the list
SELECT 2, 'e'       FROM DUAL UNION ALL -- Single item in the list
SELECT 3, NULL      FROM DUAL UNION ALL -- NULL list
SELECT 4, 'f,,g'    FROM DUAL;          -- NULL item in the list
```

Consulta :

```
SELECT t.id,
       SUBSTR( x.item.getStringVal(), 2 ) AS item,
       x.lvl
FROM   table_name t
       CROSS JOIN
       XMLTABLE(
         ( '"#' || REPLACE( t.list, ',', '","#' ) || '"' )
         COLUMNS item XMLTYPE PATH '.',
                 lvl  FOR ORDINALITY
       ) x;
```

(Nota: el carácter `#` se adjunta para facilitar la extracción de valores `NULL` ; luego se elimina mediante `SUBSTR( item, 2 )` . Si no se requieren valores `NULL` , puede simplificar la consulta y omitir esto).

Salida :

```
        ID ITEM            LVL
---------- ------- ----------
         1 a                 1
         1 b                 2
         1 c                 3
         1 d                 4
         2 e                 1
         3 (NULL)            1
         4 f                 1
         4 (NULL)            2
         4 g                 3
```

Lea División de cadenas delimitadas en línea: https://riptutorial.com/es/oracle/topic/1968/division-de-cadenas-delimitadas

Capítulo 12: Enlaces de base de datos

Examples

Creando un enlace de base de datos

```
CREATE DATABASE LINK dblink_name
CONNECT TO remote_username
IDENTIFIED BY remote_password
USING 'tns_service_name';
```

El DB remoto será accesible de la siguiente manera:

```
SELECT * FROM MY_TABLE@dblink_name;
```

Para probar una conexión de enlace de base de datos sin necesidad de conocer ninguno de los nombres de objeto en la base de datos vinculada, use la siguiente consulta:

```
SELECT * FROM DUAL@dblink_name;
```

Para especificar explícitamente un dominio para el servicio de base de datos vinculado, el nombre de dominio se agrega a la declaración `USING` . Por ejemplo:

```
USING 'tns_service_name.WORLD'
```

Si no se especifica explícitamente un nombre de dominio, Oracle usa el dominio de la base de datos en la que se crea el enlace.

Documentación de Oracle para la creación de enlace de base de datos:

- 10g: https://docs.oracle.com/cd/B19306_01/server.102/b14200/statements_5005.htm
- 11g: https://docs.oracle.com/cd/B28359_01/server.111/b28310/ds_concepts002.htm
- 12g: https://docs.oracle.com/database/121/SQLRF/statements_5006.htm#SQLRF01205

Crear enlace de base de datos

Supongamos que tenemos dos bases de datos "ORA1" y "ORA2". Podemos acceder a los objetos de "ORA2" desde la base de datos "ORA1" utilizando un enlace de base de datos.

Requisitos previos: para crear un enlace de base de datos privado necesita un privilegio `CREATE DATABASE LINK` . Para crear un enlace de base de datos privado, necesita el privilegio `CREATE PUBLIC DATABASE LINK` .

* Oracle Net debe estar presente en ambas instancias.

Cómo crear un enlace de base de datos:

Desde ORA1:

```
SQL> create <public> database link ora2 connect to user1 identified by pass1 using <tns name
of ora2>;
```

Enlace de base de datos creado.

Ahora que hemos configurado el enlace DB, podemos demostrar que ejecutando lo siguiente desde ORA1:

```
SQL> Select name from V$DATABASE@ORA2; -- should return ORA2
```

También puede acceder a los Objetos de DB de "ORA2" desde "ORA1", dado que el usuario `user1` tiene el privilegio `SELECT` sobre esos objetos en ORA2 (como la TABLA 1 a continuación):

```
  SELECT COUNT(*) FROM TABLE1@ORA2;
```

Pre-requisitos:

- Ambas bases de datos deben estar en funcionamiento (abiertas).
- Ambos oyentes de la base de datos deben estar en funcionamiento.
- TNS debe estar configurado correctamente.
- El usuario user1 debe estar presente en la base de datos ORA2, la contraseña debe ser verificada y verificada.
- El usuario user1 debe tener al menos el privilegio `SELECT` , o cualquier otro requerido para acceder a los objetos en ORA2.

Lea Enlaces de base de datos en línea: https://riptutorial.com/es/oracle/topic/3859/enlaces-de-base-de-datos

Capítulo 13: Factoraje de subconsultas recursivas utilizando la cláusula WITH (expresiones de tabla comunes AKA)

Observaciones

La factorización de subconsultas recursiva está disponible en Oracle 11g R2.

Examples

Un simple generador de enteros

Consulta :

```
WITH generator ( value ) AS (
  SELECT 1 FROM DUAL
UNION ALL
  SELECT value + 1
  FROM   generator
  WHERE  value < 10
)
SELECT value
FROM   generator;
```

Salida :

```
VALUE
-----
    1
    2
    3
    4
    5
    6
    7
    8
    9
   10
```

Dividir una cadena delimitada

Datos de muestra :

```
CREATE TABLE table_name ( value VARCHAR2(50) );

INSERT INTO table_name ( value ) VALUES ( 'A,B,C,D,E' );
```

Consulta :

```
WITH items ( list, item, lvl ) AS (
  SELECT value,
         REGEXP_SUBSTR( value, '[^,]+', 1, 1 ),
         1
  FROM   table_name
UNION ALL
  SELECT value,
         REGEXP_SUBSTR( value, '[^,]+', 1, lvl + 1 ),
         lvl + 1
  FROM   items
  WHERE  lvl < REGEXP_COUNT( value, '[^,]+' )
)
SELECT * FROM items;
```

Salida :

```
LIST      ITEM LVL
--------- ---- ---
A,B,C,D,E    A   1
A,B,C,D,E    B   2
A,B,C,D,E    C   3
A,B,C,D,E    D   4
A,B,C,D,E    E   5
```

Lea Factoraje de subconsultas recursivas utilizando la cláusula WITH (expresiones de tabla comunes AKA) en línea: https://riptutorial.com/es/oracle/topic/3506/factoraje-de-subconsultas-recursivas-utilizando-la-clausula-with--expresiones-de-tabla-comunes-aka-

Capítulo 14: fechas

Examples

Fechas de generación sin componente de tiempo

Todos los `DATE` s tienen un componente de tiempo; sin embargo, es habitual almacenar fechas que no necesitan incluir información de tiempo con las horas / minutos / segundos establecidos en cero (es decir, medianoche).

Use un literal ANSI `DATE` (utilizando el formato de fecha ISO 8601):

```
SELECT DATE '2000-01-01' FROM DUAL;
```

Conviértalo de un literal de cadena usando `TO_DATE()` :

```
SELECT TO_DATE( '2001-01-01', 'YYYY-MM-DD' ) FROM DUAL;
```

(Puede encontrar más información sobre los modelos de formato de fecha en la documentación de Oracle).

o:

```
SELECT TO_DATE(
         'January 1, 2000, 00:00 A.M.',
         'Month dd, YYYY, HH12:MI A.M.',
         'NLS_DATE_LANGUAGE = American'
       )
FROM   DUAL;
```

(Si está convirtiendo términos específicos del idioma, como los nombres de los meses, es una buena práctica incluir el tercer parámetro `nlsparam` en la función `TO_DATE()` y especificar el idioma que se espera).

Generando fechas con un componente de tiempo

Conviértalo de un literal de cadena usando `TO_DATE()` :

```
SELECT TO_DATE( '2000-01-01 12:00:00', 'YYYY-MM-DD HH24:MI:SS' ) FROM DUAL;
```

O use un literal de `TIMESTAMP` :

```
CREATE TABLE date_table(
  date_value DATE
);

INSERT INTO date_table ( date_value ) VALUES ( TIMESTAMP '2000-01-01 12:00:00' );
```

Oracle implícitamente emitirá un `TIMESTAMP` a una `DATE` cuando lo almacene en una columna `DATE` de una tabla; sin embargo, puede `CAST()` explícitamente `CAST()` el valor a una `DATE` :

```
SELECT CAST( TIMESTAMP '2000-01-01 12:00:00' AS DATE ) FROM DUAL;
```

El formato de una fecha

En Oracle, un tipo de datos `DATE` no tiene un formato; cuando Oracle envía una `DATE` al programa cliente (SQL / Plus, SQL / Developer, Toad, Java, Python, etc.) enviará 7- u 8 bytes que representan la fecha.

Una `DATE` que no se almacena en una tabla (es decir, generada por `SYSDATE` y que tiene "tipo 13" cuando se usa el comando `DUMP()`) tiene 8 bytes y tiene la estructura (los números a la derecha son la representación interna de `2012-11-26 16:41:09`):

```
BYTE VALUE                          EXAMPLE
---- ------------------------------ ---------------------------------------
1    Year modulo 256                220
2    Year multiples of 256          7   (7 * 256 + 220 = 2012)
3    Month                          11
4    Day                            26
5    Hours                          16
6    Minutes                        41
7    Seconds                        9
8    Unused                         0
```

Una `DATE` que se almacena en una tabla ("tipo 12" cuando se usa el comando `DUMP()`) tiene 7 bytes y tiene la estructura (los números a la derecha son la representación interna de `2012-11-26 16:41:09`):

```
BYTE VALUE                          EXAMPLE
---- ------------------------------ ---------------------------------------
1    ( Year multiples of 100 ) + 100 120
2    ( Year modulo 100 ) + 100      112 ((120-100)*100 + (112-100) = 2012)
3    Month                          11
4    Day                            26
5    Hours + 1                      17
6    Minutes + 1                    42
7    Seconds + 1                    10
```

Si desea que la fecha tenga un formato específico, deberá convertirlo a algo que tenga un formato (es decir, una cadena). El cliente SQL puede hacer esto implícitamente o puede convertir explícitamente el valor en una cadena usando `TO_CHAR( date, format_model, nls_params )` .

Convertir fechas a una cadena

Use `TO_CHAR( date [, format_model [, nls_params]] )` :

(Nota: si no se proporciona un modelo de formato, entonces se `NLS_DATE_FORMAT` parámetro de sesión `NLS_DATE_FORMAT` como el modelo de formato predeterminado ; esto puede ser diferente para cada sesión, por lo que no debe confiarse. Es una buena práctica especificar siempre el modelo

de formato).

