EBook Gratuito

APPRENDIMENTO
Oracle Database

Free unaffiliated eBook created from
Stack Overflow contributors.

#oracle

Sommario

Di

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

Capitolo 1: Iniziare con il database Oracle

Osservazioni

Oracle è un sistema di gestione di database relazionali (RDBMS) originariamente costruito da Larry Ellison, Bob Miner e Ed Oates alla fine degli anni '70. Doveva essere compatibile con System R di IBM.

Versioni

Versione	Data di rilascio
Versione 1 (inedita)	1978-01-01
Oracle V2	1979/01/01
Oracle versione 3	1983/01/01
Oracle versione 4	1984-01-01
Oracle versione 5	1985/01/01
Oracle versione 6	1988/01/01
Oracle7	1992/01/01
Oracle 8	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

Ciao mondo

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

Nel sapore di Oracle di SQL, "dual è solo una tabella di convenienza" . Originariamente era

destinato a raddoppiare le righe tramite un JOIN, ma ora contiene una riga con un valore `DUMMY` di 'X'.

Ciao mondo! dal tavolo

Crea una tabella semplice

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

Inserisci valori (puoi omettere le colonne di destinazione se fornisci valori per tutte le colonne)

```
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');
```

Ricordarsi di eseguire il commit, poiché Oracle utilizza le *transazioni*

```
commit;
```

Seleziona i tuoi dati:

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

SQL Query

Elenca i dipendenti che guadagnano più di $ 50000 nati in questo secolo. Elenca il loro nome, data di nascita e stipendio, in ordine alfabetico per nome.

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

Mostra il numero di dipendenti in ogni dipartimento con almeno 5 dipendenti. Elenca prima i reparti più grandi.

```
SELECT department_id, COUNT(*)
```

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

Ciao mondo da 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;
```

Leggi Iniziare con il database Oracle online: https://riptutorial.com/it/oracle/topic/558/iniziare-con-il-database-oracle

Capitolo 2: Aggiorna con Joins

introduzione

Contrariamente a diffusi fraintendimenti (anche su SO), Oracle consente aggiornamenti tramite join. Tuttavia, ci sono alcuni requisiti (piuttosto logici). Illustriamo cosa non funziona e cosa fa attraverso un semplice esempio. Un altro modo per ottenere lo stesso è la dichiarazione MERGE.

Examples

Esempi: cosa funziona e cosa 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.
```

Immagina cosa succederebbe se avessimo il valore `1` nella colonna `src.id` più di una volta, con valori diversi per `src.val` . Ovviamente, l'aggiornamento non avrebbe senso (in QUALSIASI database - questo è un problema logico). Ora, **sappiamo** che non ci sono duplicati in `src.id` , ma il motore di Oracle non sa che - quindi è lamentarsi. Forse è questo il motivo per cui così tanti professionisti credono che Oracle "non abbia UPDATE con join"?

Quello che Oracle si aspetta è che `src.id` dovrebbe essere unico e che, Oracle, lo saprebbe in anticipo. Facilmente risolto! Si noti che lo stesso funziona con i tasti compositi (su più di una colonna), se la corrispondenza per l'aggiornamento deve utilizzare più di una colonna. In pratica, `src.id` potrebbe essere PK e `tgt.id` potrebbe essere FK che punta a questo PK, ma questo non è rilevante per gli aggiornamenti con join; ciò che *è* rilevante è il vincolo univoco.

```
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
```

Lo stesso risultato potrebbe essere ottenuto con una dichiarazione MERGE (che merita il proprio articolo di documentazione) e personalmente preferisco MERGE in questi casi, ma la ragione non è che "Oracle non aggiorni gli aggiornamenti con join". Come mostra questo esempio, Oracle *esegue* aggiornamenti con join.

Leggi Aggiorna con Joins online: https://riptutorial.com/it/oracle/topic/8061/aggiorna-con-joins

Capitolo 3: Blocco anonimo di PL / SQL

Osservazioni

Poiché sono senza nome, i blocchi anonimi non possono essere referenziati da altre unità di programma.