```
CREATE TABLE table_name (
  date_value DATE
);

INSERT INTO table_name ( date_value ) VALUES ( DATE '2000-01-01' );
INSERT INTO table_name ( date_value ) VALUES ( TIMESTAMP '2016-07-21 08:00:00' );
INSERT INTO table_name ( date_value ) VALUES ( SYSDATE );
```

Entonces:

```
SELECT TO_CHAR( date_value, 'YYYY-MM-DD' ) AS formatted_date FROM table_name;
```

Salidas:

```
FORMATTED_DATE
--------------
2000-01-01
2016-07-21
2016-07-21
```

Y:

```
SELECT TO_CHAR(
         date_value,
         'FMMonth d yyyy, hh12:mi:ss AM',
         'NLS_DATE_LANGUAGE = French'
       ) AS formatted_date
FROM   table_name;
```

Salidas:

```
FORMATTED_DATE
-----------------------------
Janvier   01 2000, 12:00:00 AM
Juillet   21 2016, 08:00:00 AM
Juillet   21 2016, 19:08:31 PM
```

Configuración del modelo de formato de fecha predeterminado

Cuando Oracle se convierte implícitamente de una `DATE` a una cadena o viceversa (o cuando `TO_CHAR()` o `TO_DATE()` se llaman explícitamente sin un modelo de formato), el parámetro de sesión `NLS_DATE_FORMAT` se usará como el modelo de formato en la conversión. Si el literal no coincide con el modelo de formato, se generará una excepción.

Puedes revisar este parámetro usando:

```
SELECT VALUE FROM NLS_SESSION_PARAMETERS WHERE PARAMETER = 'NLS_DATE_FORMAT';
```

Puede establecer este valor dentro de su sesión actual usando:

```
ALTER SESSION SET NLS_DATE_FORMAT = 'YYYY-MM-DD HH24:MI:SS';
```

(Nota: esto no cambia el valor para ningún otro usuario).

Si confía en `NLS_DATE_FORMAT` para proporcionar la máscara de formato en `TO_DATE()` o `TO_CHAR()` , no debería sorprenderse cuando sus consultas se interrumpen si este valor se modifica.

Cómo cambiar las fechas de visualización de SQL / Plus o SQL Developer

Cuando SQL / Plus o SQL Developer muestran fechas, realizarán una conversión implícita a una cadena utilizando el modelo de formato de fecha predeterminado (consulte el ejemplo Configuración del modelo de formato de fecha predeterminado).

Puede cambiar la forma en que se muestra una fecha cambiando el parámetro `NLS_DATE_FORMAT` .

Aritmética de fechas: diferencia entre fechas en días, horas, minutos y / o segundos

En Oracle, la diferencia (en días y / o fracciones de los mismos) entre dos `DATE` se puede encontrar mediante la resta:

```
SELECT DATE '2016-03-23' - DATE '2015-12-25' AS difference FROM DUAL;
```

Muestra el número de días entre las dos fechas:

```
DIFFERENCE
----------
        89
```

Y:

```
SELECT TO_DATE( '2016-01-02 01:01:12', 'YYYY-MM-DD HH24:MI:SS' )
         - TO_DATE( '2016-01-01 00:00:00', 'YYYY-MM-DD HH24:MI:SS' )
         AS difference
FROM   DUAL
```

Produce la fracción de días entre dos fechas:

```
DIFFERENCE
----------
    1.0425
```

La diferencia en horas, minutos o segundos puede encontrarse multiplicando este número por `24` , `24*60` o `24*60*60` respectivamente.

El ejemplo anterior se puede cambiar para obtener los días, horas, minutos y segundos entre dos fechas utilizando:

```
SELECT TRUNC( difference                       ) AS days,
```

```
       TRUNC( MOD( difference * 24,       24 ) ) AS hours,
       TRUNC( MOD( difference * 24*60,    60 ) ) AS minutes,
       TRUNC( MOD( difference * 24*60*60, 60 ) ) AS seconds
FROM   (
  SELECT TO_DATE( '2016-01-02 01:01:12', 'YYYY-MM-DD HH24:MI:SS' )
         - TO_DATE( '2016-01-01 00:00:00', 'YYYY-MM-DD HH24:MI:SS' )
         AS difference
  FROM   DUAL
);
```

(Nota: `TRUNC()` se usa en lugar de `FLOOR()` para manejar correctamente las diferencias negativas.)

Salidas:

```
DAYS HOURS MINUTES SECONDS
---- ----- ------- -------
   1     1       1      12
```

El ejemplo anterior también se puede resolver convirtiendo la diferencia numérica en un intervalo usando `NUMTODSINTERVAL()` :

```
SELECT EXTRACT( DAY    FROM difference ) AS days,
       EXTRACT( HOUR   FROM difference ) AS hours,
       EXTRACT( MINUTE FROM difference ) AS minutes,
       EXTRACT( SECOND FROM difference ) AS seconds
FROM   (
  SELECT NUMTODSINTERVAL(
           TO_DATE( '2016-01-02 01:01:12', 'YYYY-MM-DD HH24:MI:SS' )
             - TO_DATE( '2016-01-01 00:00:00', 'YYYY-MM-DD HH24:MI:SS' ),
           'DAY'
         ) AS difference
  FROM   DUAL
);
```

Aritmética de fechas: diferencia entre fechas en meses o años

La diferencia en meses entre dos fechas se puede encontrar utilizando el `MONTHS_BETWEEN( date1, date2 )` :

```
SELECT MONTHS_BETWEEN( DATE '2016-03-10', DATE '2015-03-10' ) AS difference FROM DUAL;
```

Salidas:

```
DIFFERENCE
----------
        12
```

Si la diferencia incluye meses parciales, devolverá la fracción del mes según que haya **31** días en cada mes:

```
SELECT MONTHS_BETWEEN( DATE '2015-02-15', DATE '2015-01-01' ) AS difference FROM DUAL;
```

Salidas:

```
DIFFERENCE
----------
 1.4516129
```

Debido a que `MONTHS_BETWEEN` supone 31 días al mes cuando puede haber menos días al mes, esto puede `MONTHS_BETWEEN` valores diferentes para las diferencias que abarcan los límites entre los meses.

Ejemplo:

```
SELECT MONTHS_BETWEEN( DATE'2016-02-01', DATE'2016-02-01' - INTERVAL '1' DAY ) AS "JAN-FEB",
       MONTHS_BETWEEN( DATE'2016-03-01', DATE'2016-03-01' - INTERVAL '1' DAY ) AS "FEB-MAR",
       MONTHS_BETWEEN( DATE'2016-04-01', DATE'2016-04-01' - INTERVAL '1' DAY ) AS "MAR-APR",
       MONTHS_BETWEEN( DATE'2016-05-01', DATE'2016-05-01' - INTERVAL '1' DAY ) AS "APR-MAY"
FROM   DUAL;
```

Salida:

```
JAN-FEB FEB-MAR MAR-APR APR-MAY
------- ------- ------- -------
0.03226 0.09677 0.03226 0.06452
```

La diferencia en años se puede encontrar al dividir la diferencia del mes entre 12.

Extraiga los componentes del año, mes, día, hora, minuto o segundo de una fecha

Los componentes de año, mes o día de un tipo de datos `DATE` se pueden encontrar utilizando `EXTRACT( [ YEAR | MONTH | DAY ] FROM datevalue )`

```
SELECT EXTRACT (YEAR  FROM DATE '2016-07-25') AS YEAR,
       EXTRACT (MONTH FROM DATE '2016-07-25') AS MONTH,
       EXTRACT (DAY   FROM DATE '2016-07-25') AS DAY
FROM DUAL;
```

Salidas:

```
YEAR MONTH DAY
---- ----- ---
2016     7  25
```

Los componentes de tiempo (hora, minuto o segundo) se pueden encontrar por:

- Uso de `CAST( datevalue AS TIMESTAMP )` para convertir la `DATE` en un `TIMESTAMP` y luego usar `EXTRACT( [ HOUR | MINUTE | SECOND ] FROM timestampvalue )` ; o
- Usando `TO_CHAR( datevalue, format_model )` para obtener el valor como una cadena.

Por ejemplo:

```
SELECT EXTRACT( HOUR   FROM CAST( datetime AS TIMESTAMP ) ) AS Hours,
       EXTRACT( MINUTE FROM CAST( datetime AS TIMESTAMP ) ) AS Minutes,
       EXTRACT( SECOND FROM CAST( datetime AS TIMESTAMP ) ) AS Seconds
FROM   (
  SELECT TO_DATE( '2016-01-01 09:42:01', 'YYYY-MM-DD HH24:MI:SS' ) AS datetime FROM DUAL
);
```

Salidas:

```
HOURS MINUTES SECONDS
----- ------- -------
    9      42       1
```

Zonas horarias y horario de verano

El tipo de datos `DATE` no controla las zonas horarias ni los cambios en el horario de verano.

Ya sea:

- use el tipo de datos `TIMESTAMP WITH TIME ZONE` ; o
- Manejar los cambios en la lógica de su aplicación.

Una `DATE` se puede almacenar como hora universal coordinada (UTC) y se puede convertir a la zona horaria de la sesión actual de esta manera:

```
SELECT FROM_TZ(
         CAST(
           TO_DATE( '2016-01-01 12:00:00', 'YYYY-MM-DD HH24:MI:SS' )
           AS TIMESTAMP
         ),
         'UTC'
       )
       AT LOCAL AS time
FROM   DUAL;
```

Si ejecuta `ALTER SESSION SET TIME_ZONE = '+01:00';` entonces la salida es:

```
TIME
-----------------------------------
2016-01-01 13:00:00.000000000 +01:00
```

y `ALTER SESSION SET TIME_ZONE = 'PST';` entonces la salida es:

```
TIME
-----------------------------------
2016-01-01 04:00:00.000000000 PST
```

Leap Seconds

Oracle no maneja segundos de salto . Consulte la nota `2019397.2` y `730795.1` de My Oracle Support para obtener más detalles.

Obtención del día de la semana

Puede usar `TO_CHAR( date_value, 'D' )` para obtener el día de la semana.

Sin embargo, esto depende del parámetro de sesión `NLS_TERRITORY` :

```
ALTER SESSION SET NLS_TERRITORY = 'AMERICA';        -- First day of week is Sunday
SELECT TO_CHAR( DATE '1970-01-01', 'D' ) FROM DUAL;
```

Salidas `5`

```
ALTER SESSION SET NLS_TERRITORY = 'UNITED KINGDOM'; -- First day of week is Monday
SELECT TO_CHAR( DATE '1970-01-01', 'D' ) FROM DUAL;
```

Salidas `4`

Para hacer esto independientemente de la configuración de `NLS` , puede truncar la fecha hasta la medianoche del día actual (para eliminar cualquier fracción de días) y restar la fecha truncada al inicio de la iso-semana actual (que siempre comienza el lunes):

```
SELECT TRUNC( date_value ) - TRUNC( date_value, 'IW' ) + 1 FROM DUAL
```

Lea fechas en línea: https://riptutorial.com/es/oracle/topic/2087/fechas

Capítulo 15: Funciones de ventana

Sintaxis

- Ratio_To_Report (expr) OVER (query_partition_clause)

Examples

Ratio_To_Report

Proporciona la relación del valor de las filas actuales a todos los valores dentro de la ventana.

```
--Data
CREATE TABLE Employees (Name Varchar2(30), Salary Number(10));
INSERT INTO Employees Values ('Bob',2500);
INSERT INTO Employees Values ('Alice',3500);
INSERT INTO Employees Values ('Tom',2700);
INSERT INTO Employees Values ('Sue',2000);
--Query
SELECT Name, Salary, Ratio_To_Report(Salary) OVER () As Ratio
FROM Employees
ORDER BY Salary, Name, Ratio;
--Output
NAME                           SALARY      RATIO
------------------------------ ---------- ----------
Sue                                  2000 .186915888
Bob                                  2500  .23364486
Tom                                  2700 .252336449
Alice                                3500 .327102804
```

Lea Funciones de ventana en línea: https://riptutorial.com/es/oracle/topic/6669/funciones-de-ventana

Capítulo 16: Funciones estadisticas

Examples

Cálculo de la mediana de un conjunto de valores.

La función MEDIANA desde Oracle 10g es una función de agregación fácil de usar:

```
SELECT MEDIAN(SAL)
FROM EMP
```

Devuelve la mediana de los valores.

Funciona en los valores `DATETIME` también.

> El resultado de MEDIAN se calcula ordenando primero las filas. Usando N como el número de filas en el grupo, Oracle calcula el número de fila (RN) de interés con la fórmula RN = (1 + (0.5 * (N-1)). El resultado final de la función agregada se calcula de forma lineal interpolación entre los valores de las filas en los números de fila CRN = CEILING (RN) y FRN = FLOOR (RN).

Desde Oracle 9i puede usar PERCENTILE_CONT que funciona igual que la función MEDIAN con valores predeterminados de valor percentil de 0.5

```
SELECT PERCENTILE_CONT(.5) WITHIN GROUP(order by SAL)
FROM EMP
```

DIFERENCIA

La varianza mide qué tan lejos se distribuyen los números determinados de su media. Desde la perspectiva práctica, es la distancia al cuadrado de su media (centro): cuanto mayor sea el número, más lejos estará el punto.

El siguiente ejemplo devolvería la varianza de los valores salariales

```
SELECT name, salary, VARIANCE(salary) "Variance"
FROM employees
```

STDDEV

STDDEV devuelve la muestra de desviación estándar de expr, un conjunto de números. Puedes usarlo tanto como una función agregada como analítica. Difiere de STDDEV_SAMP en que STDDEV devuelve cero cuando tiene solo 1 fila de datos de entrada, mientras que STDDEV_SAMP devuelve nulo.

La base de datos Oracle calcula la desviación estándar como la raíz cuadrada de la varianza

definida para la función agregada VARIANCE.

Esta función toma como argumento cualquier tipo de datos numérico o cualquier tipo de datos no numéricos que puedan convertirse implícitamente en un tipo de datos numérico. La función devuelve el mismo tipo de datos que el tipo de datos numérico del argumento.

Si especifica DISTINCT, entonces puede especificar solo la query_partition_clause de la analytic_clause. El order_by_clause y windowing_clause no están permitidos.

El siguiente ejemplo devuelve la desviación estándar de los salarios en la tabla de **empleados de** muestra:

Donde hr es Schema y empleados es un nombre de tabla.

```
SELECT STDDEV(salary) "Deviation"
FROM employees;

Deviation
----------
3909.36575
```

La consulta en el siguiente ejemplo devuelve la desviación estándar acumulada de los salarios en el Departamento 80 en la tabla de muestra hr.employees, ordenada por hire_date:

```
SELECT last_name, salary,
STDDEV(salary) OVER (ORDER BY hire_date) "StdDev"
FROM employees
WHERE department_id = 30;

LAST_NAME                     SALARY     StdDev
------------------------- ---------- ----------
Raphaely                       11000          0
Khoo                            3100 5586.14357
Tobias                          2800  4650.0896
```

Lea Funciones estadisticas en línea: https://riptutorial.com/es/oracle/topic/2283/funciones-estadisticas

Capítulo 17: Índices

Introducción

Aquí explicaré diferentes índices usando ejemplos, cómo el índice aumenta el rendimiento de las consultas, cómo el índice disminuye el rendimiento del DML, etc.

Examples

índice de árbol b

```
CREATE INDEX ord_customer_ix ON orders (customer_id);
```

Por defecto, si no mencionamos nada, Oracle crea un índice como un índice B-tree. Pero debemos saber cuándo usarlo. El índice de árbol B almacena los datos como formato de árbol binario. Como sabemos, el índice es un objeto de esquema que almacena algún tipo de entrada para cada valor para la columna indexada. Por lo tanto, siempre que se realice una búsqueda en esas columnas, verifica en el índice la ubicación exacta de ese registro para acceder rápidamente. Algunos puntos sobre la indexación:

- Para buscar una entrada en el índice, se utiliza algún tipo de algoritmo de búsqueda binario.
- Cuando **la cardinalidad de los datos es alta, el** índice **b-tree** es perfecto para usar.
- El índice hace que el DML sea lento, ya que para cada registro, debe haber una entrada en el índice para la columna indexada.
- Por lo tanto, si no es necesario, deberíamos evitar crear índices.

Índice de mapa de bits

```
CREATE BITMAP INDEX
emp_bitmap_idx
ON index_demo (gender);
```

- El índice de mapa de bits se utiliza cuando **la cardinalidad de los datos es baja.**
- Aquí, el **género** tiene valor con baja cardinalidad. Los valores pueden ser masculinos, femeninos y otros.
- Por lo tanto, si creamos un árbol binario para estos 3 valores mientras lo buscamos, tendremos un recorrido innecesario.
- En las estructuras de mapa de bits, se crea una matriz bidimensional con una columna para cada fila de la tabla que se está indexando. Cada columna representa un valor distinto dentro del índice de mapa de bits. Esta matriz bidimensional representa cada valor dentro del índice multiplicado por el número de filas en la tabla.
- En el momento de la recuperación de la fila, Oracle descomprime el mapa de bits en los buffers de datos de RAM para que pueda escanearse rápidamente en busca de valores coincidentes. Estos valores coincidentes se entregan a Oracle en forma de una lista de ID

de fila, y estos valores de ID de fila pueden acceder directamente a la información requerida.

Índice basado en funciones

```
CREATE INDEX first_name_idx ON user_data (UPPER(first_name));


SELECT *
FROM   user_data
WHERE  UPPER(first_name) = 'JOHN2';
```

- El índice basado en la función significa crear un índice basado en una función.
- Si en la búsqueda (donde la cláusula), con frecuencia se utiliza cualquier función, es mejor crear un índice basado en esa función.
- Aquí, en el ejemplo, para búsqueda, se está utilizando la función **Upper ()** . Por lo tanto, es mejor crear un índice utilizando la función superior.

Lea Índices en línea: https://riptutorial.com/es/oracle/topic/9978/indices

Capítulo 18: Limitar las filas devueltas por una consulta (Paginación)

Examples

Obtener las primeras N filas con la cláusula de limitación de fila

La cláusula `FETCH` se introdujo en Oracle 12c R1:

```
SELECT   val
FROM     mytable
ORDER BY val DESC
FETCH FIRST 5 ROWS ONLY;
```

Un ejemplo sin FETCH que funciona también en versiones anteriores:

```
SELECT * FROM (
   SELECT   val
   FROM     mytable
   ORDER BY val DESC
) WHERE ROWNUM <= 5;
```

Paginación en SQL

```
SELECT val
FROM   (SELECT val, rownum AS rnum
        FROM   (SELECT val
                FROM   rownum_order_test
                ORDER BY val)
        WHERE rownum <= :upper_limit)
WHERE  rnum >= :lower_limit ;
```

De esta manera podemos paginar los datos de la tabla, al igual que la página de servicio web

Obtener N números de registros de la tabla

Podemos limitar el número de filas del resultado utilizando la cláusula rownum

```
select * from
(
  select val from  mytable
) where rownum<=5
```

Si queremos el primer o último registro, queremos una cláusula orden por en la consulta interna que dará un resultado basado en el pedido.

Los últimos cinco discos:

```
select * from
(
    select val from  mytable order by val desc
) where rownum<=5
```

Primeros cinco discos

```
select * from
(
    select val from  mytable order by val
) where rownum<=5
```

Obtenga las filas N a M de muchas filas (antes de Oracle 12c)

Utilice la función analítica row_number ():

```
with t as (
  select col1
  , col2
  , row_number() over (order by col1, col2) rn
  from table
)
select col1
, col2
from t
where rn between N and M; -- N and M are both inclusive
```

Oracle 12c maneja esto más fácilmente con `OFFSET` y `FETCH` .

Saltando algunas filas y luego tomando algunas

En Oracle 12g +

```
SELECT Id, Col1
FROM TableName
ORDER BY Id
OFFSET 20 ROWS FETCH NEXT 20 ROWS ONLY;
```

En versiones anteriores

```
SELECT Id,
   Col1
 FROM (SELECT Id,
           Col1,
           row_number() over (order by Id) RowNumber
      FROM TableName)
WHERE RowNumber BETWEEN 21 AND 40
```

Saltando algunas filas del resultado

En Oracle 12g +

```
SELECT Id, Col1
FROM TableName
ORDER BY Id
OFFSET 5 ROWS;
```

En versiones anteriores

```
SELECT Id,
   Col1
 FROM (SELECT Id,
           Col1,
           row_number() over (order by Id) RowNumber
      FROM TableName)
WHERE RowNumber > 20
```

Lea Limitar las filas devueltas por una consulta (Paginación) en línea: https://riptutorial.com/es/oracle/topic/4300/limitar-las-filas-devueltas-por-una-consulta--paginacion-

Capítulo 19: Manejo de valores nulos

Introducción

Una columna es NULA cuando no tiene valor, independientemente del tipo de datos de esa columna. Una columna nunca debe compararse con NULL usando esta sintaxis `a = NULL` ya que el resultado sería DESCONOCIDO. En su lugar, utilice las condiciones `a IS NULL` o `a IS NOT NULL` . NULL no es igual a NULL. Para comparar dos expresiones donde puede ocurrir una nula, use una de las funciones que se describen a continuación. Todos los operadores, excepto la concatenación, devuelven NULL si uno de sus operandos es NULL. Por ejemplo, el resultado de `3 * NULL + 5` es nulo.

Observaciones

NULL no puede aparecer en columnas restringidas por una CLAVE PRINCIPAL o una restricción NOT NULL. (La excepción es una nueva restricción con la cláusula NOVALIDATE)

Examples

Las columnas de cualquier tipo de datos pueden contener NULLs

```
SELECT 1 NUM_COLUMN, 'foo' VARCHAR2_COLUMN from DUAL
UNION ALL
SELECT NULL, NULL from DUAL;
```

NUM_COLUMN	VARCHAR2_COLUMN
1	foo
(nulo)	(nulo)

Las cadenas vacías son NULL

```
SELECT 1 a, '' b from DUAL;
```

UNA	segundo
1	(nulo)

Las operaciones que contienen NULL son NULL, excepto la concatenación

```
SELECT 3 * NULL + 5, 'Hello ' || NULL || 'world'   from DUAL;
```

3 * NULL + 5	'HOLA' || NULL || 'MUNDO'
(nulo)	Hola Mundo

NVL para reemplazar el valor nulo

```
SELECT a column_with_null, NVL(a, 'N/A') column_without_null FROM
  (SELECT NULL a FROM DUAL);
```

COLUMN_WITH_NULL	COLUMN_WITHOUT_NULL
(nulo)	N / A

NVL es útil para comparar dos valores que pueden contener valores nulos:

```
SELECT
    CASE WHEN a = b THEN 1 WHEN a <> b THEN 0 else -1 END comparison_without_nvl,
    CASE WHEN NVL(a, -1) = NVL(b, -1) THEN 1 WHEN NVL(a, -1) <> NVL(b, -1) THEN 0 else -1 END
comparison_with_nvl
  FROM
    (select null a, 3 b FROM DUAL
     UNION ALL
     SELECT NULL, NULL FROM DUAL);
```

COMPARISON_WITHOUT_NVL	COMPARACIÓN_WITH_NVL
-1	0
-1	1

NVL2 para obtener un resultado diferente si un valor es nulo o no

Si el primer parámetro NO es NULO, NVL2 devolverá el segundo parámetro. De lo contrario volverá el tercero.

```
SELECT NVL2(null, 'Foo', 'Bar'), NVL2(5, 'Foo', 'Bar') FROM DUAL;
```

NVL2 (NULL, 'FOO', 'BAR')	NVL2 (5, 'FOO', 'BAR')
Bar	Foo

COALESCE para devolver el primer valor no NULL

```
SELECT COALESCE(a, b, c, d, 5) FROM
    (SELECT NULL A, NULL b, NULL c, 4 d FROM DUAL);
```

COALESCE (A, B, C, D, 5)
4

En algunos casos, usar COALESCE con dos parámetros puede ser más rápido que usar NVL cuando el segundo parámetro no es una constante. NVL siempre evaluará ambos parámetros. COALESCE se detendrá en el primer valor no NULL que encuentre. Esto significa que si el primer valor no es NULL, COALESCE será más rápido.

Lea Manejo de valores nulos en línea: https://riptutorial.com/es/oracle/topic/8183/manejo-de-valores-nulos

Capítulo 20: Manipulación de cuerdas

Examples

Concatenación: Operador || o función concat ()

El Oracle SQL y PL / SQL `||` el operador le permite concatenar 2 o más cadenas juntas.

Ejemplo:

Asumiendo la siguiente tabla de `customers` :

```
 id  firstname    lastname
---  -----------  ----------
  1  Thomas       Woody
```

Consulta:

```
 SELECT firstname || ' ' || lastname || ' is in my database.' as "My Sentence"
   FROM customers;
```

Salida:

```
My Sentence
---------------------------------
Thomas Woody is in my database.
```

Oracle también admite la función estándar de SQL `CONCAT(str1, str2)` :

Ejemplo:

Consulta:

```
 SELECT CONCAT(firstname, ' is in my database.') from customers;
```

Salida:

```
Expr1
---------------------------------
Thomas is in my database.
```

SUPERIOR

La función SUPERIOR le permite convertir todas las letras minúsculas de una cadena a mayúsculas.