Examples

Un esempio di blocco anonimo

```
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;
/
```

Leggi Blocco anonimo di PL / SQL online: https://riptutorial.com/it/oracle/topic/6451/blocco-anonimo-di-pl---sql

Capitolo 4: Creare un contesto

Sintassi

- CREA lo spazio dei nomi [O REPLACE] CONTESTUO UTILIZZANDO il pacchetto [schema.];
- CREATE [O REPLACE] namespace CONTEXT UTILIZZO del pacchetto [schema.] INIZIALIZZATO ESTERNAMENTE;
- CREATE [O REPLACE] namespace CONTEXT UTILIZZO del pacchetto [schema.] INIZIALIZZATO GLOBALMENTE;
- CREATE [O REPLACE] namespace CONTEXT UTILIZZO del pacchetto [schema.] ACCESSED GLOBALLY;

Parametri

Parametro	Dettagli
`OR REPLACE`	Ridefinire uno spazio dei nomi di contesto esistente
namespace	Nome del contesto: questo è lo spazio dei nomi per le chiamate a `SYS_CONTEXT`
schema	Proprietario del pacchetto
pacchetto	Pacchetto di database che imposta o reimposta gli attributi di contesto. Nota: il pacchetto database non deve esistere per creare il contesto.
`INITIALIZED`	Specificare un'entità diversa dal Database Oracle che può impostare il contesto.
`EXTERNALLY`	Consentire all'interfaccia OCI di inizializzare il contesto.
`GLOBALLY`	Consentire alla directory LDAP di inizializzare il contesto quando si stabilisce la sessione.
`ACCESSED GLOBALLY`	Consentire al contesto di essere accessibile attraverso l'intera istanza: più sessioni possono condividere i valori degli attributi purché abbiano lo stesso ID client.

Osservazioni

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

Examples

Crea un contesto

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

Questo crea un contesto che può essere impostato solo da routine nel pacchetto database `my_pkg` , ad esempio:

```
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;
```

Ora, se una sessione fa questo:

```
my_pkg.set_ctx;
```

Ora può recuperare il valore per la chiave in questo modo:

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

Value
```

Leggi Creare un contesto online: https://riptutorial.com/it/oracle/topic/2088/creare-un-contesto

Capitolo 5: Data Pump

introduzione

Di seguito sono riportati i passaggi per creare un'importazione / esportazione di un motore di dati:

Examples

Monitorare i lavori di Datapump

I lavori di Datapump possono essere monitorati usando

1. viste del dizionario dati:

```
   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. Stato del datapump:

- Annotare il nome del lavoro dai registri di importazione / esportazione o dal nome del dizionario dati e
- Esegui il comando **attach** :
- digita lo stato nel prompt Importa / Esporta

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

Premere premere **CTRL + C** per uscire dalla richiesta di importazione / esportazione

Passaggio 3/6: creare la directory

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

Passaggio 7: comandi di esportazione

comandi:

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

* Si prega di sostituire i dati in <> con i valori appropriati secondo il proprio ambiente. È possibile aggiungere / modificare i parametri secondo le proprie esigenze. Nell'esempio sopra tutti i parametri rimanenti sono aggiunti nei file parametri come indicato di seguito: *

- Tipo di esportazione: **esportazione utente**
- Esporta l'intero schema
- Dettagli del file dei parametri [ad esempio exp.par]:

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

- Tipo di esportazione: **esportazione utente per schema di grandi dimensioni**
- Esportare l'intero schema per set di dati di grandi dimensioni: qui i file di esportazione vengono suddivisi e compressi. Parallelismo è usato qui *(Nota: l'aggiunta di parallelismo aumenterà il carico della CPU sul server)*
- Dettagli del file dei parametri [ad esempio exp.par]:

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

- Tipo di esportazione: **tabella Esporta** [Esporta serie di tabelle]
- Dettagli del file dei parametri [ad esempio exp.par]:

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

Passaggio 9: comandi di importazione

Prerequisiti:

- Prima di importare l'utente è buona pratica abbandonare lo schema o la tabella importati.

comandi:

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

* Si prega di sostituire i dati in <> con i valori appropriati secondo il proprio ambiente. È possibile aggiungere / modificare i parametri secondo le proprie esigenze. Nell'esempio sopra tutti i parametri rimanenti sono aggiunti nei file parametri come indicato di seguito: *

- Tipo di importazione: **importazione utente**
- Importa l'intero schema
- Dettagli del file di parametri [say imp.par]:

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

- Tipo di importazione: **Importazione utente per schema di grandi dimensioni**
- Importa l'intero schema per set di dati di grandi dimensioni: qui viene utilizzato il parallelismo *(Nota: l'aggiunta di parallelismo aumenterà il carico della CPU sul server)*
- Dettagli del file di parametri [say imp.par]:

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

- Tipo di importazione: **tabella Import** [Importa set di tabelle]
- Dettagli del file di parametri [say 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. Passaggi di Datapump

Server di origine [Esporta dati]	Server di destinazione [Importa dati]
1. Creare una cartella di datapump che conterrà i file di esportazione esportati	4. Creare una cartella di datapump che conterrà i file di dump di importazione
2. Accedere allo schema del database che eseguirà l'esportazione.	5. Accedere allo schema del database che eseguirà l'importazione.
3. Creare la directory che punta al passaggio 1.	6. Creare la directory che punta al passaggio 4.
7. Esegui le dichiarazioni di esportazione.	
8. Copiare / SCP i file di dettagli su Target Server.	
	9. Esegui le istruzioni di importazione
	10. controllare i dati, compilare oggetti non validi e fornire sussidi correlati

Copia le tabelle tra diversi schemi e tablespace

```
 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>
```

Leggi Data Pump online: https://riptutorial.com/it/oracle/topic/9391/data-pump

Capitolo 6: Date

Examples

Generazione di date senza componente orario

Tutti i `DATE` hanno un componente orario; tuttavia, è consuetudine memorizzare le date che non devono includere informazioni sull'ora con le ore / minuti / secondi impostati su zero (ad es. a mezzanotte).

Utilizza un valore letterale ANSI `DATE` (utilizzando il formato data ISO 8601):

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

Converti da una stringa letterale utilizzando `TO_DATE()` :

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

(Ulteriori informazioni sui modelli di formato della data sono disponibili nella documentazione di 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;
```

(Se si convertono termini specifici della lingua, come i nomi dei mesi, è buona norma includere il terzo parametro `nlsparam` nella funzione `TO_DATE()` e specificare la lingua da aspettarsi.)

Generazione di date con un componente temporale

Converti da una stringa letterale utilizzando `TO_DATE()` :

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

Oppure usa un letterale `TIMESTAMP` :

```
CREATE TABLE date_table(
  date_value DATE
);

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

Oracle trasmetterà implicitamente un `TIMESTAMP` a un `DATE` quando lo memorizza in una colonna `DATE` di una tabella; tuttavia puoi `CAST()` esplicitamente il valore di una `DATE` :

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

Il formato di una data

In Oracle, un tipo di dati `DATE` non ha un formato; quando Oracle invia un `DATE` al programma client (SQL / Plus, SQL / Developer, Toad, Java, Python, ecc.) invierà 7 o 8 byte che rappresentano la data.

Una `DATE` che non è memorizzata in una tabella (cioè generata da `SYSDATE` e che ha "tipo 13" quando si usa il comando `DUMP()`) ha 8 byte e ha la struttura (i numeri sulla destra sono la rappresentazione interna di `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` che è memorizzata in una tabella ("tipo 12" quando si usa il comando `DUMP()`) ha 7 byte e ha la struttura (i numeri sulla destra sono la rappresentazione interna di `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
```

Se vuoi che la data abbia un formato specifico, dovrai convertirla in qualcosa che ha un formato (cioè una stringa). Il client SQL può implicitamente farlo o è possibile convertire esplicitamente il valore in una stringa utilizzando `TO_CHAR( date, format_model, nls_params )` .

Conversione delle date in una stringa

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

(Nota: se non viene fornito un modello di formato, il parametro della sessione `NLS_DATE_FORMAT` verrà utilizzato come modello di formato predefinito , che può essere diverso per ogni sessione, quindi non dovrebbe essere invocato.È buona norma specificare sempre il modello di 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 );
```