```
SELECT UPPER('My text 123!') AS result FROM dual;
```

Salida:

```
RESULT
------------
MY TEXT 123!
```

INITCAP

La función `INITCAP` convierte el caso de una cadena de modo que cada palabra comience con una letra mayúscula y todas las letras subsiguientes estén en minúscula.

```
SELECT INITCAP('HELLO mr macdonald!') AS NEW FROM dual;
```

Salida

```
NEW
--------------------
Hello Mr Macdonald!
```

INFERIOR

LOWER convierte todas las letras mayúsculas de una cadena a minúsculas.

```
SELECT LOWER('HELLO World123!') text FROM dual;
```

Salidas:

texto
hola world123!

Expresión regular

Digamos que queremos reemplazar solo números con 2 dígitos: la expresión regular los encontrará con `(\d\d)`

```
SELECT REGEXP_REPLACE ('2, 5, and 10 are numbers in this example', '(\d\d)', '#')
FROM dual;
```

Resultados en:

```
'2, 5, and # are numbers in this example'
```

Si quiero intercambiar partes del texto, uso `\1` , `\2` , `\3` para llamar a las cadenas coincidentes:

```
SELECT REGEXP_REPLACE ('swap around 10 in that one ', '(.*)(\d\d )(.*)', '\3\2\1\3')
FROM dual;
```

SUBSTR

`SUBSTR` recupera parte de una cadena indicando la posición inicial y el número de caracteres que se extraerán

```
SELECT SUBSTR('abcdefg',2,3) FROM DUAL;
```

devoluciones:

```
bcd
```

Para contar desde el final de la cadena, `SUBSTR` acepta un número negativo como segundo parámetro, por ejemplo

```
SELECT SUBSTR('abcdefg',-4,2) FROM DUAL;
```

devoluciones:

```
de
```

Para obtener el último carácter de una cadena: `SUBSTR(mystring,-1,1)`

LTRIM / RTRIM

`LTRIM` y `RTRIM` eliminan los caracteres del principio o del final (respectivamente) de una cadena. Se puede suministrar un conjunto de uno o más caracteres (el valor predeterminado es un espacio) para eliminar.

Por ejemplo,

```
select LTRIM('<===>HELLO<===>', '=<>')
      ,RTRIM('<===>HELLO<===>', '=<>')
from dual;
```

Devoluciones:

```
HELLO<===>
<===>HELLO
```

Lea Manipulación de cuerdas en línea: https://riptutorial.com/es/oracle/topic/1518/manipulacion-de-cuerdas

Capítulo 21: Mesa doble

Observaciones

`DUAL` tabla `DUAL` tiene una columna `DUMMY` , definida como `VARCHAR2(1)` y solo una fila con un valor `x` .

`DUAL` tabla `DUAL` se crea automáticamente en el esquema `SYS` cuando se crea la base de datos. Puedes acceder desde cualquier esquema.

No se puede cambiar la tabla `DUAL` .

Puede usar la tabla `DUAL` para llamar a cualquier función desde la declaración SQL. Es útil porque solo tiene una fila y Oracle Optimizer lo sabe todo al respecto.

Examples

El siguiente ejemplo devuelve la fecha y hora actuales del sistema operativo

```
select sysdate from dual
```

El siguiente ejemplo genera números entre start_value y end_value

```
select :start_value + level -1 n
from dual
connect by level <= :end_value  - :start_value + 1
```

Lea Mesa doble en línea: https://riptutorial.com/es/oracle/topic/7328/mesa-doble

Capítulo 22: Oracle Advanced Queuing (AQ)

Observaciones

- Nunca use DDL o DML contra tablas creadas por `dbms_aqadm.create_queue_table` . Utilice solo dbms_aqadm y dbms_aq para trabajar con estas tablas. Oracle puede crear varias tablas de soporte, índices, etc. de los que no tendrá conocimiento. La ejecución manual de DDL o DML en la tabla puede llevarlo a un escenario en el que el soporte de Oracle necesitará que suelte y vuelva a crear la tabla y las colas para resolver la situación.
- Se recomienda encarecidamente que no utilice `dbms_aq.forever` para una opción de espera. Esto ha provocado problemas en el pasado, ya que Oracle puede comenzar a programar un número excesivo de trabajos de los trabajadores para que funcionen en las colas que no son necesarias (consulte Oracle Doc ID 2001165.1).
- Se recomienda que no configure el parámetro AQ_TM_PROCESSES en la versión 10.1 y posteriores. Evite especialmente establecer esto en cero ya que esto deshabilitará el trabajo de fondo QMON que es necesario para mantener las colas. Puede restablecer este valor al valor predeterminado de Oracle utilizando el siguiente comando y reiniciando la base de datos. `alter system reset aq_tm_processes scope=spfile sid='*';`

Examples

Productor / Consumidor Simple

Visión general

Crea una cola a la que podamos enviar un mensaje. Oracle notificará a nuestro procedimiento almacenado que un mensaje ha sido puesto en cola y debe ser trabajado. También agregaremos algunos subprogramas que podemos usar en una emergencia para evitar que los mensajes se desactualicen, permitir que se vuelva a poner en cola y ejecutar un trabajo por lotes simple para trabajar con todos los mensajes.

Estos ejemplos se probaron en Oracle Database 12c Enterprise Edition Release 12.1.0.2.0 - 64bit Production.

Crear cola

Crearemos un tipo de mensaje, una tabla de cola que puede contener los mensajes y una cola. Los mensajes en la cola se eliminarán en primer lugar por prioridad y luego su tiempo de salida. Si algo sale mal al trabajar el mensaje y la salida de la cola se retrotrae, AQ hará que el mensaje esté disponible para la salida 3600 segundos después. Lo hará 48 veces antes de moverlo a una cola de excepciones.

```
create type message_t as object
   (
   sender  varchar2 ( 50 ),
   message varchar2 ( 512 )
   );
/
-- Type MESSAGE_T compiled
begin dbms_aqadm.create_queue_table(
     queue_table        => 'MESSAGE_Q_TBL',
     queue_payload_type => 'MESSAGE_T',
     sort_list          => 'PRIORITY,ENQ_TIME',
     multiple_consumers =>  false,
     compatible         => '10.0.0');
  end;
/
-- PL/SQL procedure successfully completed.
begin dbms_aqadm.create_queue(
     queue_name          => 'MESSAGE_Q',
     queue_table         => 'MESSAGE_Q_TBL',
     queue_type          =>  0,
     max_retries         =>  48,
     retry_delay         =>  3600,
     dependency_tracking =>  false);
  end;
/
-- PL/SQL procedure successfully completed.
```

Ahora que tenemos un lugar para colocar los mensajes, podemos crear un paquete para administrar y trabajar los mensajes en la cola.

```
create or replace package message_worker_pkg
is
   queue_name_c constant varchar2(20) := 'MESSAGE_Q';

   -- allows the workers to process messages in the queue
   procedure enable_dequeue;

   -- prevents messages from being worked but will still allow them to be created and enqueued
   procedure disable_dequeue;

   -- called only by Oracle Advanced Queueing.  Do not call anywhere else.
   procedure on_message_enqueued (context        in raw,
                                  reginfo        in sys.aq$_reg_info,
                                  descr          in sys.aq$_descriptor,
                                  payload        in raw,
                                  payloadl       in number);

   -- allows messages to be worked if we missed the notification (or a retry
   -- is pending)
   procedure work_old_messages;

end;
/

create or replace package body message_worker_pkg
is
   -- raised by Oracle when we try to dequeue but no more messages are ready to
   -- be dequeued at this moment
   no_more_messages_ex          exception;
   pragma exception_init (no_more_messages_ex,
```

```
                        -25228);

   -- allows the workers to process messages in the queue
   procedure enable_dequeue
   as
   begin
      dbms_aqadm.start_queue (queue_name => queue_name_c, dequeue => true);
   end enable_dequeue;

   -- prevents messages from being worked but will still allow them to be created and enqueued
   procedure disable_dequeue
   as
   begin
      dbms_aqadm.stop_queue (queue_name => queue_name_c, dequeue => true, enqueue => false);
   end disable_dequeue;

   procedure work_message (message_in in out nocopy message_t)
   as
   begin
      dbms_output.put_line ( message_in.sender || ' says ' || message_in.message );
   end work_message;

   -- called only by Oracle Advanced Queueing.  Do not call anywhere else.

   procedure on_message_enqueued (context        in raw,
                                  reginfo        in sys.aq$_reg_info,
                                  descr          in sys.aq$_descriptor,
                                  payload        in raw,
                                  payloadl       in number)
   as
      pragma autonomous_transaction;
      dequeue_options_l      dbms_aq.dequeue_options_t;
      message_id_l           raw (16);
      message_l              message_t;
      message_properties_l   dbms_aq.message_properties_t;
   begin
      dequeue_options_l.msgid         := descr.msg_id;
      dequeue_options_l.consumer_name := descr.consumer_name;
      dequeue_options_l.wait          := dbms_aq.no_wait;
      dbms_aq.dequeue (queue_name           => descr.queue_name,
                       dequeue_options      => dequeue_options_l,
                       message_properties   => message_properties_l,
                       payload              => message_l,
                       msgid                => message_id_l);
      work_message (message_l);
      commit;
   exception
      when no_more_messages_ex
      then
         -- it's possible work_old_messages already dequeued the message
         commit;
      when others
      then
         -- we don't need to have a raise here.  I just wanted to point out that
         -- since this will be called by AQ throwing the exception back to it
         -- will have it put the message back on the queue and retry later
         raise;
   end on_message_enqueued;

   -- allows messages to be worked if we missed the notification (or a retry
   -- is pending)
```

```
   procedure work_old_messages
   as
      pragma autonomous_transaction;
      dequeue_options_l      dbms_aq.dequeue_options_t;
      message_id_l           raw (16);
      message_l              message_t;
      message_properties_l   dbms_aq.message_properties_t;
   begin
      dequeue_options_l.wait       := dbms_aq.no_wait;
      dequeue_options_l.navigation := dbms_aq.first_message;

      while (true) loop -- way out is no_more_messages_ex
         dbms_aq.dequeue (queue_name           => queue_name_c,
                          dequeue_options      => dequeue_options_l,
                          message_properties   => message_properties_l,
                          payload              => message_l,
                          msgid                => message_id_l);
         work_message (message_l);
         commit;
      end loop;
   exception
      when no_more_messages_ex
      then
         null;
   end work_old_messages;
end;
```

A continuación, informe a AQ que cuando se envía un mensaje a MESSAGE_Q (y se confirma), notifique a nuestro procedimiento que tiene trabajo que hacer. AQ iniciará un trabajo en su propia sesión para manejar esto.