Poi:

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

Uscite:

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

E:

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

Uscite:

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

Impostazione del modello di formato data predefinito

Quando Oracle converte implicitamente da un `DATE` a una stringa o viceversa (o quando `TO_CHAR()` o `TO_DATE()` vengono chiamati esplicitamente senza un modello di formato) il parametro della sessione `NLS_DATE_FORMAT` verrà utilizzato come modello di formato nella conversione. Se il valore letterale non corrisponde al modello di formato, verrà sollevata un'eccezione.

È possibile rivedere questo parametro utilizzando:

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

È possibile impostare questo valore all'interno della sessione corrente usando:

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

(Nota: questo non cambia il valore per nessun altro utente).

Se ti affidi a `NLS_DATE_FORMAT` per fornire la maschera di formato in `TO_DATE()` o `TO_CHAR()` non dovresti sorprenderti quando le query si interrompono se questo valore viene modificato.

Modifica delle date di visualizzazione degli sviluppatori SQL / Plus o SQL

Quando SQL / Plus o SQL Developer visualizzano le date, eseguiranno una conversione implicita in una stringa utilizzando il modello di formato data predefinito (vedere l'esempio Impostazione del modello di formato data predefinito).

È possibile modificare la modalità di visualizzazione di una data modificando il parametro `NLS_DATE_FORMAT` .

Data Aritmetica - Differenza tra le date in giorni, ore, minuti e / o secondi

In oracolo, la differenza (in giorni e / o frazioni) tra due `DATE` può essere trovata usando la sottrazione:

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

Emette il numero di giorni tra le due date:

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

E:

```
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
```

Emette la frazione di giorni tra due date:

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

La differenza in ore, minuti o secondi può essere trovata moltiplicando questo numero rispettivamente di `24` , `24*60` o `24*60*60` .

L'esempio precedente può essere modificato per ottenere i giorni, le ore, i minuti e i secondi tra due date utilizzando:

```
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()` viene utilizzato anziché `FLOOR()` per gestire correttamente le differenze negative).

Uscite:

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

L'esempio precedente può anche essere risolto convertendo la differenza numerica in un intervallo utilizzando `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
);
```

Data Aritmetica - Differenza tra le date in mesi o anni

La differenza di mesi tra due date può essere trovata utilizzando il `MONTHS_BETWEEN( date1, date2 )` :

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

Uscite:

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

Se la differenza include mesi parte, restituirà la frazione del mese in base a **31** giorni in ciascun mese:

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

Uscite:

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

A causa di `MONTHS_BETWEEN` ipotizzando 31 giorni al mese in cui ci possono essere meno giorni al mese, questo può generare valori diversi per le differenze che si estendono tra i mesi.

Esempio:

```
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;
```

Produzione:

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

La differenza in anni può essere trovata dividendo la differenza del mese di 12.

Estrarre l'anno, mese, giorno, ora, minuti o secondi componenti di una data

I componenti di anno, mese o giorno di un tipo di dati `DATE` possono essere trovati utilizzando `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;
```

Uscite:

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

I componenti del tempo (ore, minuti o secondi) possono essere trovati da entrambi:

- Utilizzando `CAST( datevalue AS TIMESTAMP )` per convertire il `DATE` in un `TIMESTAMP` e quindi usando `EXTRACT( [ HOUR | MINUTE | SECOND ] FROM timestampvalue )` ; o
- Usare `TO_CHAR( datevalue, format_model )` per ottenere il valore come stringa.

Per esempio:

```
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
);
```

Uscite:

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

Fusi orari e ora legale

Il tipo di dati `DATE` non gestisce i fusi orari o le modifiche all'ora legale.

O:

- utilizzare il tipo di dati `TIMESTAMP WITH TIME ZONE` ; o
- gestire le modifiche nella logica dell'applicazione.

Una `DATE` può essere archiviata come Coordinated Universal Time (UTC) e convertita nel fuso orario della sessione corrente in questo modo:

```
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;
```

Se esegui `ALTER SESSION SET TIME_ZONE = '+01:00';` quindi l'output è:

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

e `ALTER SESSION SET TIME_ZONE = 'PST';` quindi l'output è:

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

Leap Seconds

Oracle non gestisce i secondi bisestili . Per ulteriori dettagli, `2019397.2` nota My Oracle Support `2019397.2` e `730795.1` .

Ottenere il giorno della settimana

Puoi utilizzare `TO_CHAR( date_value, 'D' )` per ottenere il giorno della settimana.

Tuttavia, questo dipende dal parametro della sessione `NLS_TERRITORY` :

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

Uscite `5`

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

Uscite `4`

Per fare ciò indipendentemente dalle impostazioni `NLS` , puoi troncare la data a mezzanotte del giorno corrente (per rimuovere eventuali frazioni di giorni) e sottrarre la data troncata all'inizio della iso-settimana corrente (che inizia sempre il lunedì):

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

Leggi Date online: https://riptutorial.com/it/oracle/topic/2087/date

Capitolo 7: Delimitare parole chiave o caratteri speciali

Examples

Delimitare il nome della tabella o della colonna con caratteri speciali

Seleziona * da ditta's_ress;

Seleziona * da "ditta's_address";

Delimitare il nome della tabella o della colonna che è anche una parola riservata

Supponi di avere una tabella denominata tabella o di creare una tabella con un nome che sia anche una parola chiave, devi includere la tabella dei nomi in coppia di virgolette "tabella"

Seleziona * dalla tabella; La query precedente non riuscirà con l'errore di sintassi, dove la query di seguito verrà eseguita correttamente.

Seleziona * da "tabella";

Leggi Delimitare parole chiave o caratteri speciali online:
https://riptutorial.com/it/oracle/topic/6553/delimitare-parole-chiave-o-caratteri-speciali

Capitolo 8: Diversi modi per aggiornare i record

Sintassi

- UPDATE table-Name [[AS] correlation-Name] SET column-Name = Value [, column-Name = Value}] * [WHERE clause]
- UPDATE tabella-Nome SET colonna-Nome = Valore [, colonna-Nome = Valore] * WHERE CORRENTE DI

Examples

Aggiorna la sintassi con l'esempio

Aggiornamento normale

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

Aggiorna usando Vista in linea

Utilizzo della vista in linea (Se è considerato aggiornabile da Oracle)

Nota : se si affronta un errore di riga preservato non chiave, aggiungere un indice per risolvere lo stesso per renderlo aggiornabile

```
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;
```

Aggiorna usando Unisci

Utilizzando l'unione

```
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';
```

Unisci con dati di esempio

```
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);
```

Leggi Diversi modi per aggiornare i record online: https://riptutorial.com/it/oracle/topic/4193/diversi-modi-per-aggiornare-i-record

Capitolo 9: Divisione di stringhe delimitate

Examples

Divisione di stringhe utilizzando una clausola di factoring in sub-query ricorsiva

Dati di esempio :

```
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
```

Domanda :

```
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;
```

Uscita :

```
        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
```

Divisione di stringhe utilizzando una funzione PL / SQL

Funzione 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;
/
```

Dati di esempio :

```
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
```

Domanda :

```
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
```

Uscita :

```
        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
```

Divisione di stringhe utilizzando un'espressione di tabella correlata

Dati di esempio :

```
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
```

Domanda :

```
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;
```

Uscita :

```
        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
```

Divisione di stringhe mediante una query gerarchica

Dati di esempio :

```
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
```

Domanda :

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

Uscita :

```
        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
```

Divisione di stringhe mediante espressioni XMLTable e FLWOR

Questa soluzione utilizza l' `ora:tokenize` funzione XQuery disponibile da Oracle 11.