```
begin
  dbms_aq.register (
     sys.aq$_reg_info_list (
        sys.aq$_reg_info (user || '.' || message_worker_pkg.queue_name_c,
                          dbms_aq.namespace_aq,
                          'plsql://' || user || '.message_worker_pkg.on_message_enqueued',
                          hextoraw ('FF'))),
     1);
  commit;
end;
```

Iniciar cola y enviar un mensaje

```
declare
   enqueue_options_l      dbms_aq.enqueue_options_t;
   message_properties_l   dbms_aq.message_properties_t;
   message_id_l           raw (16);
   message_l              message_t;
begin
   -- only need to do this next line ONCE
   dbms_aqadm.start_queue (queue_name => message_worker_pkg.queue_name_c, enqueue => true ,
dequeue => true);

   message_l := new message_t ( 'Jon', 'Hello, world!' );
   dbms_aq.enqueue (queue_name           => message_worker_pkg.queue_name_c,
                    enqueue_options      => enqueue_options_l,
```

```
                    message_properties   => message_properties_l,
                    payload              => message_l,
                    msgid                => message_id_l);
   commit;
end;
```

Lea Oracle Advanced Queuing (AQ) en línea: https://riptutorial.com/es/oracle/topic/4362/oracle-advanced-queuing--aq-

Capítulo 23: Oracle MAF

Examples

Para obtener valor de Binding

```
        ValueExpression ve = AdfmfJavaUtilities.getValueExpression(<binding>, String.class);
       String <variable_name> = (String) ve.getValue(AdfmfJavaUtilities.getELContext());
```

Aquí "vinculante" indica la expresión EL de la que se obtendrá el valor.

"nombre_variable" el parámetro en el que se almacena el valor del enlace

Para establecer el valor de enlace

```
        ValueExpression ve = AdfmfJavaUtilities.getValueExpression(<binding>, String.class);
        ve.setValue(AdfmfJavaUtilities.getELContext(), <value>);
```

Aquí "enlace" indica la expresión EL en la que se almacenará el valor.

"valor" es el valor que se desea agregar al enlace

Invocar un método desde el enlace.

```
    AdfELContext adfELContext = AdfmfJavaUtilities.getAdfELContext();
    MethodExpression me;
    me = AdfmfJavaUtilities.getMethodExpression(<binding>, Object.class, new Class[] { });
    me.invoke(adfELContext, new Object[] { });
```

"vinculante" indica la expresión EL desde la cual se invoca un método

Para llamar a una función javaScript

```
AdfmfContainerUtilities.invokeContainerJavaScriptFunction(AdfmfJavaUtilities.getFeatureId(),
<function>, new Object[] {
                                                                    });
```

"función" es la función js que se desea invocar

Lea Oracle MAF en línea: https://riptutorial.com/es/oracle/topic/6352/oracle-maf

Capítulo 24: Particionamiento de tablas

Introducción

La partición es una funcionalidad para dividir tablas e índices en partes más pequeñas. Se utiliza para mejorar el rendimiento y para gestionar las piezas más pequeñas de forma individual. La clave de partición es una columna o un conjunto de columnas que define en qué partición se almacenará cada fila. Resumen de particiones en la documentación oficial de Oracle

Observaciones

La partición es una opción de costo adicional y solo está disponible para la Edición Enterprise.

Examples

Particionamiento hash

Esto crea una tabla particionada por hash, en este ejemplo en el ID de tienda.

```
CREATE TABLE orders (
    order_nr NUMBER(15),
    user_id VARCHAR2(2),
    order_value NUMBER(15),
    store_id NUMBER(5)
)
PARTITION BY HASH(store_id) PARTITIONS 8;
```

Debería usar una potencia de 2 para el número de particiones hash, de modo que obtenga una distribución uniforme en el tamaño de la partición.

Partición de rango

Esto crea una tabla particionada por rangos, en este ejemplo en valores de orden.

```
CREATE TABLE orders (
    order_nr NUMBER(15),
    user_id VARCHAR2(2),
    order_value NUMBER(15),
    store_id NUMBER(5)
)
PARTITION BY RANGE(order_value) (
    PARTITION p1 VALUES LESS THAN(10),
    PARTITION p2 VALUES LESS THAN(40),
    PARTITION p3 VALUES LESS THAN(100),
    PARTITION p4 VALUES LESS THAN(MAXVALUE)
);
```

Seleccionar particiones existentes

Compruebe las particiones existentes en el esquema

```
SELECT * FROM user_tab_partitions;
```

Lista de particionamiento

Esto crea una tabla particionada por listas, en este ejemplo en el ID de tienda.

```
CREATE TABLE orders (
    order_nr NUMBER(15),
    user_id VARCHAR2(2),
    order_value NUMBER(15),
    store_id NUMBER(5)
)
PARTITION BY LIST(store_id) (
    PARTITION p1 VALUES (1,2,3),
    PARTITION p2 VALUES(4,5,6),
    PARTITION p3 VALUES(7,8,9),
    PARTITION p4 VALUES(10,11)
);
```

Soltar partición

```
ALTER TABLE table_name DROP PARTITION partition_name;
```

Seleccionar datos de una partición

Seleccionar datos de una partición

```
SELECT * FROM orders PARTITION(partition_name);
```

Truncar una partición

```
ALTER TABLE table_name TRUNCATE PARTITION partition_name;
```

Renombrar una partición

```
ALTER TABLE table_name RENAME PARTITION p3 TO p6;
```

Mueve la partición a un espacio de tabla diferente

```
ALTER TABLE table_name
MOVE PARTITION partition_name TABLESPACE tablespace_name;
```

Agregar nueva partición

```
ALTER TABLE table_name
```

```
ADD PARTITION new_partition VALUES LESS THAN(400);
```

Partición dividida

Divide algunas particiones en dos particiones con otro límite alto.

```
ALTER TABLE table_name SPLIT PARTITION old_partition
    AT (new_high_bound) INTO (PARTITION new_partition TABLESPACE new_tablespace,
    PARTITION old_partition)
```

Fusionar particiones

Combina dos particiones en una sola

```
ALTER TABLE table_name
  MERGE PARTITIONS first_partition, second_partition
  INTO  PARTITION  splitted_partition TABLESPACE new_tablespace
```

Intercambiar una partición

Intercambie / convierta una partición a una tabla no particionada y viceversa. Esto facilita un "movimiento" rápido de datos entre los segmentos de datos (en lugar de hacer algo como "insertar ... seleccionar" o "crear tabla ... como seleccionar") ya que la operación es DDL (la operación de intercambio de partición es un dato). actualización del diccionario sin mover los datos reales) y no DML (gran sobrecarga de deshacer / rehacer).

Ejemplos más básicos:

1. Convierta una tabla no particionada (tabla "B") a una partición (de la tabla "A"):

La tabla "A" no contiene datos en la partición "OLD_VALUES" y la tabla "B" contiene datos

```
ALTER TABLE "A" EXCHANGE PARTITION "OLD_VALUES" WITH TABLE "B";
```

Resultado: los datos se "mueven" de la tabla "B" (no contiene datos después de la operación) a la partición "OLD_VALUES"

2. Convertir una partición en una tabla no particionada:

La tabla "A" contiene datos en la partición "OLD_VALUES" y la tabla "B" no contiene datos

```
ALTER TABLE "A" EXCHANGE PARTITION "OLD_VALUES" WITH TABLE "B";
```

Resultado: los datos se "mueven" de la partición "OLD_VALUES" (no contiene datos después de la operación) a la tabla "B"

Nota: hay varias opciones, características y restricciones adicionales para esta operación.

Puede encontrar más información en este enlace ---> "
https://docs.oracle.com/cd/E11882_01/server.112/e25523/part_admin002.htm#i1107555 "
(sección "Intercambiar particiones")

Lea Particionamiento de tablas en línea:
https://riptutorial.com/es/oracle/topic/3955/particionamiento-de-tablas

Capítulo 25: Recuperación jerárquica con Oracle Database 12C

Introducción

Puede usar consultas jerárquicas para recuperar datos basados en una relación jerárquica natural entre filas en una tabla

Examples

Usando el CONNECT BY Caluse

```
SELECT E.EMPLOYEE_ID,E.LAST_NAME,E.MANAGER_ID FROM HR.EMPLOYEES E
CONNECT BY PRIOR E.EMPLOYEE_ID = E.MANAGER_ID;
```

La cláusula `CONNECT BY` para definir la relación entre empleados y gerentes.

Especificando la dirección de la consulta de arriba a abajo

```
SELECT E.LAST_NAME|| ' reports to ' ||
PRIOR E.LAST_NAME "Walk Top Down"
FROM HR.EMPLOYEES E
START WITH E.MANAGER_ID IS NULL
CONNECT BY PRIOR E.EMPLOYEE_ID = E.MANAGER_ID;
```

Lea Recuperación jerárquica con Oracle Database 12C en línea:
https://riptutorial.com/es/oracle/topic/8777/recuperacion-jerarquica-con-oracle-database-12c

Capítulo 26: Registro de errores

Examples

Registro de errores al escribir en la base de datos

Cree la tabla de registro de errores de Oracle ERR $ _EXAMPLE para la tabla EXAMPLE existente:

```
EXECUTE DBMS_ERRLOG.CREATE_ERROR_LOG('EXAMPLE', NULL, NULL, NULL, TRUE);
```

Hacer la operación de escritura con SQL:

```
insert into EXAMPLE (COL1) values ('example')
LOG ERRORS INTO ERR$_EXAMPLE reject limit unlimited;
```

Lea Registro de errores en línea: https://riptutorial.com/es/oracle/topic/3505/registro-de-errores

Capítulo 27: restricciones

Examples

Actualizar claves foráneas con nuevo valor en Oracle

Supongamos que tiene una tabla y desea cambiar una de las ID principales de esta tabla. Puedes usar el siguiente scrpit. ID primario aquí es "PK_S"

```
begin
  for i in (select a.table_name, c.column_name
              from user_constraints a, user_cons_columns c
             where a.CONSTRAINT_TYPE = 'R'
               and a.R_CONSTRAINT_NAME = 'PK_S'
               and c.constraint_name = a.constraint_name) loop


         execute immediate 'update ' || i.table_name || ' set ' || i.column_name ||
                     '=to_number(''1000'' || ' ||  i.column_name || ') ';



  end loop;

end;
```

Desactivar todas las claves externas relacionadas en Oracle

Supongamos que tiene la tabla T1 y tiene relación con muchas tablas y su nombre de restricción de clave principal es "pk_t1" que desea deshabilitar estas claves externas que puede usar:

```
Begin
    For I in (select table_name, constraint_name from user_constraint t where
r_constraint_name='pk_t1') loop

Execute immediate ' alter table ' || I.table_name || ' disable constraint ' ||
i.constraint_name;

   End loop;
End;
```

Lea restricciones en línea: https://riptutorial.com/es/oracle/topic/6040/restricciones

Capítulo 28: Se une

Examples

Unirse a la cruz

Una `CROSS JOIN` realiza una unión entre dos tablas que no usa una cláusula de unión explícita y da como resultado el producto cartesiano de dos tablas. Un producto cartesiano significa que cada fila de una tabla se combina con cada fila de la segunda tabla en la unión. Por ejemplo, si `TABLEA` tiene 20 filas y `TABLEB` tiene 20 filas, el resultado sería `20*20 = 400` filas de salida.

Ejemplo:

```
SELECT *
FROM TABLEA CROSS JOIN TABLEB;
```

Esto también se puede escribir como:

```
SELECT *
FROM TABLEA, TABLEB;
```

Aquí hay un ejemplo de unión cruzada en SQL entre dos tablas:

Tabla de muestra: TABLEA

```
+-------+---------+
| VALUE |   NAME  |
+-------+---------+
|   1   |   ONE   |
|   2   |   TWO   |
+-------+---------+
```

Tabla de muestra: TABLEB