Dati di esempio :

```
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
```

Domanda :

```
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;
```

Uscita :

```
        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
```

Divisione di stringhe utilizzando CROSS APPLY (Oracle 12c)

Dati di esempio :

```
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
```

Domanda :

```
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;
```

Uscita :

```
        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
```

Dividere le stringhe delimitate usando XMLTable

Dati di esempio :

```
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
```

Domanda :

```
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: il carattere `#` viene aggiunto per facilitare l'estrazione dei valori `NULL` , viene successivamente rimosso usando `SUBSTR( item, 2 )` . Se i valori `NULL` non sono richiesti, è possibile semplificare la query e ometterla.)

Uscita :

```
        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
```

Leggi Divisione di stringhe delimitate online: https://riptutorial.com/it/oracle/topic/1968/divisione-di-stringhe-delimitate

Capitolo 10: Dizionario dei dati

Osservazioni

Le viste del dizionario dati, note anche come viste del catalogo, consentono di monitorare lo stato del database in tempo reale:

Le viste con prefisso `USER_` , `ALL_` e `DBA_` , mostrano informazioni sugli oggetti dello schema di proprietà dell'utente (`USER_`), accessibili dall'utente (`ALL_`) o accessibili da un utente con privilegio SYSDBA (`DBA_`). Ad esempio, la vista `ALL_TABLES` mostra tutte le tabelle su cui si dispone di privilegi.

Le visualizzazioni `V$` mostrano informazioni relative alle prestazioni.

Le viste `_PRIVS` mostrano le informazioni sui privilegi per diverse combinazioni di utenti, ruoli e oggetti.

Documentazione Oracle: Viste catalogo / Viste dizionario dati

Examples

Origine del testo degli oggetti memorizzati

`USER_SOURCE` descrive l'origine del testo degli oggetti memorizzati di proprietà dell'utente corrente. Questa vista non mostra la colonna `OWNER` .

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

`ALL_SOURCE` descrive la fonte di testo degli oggetti memorizzati accessibili all'utente corrente.

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

`DBA_SOURCE` descrive l'origine del testo di tutti gli oggetti memorizzati nel database.

```
select * from dba_source
```

Ottieni l'elenco di tutte le tabelle in Oracle

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

`ALL_TAB_COLUMNS` descrive le colonne di tabelle, viste e cluster accessibili all'utente corrente. `COLS` è un sinonimo di `USER_TAB_COLUMNS` .

```
select *
```

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

Informazioni privilegiate

Tutti i ruoli concessi all'utente.

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

Privilegi concessi all'utente:

1. privilegi di sistema

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

2. concessioni di oggetti

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

Autorizzazioni concesse ai ruoli.

Ruoli concessi ad altri ruoli.

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

1. privilegi di sistema

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

2. concessioni di oggetti

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

Versione Oracle

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

Descrive tutti gli oggetti nel database.

```
select *
from dba_objects
```

Per vedere tutte le viste del dizionario dati a cui si ha accesso

```
select * from dict
```

Leggi Dizionario dei dati online: https://riptutorial.com/it/oracle/topic/7347/dizionario-dei-dati

Capitolo 11: Factoring ricorsivo sotto-query utilizzando la clausola WITH (espressioni di tabella comuni AKA)

Osservazioni

Il factoring sub-query ricorsivo è disponibile in Oracle 11g R2.

Examples

Un semplice generatore di numeri interi

Domanda :

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

Uscita :

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

Divisione di una stringa delimitata

Dati di esempio :

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

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

Domanda :

```
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;
```

Uscita :

```
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
```

Leggi Factoring ricorsivo sotto-query utilizzando la clausola WITH (espressioni di tabella comuni AKA) online: https://riptutorial.com/it/oracle/topic/3506/factoring-ricorsivo-sotto-query-utilizzando-la-clausola-with--espressioni-di-tabella-comuni-aka-

Capitolo 12: Funzioni della finestra

Sintassi

- Ratio_To_Report (expr) OVER (query_partition_clause)

Examples

Ratio_To_Report

Fornisce il rapporto tra il valore corrente delle righe e tutti i valori all'interno della finestra.