```
+-------+--------+
| VALUE |  NAME  |
+-------+--------+
|   3   |  THREE |
|   4   |  FOUR  |
+-------+--------+
```

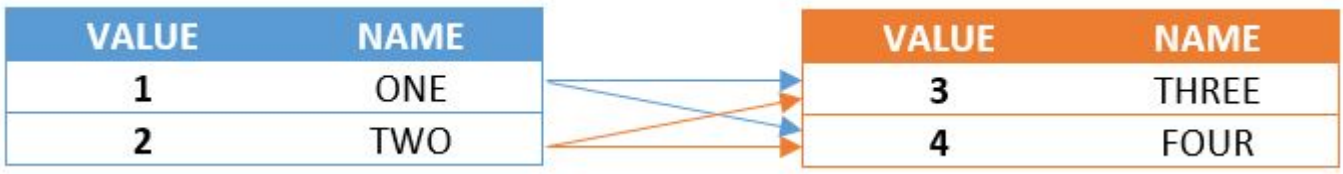
Ahora, si ejecuta la consulta:

```
SELECT *
FROM TABLEA CROSS JOIN TABLEB;
```

Salida:

```
+-------+--------+-------+--------+
| VALUE |  NAME  | VALUE |  NAME  |
+-------+--------+-------+--------+
|   1   |   ONE  |   3   |  THREE |
|   1   |   ONE  |   4   |  FOUR  |
|   2   |   TWO  |   3   |  THREE |
|   2   |   TWO  |   4   |  FOUR  |
+-------+--------+-------+--------+
```

Así es como ocurre la unión cruzada entre dos tablas:



Más sobre Cross Join: documentación de Oracle

UNIR INTERNAMENTE

Un INNER JOIN es una operación JOIN que le permite especificar una cláusula de unión explícita.

Sintaxis

TableExpression [INNER] JOIN TableExpression {ON booleanExpression | Cláusula USING}

Puede especificar la cláusula de unión especificando ON con una expresión booleana.

El alcance de las expresiones en la cláusula ON incluye las tablas actuales y cualquier tabla en bloques de consulta externos al SELECT actual. En el siguiente ejemplo, la cláusula ON se refiere a las tablas actuales:

```
-- Join the EMP_ACT and EMPLOYEE tables
-- select all the columns from the EMP_ACT table and
-- add the employee's surname (LASTNAME) from the EMPLOYEE table
-- to each row of the result
SELECT SAMP.EMP_ACT.*, LASTNAME
 FROM SAMP.EMP_ACT JOIN SAMP.EMPLOYEE
 ON EMP_ACT.EMPNO = EMPLOYEE.EMPNO
-- Join the EMPLOYEE and DEPARTMENT tables,
-- select the employee number (EMPNO),
-- employee surname (LASTNAME),
-- department number (WORKDEPT in the EMPLOYEE table and DEPTNO in the
-- DEPARTMENT table)
-- and department name (DEPTNAME)
-- of all employees who were born (BIRTHDATE) earlier than 1930.
SELECT EMPNO, LASTNAME, WORKDEPT, DEPTNAME
 FROM SAMP.EMPLOYEE JOIN SAMP.DEPARTMENT
 ON WORKDEPT = DEPTNO
 AND YEAR(BIRTHDATE) < 1930

-- Another example of "generating" new data values,
-- using a query which selects from a VALUES clause (which is an
```

```
-- alternate form of a fullselect).
-- This query shows how a table can be derived called "X"
-- having 2 columns "R1" and "R2" and 1 row of data
SELECT *
FROM (VALUES (3, 4), (1, 5), (2, 6))
AS VALUESTABLE1(C1, C2)
JOIN (VALUES (3, 2), (1, 2),
(0, 3)) AS VALUESTABLE2(c1, c2)
ON VALUESTABLE1.c1 = VALUESTABLE2.c1
-- This results in:
-- C1          |C2          |C1          |2
-- -----------------------------------------------
-- 3           |4           |3           |2
-- 1           |5           |1           |2


-- List every department with the employee number and
-- last name of the manager

SELECT DEPTNO, DEPTNAME, EMPNO, LASTNAME
FROM DEPARTMENT INNER JOIN EMPLOYEE
ON MGRNO = EMPNO

-- List every employee number and last name
-- with the employee number and last name of their manager
SELECT E.EMPNO, E.LASTNAME, M.EMPNO, M.LASTNAME
FROM EMPLOYEE E INNER JOIN
DEPARTMENT INNER JOIN EMPLOYEE M
    ON MGRNO = M.EMPNO
    ON E.WORKDEPT = DEPTNO
```

IZQUIERDA COMBINACIÓN EXTERNA

Una `LEFT OUTER JOIN` realiza una unión entre dos tablas que requiere una cláusula de unión explícita pero no excluye filas no coincidentes de la primera tabla.

Ejemplo:

```
SELECT
      ENAME,
      DNAME,
      EMP.DEPTNO,
      DEPT.DEPTNO
FROM
      SCOTT.EMP LEFT OUTER JOIN SCOTT.DEPT
      ON EMP.DEPTNO = DEPT.DEPTNO;
```

Aunque la sintaxis ANSI es la forma recomendada , es probable que se encuentre con una sintaxis heredada muy a menudo. El uso de `(+)` dentro de una condición determina qué lado de la ecuación debe considerarse como *externo* .

```
SELECT
      ENAME,
      DNAME,
      EMP.DEPTNO,
      DEPT.DEPTNO
```

```
FROM
      SCOTT.EMP,
      SCOTT.DEPT
WHERE
      EMP.DEPTNO = DEPT.DEPTNO(+);
```

Aquí hay un ejemplo de la combinación externa izquierda entre dos tablas:

Tabla de muestra: EMPLEADO

```
+-----------+---------+
|    NAME   |  DEPTNO |
+-----------+---------+
|     A     |    2    |
|     B     |    1    |
|     C     |    3    |
|     D     |    2    |
|     E     |    1    |
|     F     |    1    |
|     G     |    4    |
|     H     |    4    |
+-----------+---------+
```

Tabla de muestra: DEPTO

```
+---------+--------------+
|  DEPTNO |   DEPTNAME   |
+---------+--------------+
|    1    |  ACCOUNTING  |
|    2    |    FINANCE   |
|    5    |   MARKETING  |
|    6    |      HR      |
+---------+--------------+
```

Ahora, si ejecuta la consulta:

```
SELECT
      *
FROM
      EMPLOYEE LEFT OUTER JOIN DEPT
      ON EMPLOYEE.DEPTNO = DEPT.DEPTNO;
```

Salida:

```
+-----------+---------+---------+--------------+
|    NAME   |  DEPTNO |  DEPTNO |   DEPTNAME   |
+-----------+---------+---------+--------------+
|     F     |    1    |    1    |  ACCOUNTING  |
|     E     |    1    |    1    |  ACCOUNTING  |
|     B     |    1    |    1    |  ACCOUNTING  |
|     D     |    2    |    2    |    FINANCE   |
|     A     |    2    |    2    |    FINANCE   |
|     C     |    3    |         |              |
|     H     |    4    |         |              |
|     G     |    4    |         |              |
+-----------+---------+---------+--------------+
```

JUSTE EXTERIOR DERECHO

A `RIGHT OUTER JOIN` realiza una unión entre dos tablas que requiere una cláusula de unión explícita pero no excluye filas no coincidentes de la segunda tabla.

Ejemplo:

```
SELECT
      ENAME,
      DNAME,
      EMP.DEPTNO,
      DEPT.DEPTNO
FROM
      SCOTT.EMP RIGHT OUTER JOIN SCOTT.DEPT
      ON EMP.DEPTNO = DEPT.DEPTNO;
```

Como se incluyen las filas no `SCOTT.DEPT` de `SCOTT.DEPT` , pero no se incluyen las filas no `SCOTT.EMP` de `SCOTT.EMP` , lo anterior es equivalente a la siguiente declaración que usa `LEFT OUTER JOIN` .

```
SELECT
      ENAME,
      DNAME,
      EMP.DEPTNO,
      DEPT.DEPTNO
FROM
      SCOTT.DEPT RIGHT OUTER JOIN SCOTT.EMP
      ON DEPT.DEPTNO = EMP.DEPTNO;
```

Aquí hay un ejemplo de unión externa derecha entre dos tablas:

Tabla de muestra: EMPLEADO

```
+-----------+---------+
|    NAME   |  DEPTNO |
+-----------+---------+
|     A     |    2    |
|     B     |    1    |
|     C     |    3    |
|     D     |    2    |
|     E     |    1    |
|     F     |    1    |
|     G     |    4    |
|     H     |    4    |
+-----------+---------+
```

Tabla de muestra: DEPTO

```
+---------+--------------+
|  DEPTNO |   DEPTNAME   |
+---------+--------------+
|    1    |  ACCOUNTING  |
|    2    |    FINANCE   |
|    5    |   MARKETING  |
|    6    |      HR      |
```

```
+---------+--------------+
```

Ahora, si ejecuta la consulta:

```
SELECT
      *
FROM
      EMPLOYEE RIGHT OUTER JOIN DEPT
      ON EMPLOYEE.DEPTNO = DEPT.DEPTNO;
```

Salida:

```
+-----------+---------+---------+--------------+
|    NAME   |  DEPTNO |  DEPTNO |   DEPTNAME   |
+-----------+---------+---------+--------------+
|     A     |    2    |    2    |    FINANCE   |
|     B     |    1    |    1    |  ACCOUNTING  |
|     D     |    2    |    2    |    FINANCE   |
|     E     |    1    |    1    |  ACCOUNTING  |
|     F     |    1    |    1    |  ACCOUNTING  |
|           |         |    5    |   MARKETING  |
|           |         |    6    |      HR      |
+-----------+---------+---------+--------------+
```

La sintaxis de Oracle (+) equivalente para la consulta es:

```
SELECT *
FROM EMPLOYEE, DEPT
WHERE EMPLOYEE.DEPTNO(+) = DEPT.DEPTNO;
```

ÚNICAMENTE EN EL EXTERIOR

Un `FULL OUTER JOIN` realiza una unión entre dos tablas que requiere una cláusula de unión explícita pero no excluye filas no coincidentes en ninguna de las tablas. En otras palabras, devuelve todas las filas en cada tabla.

Ejemplo:

```
SELECT
      *
FROM
      EMPLOYEE FULL OUTER JOIN DEPT
      ON EMPLOYEE.DEPTNO = DEPT.DEPTNO;
```

Aquí hay un ejemplo de Full Outer Join entre dos tablas:

Tabla de muestra: EMPLEADO

```
+-----------+---------+
|    NAME   |  DEPTNO |
+-----------+---------+
|     A     |    2    |
```

```
|     B     |    1    |
|     C     |    3    |
|     D     |    2    |
|     E     |    1    |
|     F     |    1    |
|     G     |    4    |
|     H     |    4    |
+-----------+---------+
```

Tabla de muestra: DEPTO

```
+---------+--------------+
|  DEPTNO |   DEPTNAME   |
+---------+--------------+
|    1    |  ACCOUNTING  |
|    2    |    FINANCE   |
|    5    |   MARKETING  |
|    6    |      HR      |
+---------+--------------+
```

Ahora, si ejecuta la consulta:

```
SELECT
      *
FROM
      EMPLOYEE FULL OUTER JOIN DEPT
      ON EMPLOYEE.DEPTNO = DEPT.DEPTNO;
```

Salida

```
+-----------+---------+---------+--------------+
|    NAME   |  DEPTNO |  DEPTNO |   DEPTNAME   |
+-----------+---------+---------+--------------+
|     A     |    2    |    2    |    FINANCE   |
|     B     |    1    |    1    |  ACCOUNTING  |
|     C     |    3    |         |              |
|     D     |    2    |    2    |    FINANCE   |
|     E     |    1    |    1    |  ACCOUNTING  |
|     F     |    1    |    1    |  ACCOUNTING  |
|     G     |    4    |         |              |
|     H     |    4    |         |              |
|           |         |    6    |      HR      |
|           |         |    5    |   MARKETING  |
+-----------+---------+---------+--------------+
```

Aquí las columnas que no coinciden se han mantenido en NULL.

Aniquilar

Un antijoin devuelve filas desde el lado izquierdo del predicado para el que no hay filas correspondientes en el lado derecho del predicado. Devuelve las filas que no logran coincidir (NO EN) con la subconsulta en el lado derecho.

```
SELECT * FROM employees
```

```
   WHERE department_id NOT IN
   (SELECT department_id FROM departments
       WHERE location_id = 1700)
   ORDER BY last_name;
```

Aquí hay un ejemplo de Anti Join entre dos tablas:

Tabla de muestra: EMPLEADO

```
+-----------+---------+
|    NAME   |  DEPTNO |
+-----------+---------+
|     A     |    2    |
|     B     |    1    |
|     C     |    3    |
|     D     |    2    |
|     E     |    1    |
|     F     |    1    |
|     G     |    4    |
|     H     |    4    |
+-----------+---------+
```

Tabla de muestra: DEPTO

```
+---------+--------------+
|  DEPTNO |   DEPTNAME   |
+---------+--------------+
|    1    |  ACCOUNTING  |
|    2    |    FINANCE   |
|    5    |   MARKETING  |
|    6    |      HR      |
+---------+--------------+
```

Ahora, si ejecuta la consulta:

```
SELECT
      *
FROM
      EMPLOYEE WHERE DEPTNO NOT IN
      (SELECT DEPTNO FROM DEPT);
```

Salida:

```
+-----------+---------+
|    NAME   |  DEPTNO |
+-----------+---------+
|     C     |    3    |
|     H     |    4    |
|     G     |    4    |
+-----------+---------+
```

La salida muestra que solo las filas de la tabla EMPLOYEE, de las cuales DEPTNO no estaban presentes en la tabla DEPT.

SEMIJOIN

Se puede usar una consulta de semijoin, por ejemplo, para encontrar todos los departamentos con al menos un empleado cuyo salario exceda de 2500.

```
SELECT * FROM departments
   WHERE EXISTS
   (SELECT 1 FROM employees
       WHERE departments.department_id = employees.department_id
       AND employees.salary > 2500)
   ORDER BY department_name;
```

Esto es más eficiente que las alternativas de unión completa, ya que la unión interna de los empleados y luego una cláusula donde se detalla que el salario debe ser superior a 2500 podría devolver el mismo departamento varias veces. Diga si el departamento de Bomberos tiene `n` empleados, todos con el salario 3000, `select * from departments, employees` con las inscripciones necesarias en las identificaciones y nuestra cláusula de dónde devolvería el departamento de Bomberos `n` veces.

UNIRSE

La operación `JOIN` realiza una unión entre dos tablas, excluyendo cualquier fila no coincidente de la primera tabla. Desde Oracle 9i adelante, la función `JOIN` es equivalente en función a la `INNER JOIN` . Esta operación requiere una cláusula de unión explícita, a diferencia de los operadores `CROSS JOIN` y `NATURAL JOIN` .

Ejemplo:

```
select t1.*,
       t2.DeptId
  from table_1 t1
  join table_2 t2 on t2.DeptNo = t1.DeptNo
```

Documentación de Oracle:

- 10g
- 11g
- 12g

Unirse natural

La unión natural no requiere una condición de unión explítica; construye uno basado en todos los campos con el mismo nombre en las tablas unidas.

```
create table tab1(id number,  descr varchar2(100));
create table tab2(id number,  descr varchar2(100));
insert into tab1 values(1, 'one');
insert into tab1 values(2, 'two');
insert into tab1 values(3, 'three');
insert into tab2 values(1, 'ONE');
```

```
insert into tab2 values(3, 'three');
```

La unión se realizará en los campos ID y DESCR, comunes a ambas tablas:

```
SQL> select *
  2  from tab1
  3       natural join
  4      tab2;

        ID DESCR
---------- ----------
         3 three
```

Las columnas con nombres diferentes no se utilizarán en la condición de ÚNETE:

```
SQL> select *
  2  from (select id as id, descr as descr1 from tab1)
  3       natural join
  4       (select id as id, descr as descr2 from tab2);

        ID DESCR1     DESCR2
---------- ---------- ----------
         1 one        ONE
         3 three      three
```

Si las tablas unidas no tienen columnas comunes, se realizará una ÚNETE sin condiciones:

```
SQL> select *
  2  from (select id as id1, descr as descr1 from tab1)
  3       natural join
  4       (select id as id2, descr as descr2 from tab2);

       ID1 DESCR1            ID2 DESCR2
---------- ---------- ---------- ----------
         1 one                 1 ONE
         2 two                 1 ONE
         3 three               1 ONE
         1 one                 3 three
         2 two                 3 three
         3 three               3 three
```

Lea Se une en línea: https://riptutorial.com/es/oracle/topic/4192/se-une

Capítulo 29: Secuencias

Sintaxis

- CREAR SECUENCIA SCHEMA.SEQUENCE {INCREMENTO DE INTEGER | COMENZAR CON INTEGER | MAXVALUE INTEGER | INTEGER NOMAXVALUE | MINVALUE INTEGER | NOMINVALUE INTEGER | CICLO INTEGER | INTEGER DE CICLO | CACHE | NOCACHE | ORDEN | NOODER}

Parámetros

Parámetro	Detalles
esquema	nombre de esquema
incrementar por	intervalo entre los números
Empezar con	primer numero necesario
valor máximo	Valor máximo para la secuencia.
nomaxvalue	El valor máximo está predeterminado
minvalor	valor mínimo para la secuencia
valor nominal	el valor mínimo está predeterminado
ciclo	Restablecer al inicio después de alcanzar este valor.
nociciclo	Defecto
cache	Límite de Preubicación
nocache	Defecto
orden	Garantizar el orden de los números.
sin orden	defecto

Examples

Creando una secuencia: Ejemplo

Propósito

Utilice la instrucción CREATE SEQUENCE para crear una secuencia, que es un objeto de base

de datos desde el cual varios usuarios pueden generar enteros únicos. Puedes usar secuencias para generar automáticamente valores de clave primaria.

Cuando se genera un número de secuencia, la secuencia se incrementa, independientemente de la transacción confirmada o retrotraída. Si dos usuarios incrementan simultáneamente la misma secuencia, entonces los números de secuencia que cada usuario adquiere pueden tener huecos, ya que el otro usuario está generando números de secuencia. Un usuario nunca puede adquirir el número de secuencia generado por otro usuario. Después de que un usuario genere un valor de secuencia, ese usuario puede continuar accediendo a ese valor independientemente de si la secuencia es incrementada por otro usuario.

Los números de secuencia se generan independientemente de las tablas, por lo que la misma secuencia se puede usar para una o varias tablas. Es posible que los números de secuencia individuales parezcan omitidos, ya que se generaron y usaron en una transacción que finalmente se retrotrajo. Además, un solo usuario puede no darse cuenta de que otros usuarios están dibujando desde la misma secuencia.

Después de crear una secuencia, puede acceder a sus valores en sentencias de SQL con la pseudocolumna CURRVAL, que devuelve el valor actual de la secuencia, o la pseudocolumna NEXTVAL, que incrementa la secuencia y devuelve el nuevo valor.

Prerrequisitos

Para crear una secuencia en su propio esquema, debe tener el privilegio del sistema CREATE SEQUENCE.

Para crear una secuencia en el esquema de otro usuario, debe tener el privilegio del sistema CREAR CUALQUIER SECUENCIA.

Creación de una secuencia: Ejemplo La siguiente declaración crea la secuencia clients_seq en el esquema de muestra oe. Esta secuencia podría usarse para proporcionar números de ID de cliente cuando se agregan filas a la tabla de clientes.

```
CREATE SEQUENCE customers_seq
START WITH     1000
INCREMENT BY   1
NOCACHE
NOCYCLE;
```

La primera referencia a customers_seq.nextval devuelve 1000. La segunda devuelve 1001. Cada referencia posterior devolverá un valor 1 mayor que la referencia anterior.

Lea Secuencias en línea: https://riptutorial.com/es/oracle/topic/3709/secuencias

Capítulo 30: Seguridad de aplicación real

Introducción

Oracle Real Application Security se introdujo en Oracle 12c. Resume muchos temas de seguridad, como el modelo de rol del usuario, el control de acceso, la aplicación frente a la base de datos, la seguridad del usuario final o la seguridad a nivel de fila y columna.

Examples

Solicitud

Para asociar una aplicación con algo en la base de datos hay tres partes principales:

Privilegio de la aplicación: un privilegio de la aplicación describe privilegios como `SELECT` , `INSERT` , `UPDATE` , `DELETE` , ... Los privilegios de la aplicación se pueden resumir como un privilegio agregado.

```
XS$PRIVILEGE(
    name=>'privilege_name'
    [, implied_priv_list=>XS$NAME_LIST('"SELECT"', '"INSERT"', '"UPDATE"', '"DELETE"')]
)

XS$PRIVILEGE_LIST(
    XS$PRIVILEGE(...),
    XS$PRIVILEGE(...),
    ...
);
```

Usuario de la aplicación:

Usuario de aplicación simple:

```
BEGIN
    SYS.XS_PRINCIPAL.CREATE_USER('user_name');
END;
```

Usuario de la aplicación de inicio de sesión directo:

```
BEGIN
    SYS.XS_PRINCIPAL.CREATE_USER(name => 'user_name', schema => 'schema_name');
END;

BEGIN
    SYS.XS_PRINCIPAL.SET_PASSWORD('user_name', 'password');
END;
CREATE PROFILE prof LIMIT
    PASSWORD_REUSE_TIME 1/4440
    PASSWORD_REUSE_MAX 3
    PASSWORD_VERIFY_FUNCTION Verify_Pass;
```

```
BEGIN
    SYS.XS_PRINCIPAL.SET_PROFILE('user_name', 'prof');
END;

BEGIN
    SYS.XS_PRINCIPAL.GRANT_ROLES('user_name', 'XSONNCENT');
END;
```

(Opcional:)

```
BEGIN
    SYS.XS_PRINCIPAL.SET_VERIFIER('user_name', '6DFF060084ECE67F', XS_PRINCIPAL.XS_SHA512");
END;
```

Papel de la aplicación:

Rol de aplicación regular:

```
DECLARE
    st_date TIMESTAMP WITH TIME ZONE;
    ed_date TIMESTAMP WITH TIME ZONE;
BEGIN
    st_date := SYSTIMESTAMP;
    ed_date := TO_TIMESTAMP_TZ('2013-06-18 11:00:00 -5:00','YYYY-MM-DD HH:MI:SS');
    SYS.XS_PRINCIPAL.CREATE_ROLE
        (name => 'app_regular_role',
        enabled => TRUE,
        start_date => st_date,
        end_date => ed_date);
END;
```

Rol de aplicación dinámica: (se habilita dinámicamente en función del estado de autenticación)

```
BEGIN
    SYS.XS_PRINCIPAL.CREATE_DYNAMIC_ROLE
        (name => 'app_dynamic_role',
        duration => 40,
        scope => XS_PRINCIPAL.SESSION_SCOPE);
END;
```

Roles de aplicación predefinidos:

Regular:

- XSPUBLIC
- XSBYPASS
- XSSESSIONADMIN
- XSNAMESPACEADMIN
- XSPROVISIONER
- XSCACHEADMIN
- XSDISPATCHER

Dinámico: (depende del estado de autenticación del usuario de la aplicación)

- `DBMS_AUTH` : (inicio de sesión directo u otro método de autenticación de base de datos)
- `EXTERNAL_DBMS_AUTH` : (inicio de sesión directo u otro método de autenticación de base de datos y el usuario es externo)
- `DBMS_PASSWD` : (inicio de sesión directo con contraseña)
- `MIDTIER_AUTH` : (autenticación a través de la aplicación de nivel medio)
- `XSAUTHENTICATED` : (aplicación de nivel directo o medio)
- `XSSWITCH` : (usuario cambiado de usuario proxy a usuario de aplicación)

Lea Seguridad de aplicación real en línea: https://riptutorial.com/es/oracle/topic/10864/seguridad-de-aplicacion-real

Capítulo 31: SQL dinámico

Introducción

SQL dinámico le permite ensamblar un código de consulta SQL en el tiempo de ejecución. Esta técnica tiene algunas desventajas y debe ser usada muy cuidadosamente. Al mismo tiempo, le permite implementar una lógica más compleja. PL / SQL requiere que todos los objetos, utilizados en el código, tengan que existir y ser válidos en el momento de la compilación. Es por eso que no puede ejecutar sentencias DDL en PL / SQL directamente, pero SQL dinámico le permite hacer eso.

Observaciones

Algunas observaciones importantes:

1. Nunca use la concatenación de cadenas para agregar valores a la consulta, use parámetros en su lugar. Esto está mal:

   ```
   execute immediate 'select value from my_table where id = ' ||
        id_valiable into result_variable;
   ```

   Y esto es correcto:

   ```
   execute immediate 'select value from my_table where id = :P '
       using id_valiable into result_variable;
   ```

   Hay dos razones para esto. El primero es la seguridad. La concatenación de cadenas permite realizar inyecciones SQL. En la consulta a continuación, si una variable contendrá el valor `1 or 1 = 1` , la instrucción `UPDATE` actualizará todas las líneas de la tabla:

   ```
   execute immediate 'update my_table set value = ''I have bad news for you'' where id = '
   || id;
   ```

   La segunda razón es el rendimiento. Oracle analizará la consulta sin parámetros cada vez que se ejecute, mientras que la consulta con el parámetro se analizará solo una vez en la sesión.

2. Tenga en cuenta que cuando el motor de base de datos ejecuta una sentencia DDL, ejecuta un compromiso implícito antes.

Examples

Seleccionar valor con SQL dinámico

Digamos que un usuario quiere seleccionar datos de diferentes tablas. Una tabla es especificada

por el usuario.

```
 function get_value(p_table_name varchar2, p_id number) return varchar2 is
    value varchar2(100);
  begin
    execute immediate 'select column_value from ' || p_table_name ||
                      ' where id = :P' into value using p_id;
    return value;
  end;
```

Llame a esta función como de costumbre:

```
declare
  table_name varchar2(30) := 'my_table';
  id number := 1;
begin
  dbms_output.put_line(get_value(table_name, id));
end;
```

Tabla a probar:

```
create table my_table (id number, column_value varchar2(100));
insert into my_table values (1, 'Hello, world!');
```

Insertar valores en SQL dinámico

El siguiente ejemplo inserta valor en la tabla del ejemplo anterior:

```
declare
  query_text varchar2(1000) := 'insert into my_table(id, column_value) values (:P_ID,
:P_VAL)';
  id number := 2;
  value varchar2(100) := 'Bonjour!';
begin
  execute immediate query_text using id, value;
end;
/
```

Actualizar valores en SQL dinámico

Vamos a actualizar la tabla del primer ejemplo:

```
declare
  query_text varchar2(1000) := 'update my_table set column_value = :P_VAL where id = :P_ID';
  id number := 2;
  value varchar2(100) := 'Bonjour le monde!';
begin
  execute immediate query_text using value, id;
end;
/
```

Ejecutar sentencia DDL

Este código crea la tabla:

```
begin
  execute immediate 'create table my_table (id number, column_value varchar2(100))';
end;
/
```

Ejecutar bloque anonimo

Puede ejecutar bloque anónimo. Este ejemplo muestra también cómo devolver valor desde SQL dinámico:

```
declare
  query_text varchar2(1000) := 'begin :P_OUT := cos(:P_IN); end;';
  in_value number := 0;
  out_value number;
begin
  execute immediate query_text using out out_value, in in_value;
  dbms_output.put_line('Result of anonymous block: ' || to_char(out_value));
end;
/
```

Lea SQL dinámico en línea: https://riptutorial.com/es/oracle/topic/10905/sql-dinamico

Capítulo 32: Trabajando con fechas

Examples

Fecha aritmética

Oracle admite los tipos de datos `DATE` (incluye el tiempo al segundo más cercano) y `TIMESTAMP` (incluye el tiempo a fracciones de un segundo), lo que permite la aritmética (suma y resta) de forma nativa. Por ejemplo:

Para llegar al día siguiente:

```
select to_char(sysdate + 1, 'YYYY-MM-DD') as tomorrow from dual;
```

Para obtener el día anterior:

```
select to_char(sysdate - 1, 'YYYY-MM-DD') as yesterday from dual;
```

Para agregar 5 días a la fecha actual:

```
select to_char(sysdate + 5, 'YYYY-MM-DD') as five_days_from_now from dual;
```

Para agregar 5 horas a la fecha actual:

```
select to_char(sysdate + (5/24), 'YYYY-MM-DD HH24:MI:SS') as five_hours_from_now from dual;
```

Para agregar 10 minutos a la fecha actual:

```
select to_char(sysdate + (10/1440), 'YYYY-MM-DD HH24:MI:SS') as ten_mintues_from_now from
dual;
```

Para agregar 7 segundos a la fecha actual:

```
select to_char(sysdate + (7/86400), 'YYYY-MM-DD HH24:MI:SS') as seven_seconds_from_now from
dual;
```

Para seleccionar filas en las que `hire_date` sea hace 30 días o más:

```
select * from emp where hire_date < sysdate - 30;
```

Para seleccionar filas donde la columna `last_updated` está en la última hora:

```
select * from logfile where last_updated >= sysdate - (1/24);
```

Oracle también proporciona el `INTERVAL` tipo de datos incorporado, que representa una duración de

tiempo (por ejemplo, 1,5 días, 36 horas, 2 meses, etc.). También se pueden usar con aritmética con expresiones `DATE` y `TIMESTAMP` . Por ejemplo:

```
select * from logfile where last_updated >= sysdate - interval '1' hour;
```

Función Add_months

Sintaxis: `add_months(p_date, integer) return date;`

La función Add_months agrega meses amt a la fecha p_date.

```
SELECT add_months(date'2015-01-12', 2) m FROM dual;
```

METRO
2015-03-12

También puedes restar meses usando un `amt` negativo.

```
SELECT add_months(date'2015-01-12', -2) m FROM dual;
```

METRO
2014-11-12

Cuando el mes calculado tenga menos días como la fecha dada, se devolverá el último día del mes calculado.

```
SELECT to_char( add_months(date'2015-01-31', 1),'YYYY-MM-DD') m FROM dual;
```

METRO
2015-02-28

Lea Trabajando con fechas en línea: https://riptutorial.com/es/oracle/topic/768/trabajando-con-fechas

Capítulo 33: Transacciones Autónomas

Observaciones

Los casos típicos de uso para transacciones autónomas son.

1. Para crear cualquier tipo de marco de registro como el marco de registro de errores explicado en el ejemplo anterior.
2. Para auditar operaciones DML en disparadores en tablas, independientemente del estado final de la transacción (COMPROMISO o ROLLBACK).

Examples

Uso de transacciones autónomas para errores de registro

El siguiente procedimiento es genérico y se utilizará para registrar todos los errores en una aplicación en una tabla de registro de errores común.

```
CREATE OR REPLACE PROCEDURE log_errors
(
  p_calling_program IN VARCHAR2,
  p_error_code IN INTEGER,
  p_error_description IN VARCHAR2
)
IS
  PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
  INSERT INTO error_log
  VALUES
  (
  p_calling_program,
  p_error_code,
  p_error_description,
  SYSDATE,
  USER
  );
  COMMIT;
END log_errors;
```

El siguiente bloque PLSQL anónimo muestra cómo llamar al procedimiento log_errors.

```
BEGIN
   DELETE FROM dept WHERE deptno = 10;
EXCEPTION
   WHEN OTHERS THEN
      log_errors('Delete dept',sqlcode, sqlerrm);
      RAISE;
END;

SELECT * FROM error_log;
```

```
CALLING_PROGRAM    ERROR_CODE    ERROR_DESCRIPTION
ERROR_DATETIME         DB_USER
Delete dept        -2292         ORA-02292: integrity constraint violated - child record found
08/09/2016             APEX_PUBLIC_USER
```

Lea Transacciones Autónomas en línea: https://riptutorial.com/es/oracle/topic/6103/transacciones-autonomas

Creditos

S. No	Capítulos	Contributors
1	Comenzando con la base de datos Oracle	Community, J. Chomel, Jeffrey Kemp, Jon Ericson, Kevin Montrose, Mark Stewart, Sanjay Radadiya, Steven Feuerstein, tonirush
2	Actualizar con uniones	mathguy
3	Bloque PL / SQL anónimo	Jon Heller, Skynet, Zohar Elkayam
4	Bomba de datos	Vidya Thotangare
5	Consejos	Aleksej, Florin Ghita, Jon Heller, Mark Stewart, Pirate X
6	consulta de nivel	Sanjay Radadiya, TechEnthusiast
7	Creando un contexto	Jeffrey Kemp
8	Delimitando palabras clave o caracteres especiales	dev
9	Diccionario de datos	Mark Stewart, Pancho, Slava Babin
10	Diferentes formas de actualizar registros.	nimour pristou, Nogueira Jr, SriniV
11	División de cadenas delimitadas	Arkadiusz Łukasiewicz, MT0
12	Enlaces de base de datos	carlosb, Daniel Langemann, g00dy, kasi
13	Factoraje de subconsultas recursivas utilizando la cláusula WITH (expresiones de tabla comunes AKA)	B Samedi, MT0
14	fechas	carlosb, MT0, Roman, tonirush
15	Funciones de	Leigh Riffel

	ventana	
16	Funciones estadisticas	Evgeniy K., Matas Vaitkevicius, ppeterka, Pranav Shah
17	Índices	smshafiqulislam
18	Limitar las filas devueltas por una consulta (Paginación)	Ahmed Mohamed, Martin Schapendonk, Matas Vaitkevicius, Sanjay Radadiya, tonirush, trincot
19	Manejo de valores nulos	Dalex, JeromeFr
20	Manipulación de cuerdas	carlosb, Eric B., Florin Ghita, Francesco Serra, J. Chomel, J.Hudler, Jeffrey Kemp, Mark Stewart, SriniV, Thunder, walen, zhliu03
21	Mesa doble	Slava Babin
22	Oracle Advanced Queuing (AQ)	Jon Theriault
23	Oracle MAF	Anand Raj
24	Particionamiento de tablas	BobC, carlosb, ivanzg, JeromeFr, Kamil Islamov, Stephen Leppik, tonirush
25	Recuperación jerárquica con Oracle Database 12C	Muntasir, Vahid
26	Registro de errores	zygimantus
27	restricciones	SSD
28	Se une	Aleksej, B Samedi, Bakhtiar Hasan, Daniel Langemann, Erkan Haspulat, Pranav Shah, Robin James, SriniV, Sumner Evans
29	Secuencias	Pranav Shah, SriniV
30	Seguridad de aplicación real	Ben H
31	SQL dinámico	Dmitry
32	Trabajando con fechas	David Aldridge, Florin Ghita, Jeffrey Kemp, Mark Stewart, tonirush, zygimantus

33	Transacciones Autónomas	phonetic_man