```
--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
```

Leggi Funzioni della finestra online: https://riptutorial.com/it/oracle/topic/6669/funzioni-della-finestra

Capitolo 13: Funzioni statistiche

Examples

Calcolo della mediana di un insieme di valori

La funzione MEDIAN dal momento che Oracle 10g è una funzione di aggregazione facile da usare:

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

Restituisce la mediana dei valori

Funziona anche con i valori `DATETIME` .

> Il risultato di MEDIAN viene calcolato prima ordinando le righe. Utilizzando N come numero di righe nel gruppo, Oracle calcola il numero di riga (RN) di interesse con la formula RN = (1 + (0,5 * (N-1)). Il risultato finale della funzione di aggregazione è calcolato da lineare interpolazione tra i valori delle righe ai numeri di riga CRN = CEILING (RN) e FRN = FLOOR (RN).

Dal momento che Oracle 9i è possibile utilizzare PERCENTILE_CONT che funziona come la funzione MEDIAN con valori percentili predefiniti a 0,5

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

VARIANZA

La varianza misura quanto distano un numero impostato dalla sua media. Dal punto di vista pratico è una distanza quadrata dalla sua media (al centro) - più grande è il numero più lontano è il punto.

L'esempio seguente restituirebbe la varianza dei valori di stipendio

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

STDDEV

STDDEV restituisce la deviazione standard campionaria di expr, un insieme di numeri. Puoi usarlo sia come funzione aggregata che analitica. Differisce da STDDEV_SAMP in quanto STDDEV restituisce zero quando ha solo 1 riga di dati di input, mentre STDDEV_SAMP restituisce null.

Oracle Database calcola la deviazione standard come radice quadrata della varianza definita per

la funzione di aggregazione VARIANCE.

Questa funzione accetta come argomento qualsiasi tipo di dato numerico o qualsiasi tipo di dati non numerico che può essere convertito implicitamente in un tipo di dati numerico. La funzione restituisce lo stesso tipo di dati del tipo di dati numerici dell'argomento.

Se si specifica DISTINCT, è possibile specificare solo la query_partition_clause di analytic_clause. Order_by_clause e windowing_clause non sono consentiti.

L'esempio seguente restituisce la deviazione standard degli stipendi nella tabella **hr.employees di** esempio:

Dove hr è Schema e dipendenti è un nome di tabella.

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

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

La query nel seguente esempio restituisce la deviazione standard cumulativa degli stipendi nel reparto 80 nella tabella di esempio hr.employees, ordinata in 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
```

Leggi Funzioni statistiche online: https://riptutorial.com/it/oracle/topic/2283/funzioni-statistiche

Capitolo 14: Gestire valori NULL

introduzione

Una colonna è NULL quando non ha alcun valore, indipendentemente dal tipo di dati di quella colonna. Una colonna non dovrebbe mai essere paragonata a NULL usando questa sintassi `a = NULL` quanto il risultato sarebbe UNKNOWN. Usa invece `a IS NULL` o `a IS NOT NULL` . NULL non è uguale a NULL. Per confrontare due espressioni in cui può verificarsi null, utilizzare una delle funzioni descritte di seguito. Tutti gli operatori tranne la concatenazione restituiscono NULL se uno dei loro operandi è NULL. Ad esempio il risultato di `3 * NULL + 5` è nullo.

Osservazioni

NULL non può apparire in colonne limitate da una PRIMARY KEY o da un vincolo NOT NULL. (L'eccezione è un nuovo vincolo con la clausola NOVALIDATE)

Examples

Le colonne di qualsiasi tipo di dati possono contenere valori NULL

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

NUM_COLUMN	VARCHAR2_COLUMN
1	foo
(nullo)	(nullo)

Le stringhe vuote sono NULL

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

UN	B
1	(nullo)

Le operazioni che contengono NULL sono NULL, tranne la concatenazione

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

3 * NULL + 5	'CIAO' || NULL || 'Mondo'
(nullo)	Ciao mondo

NVL per sostituire il valore nullo

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

COLUMN_WITH_NULL	COLUMN_WITHOUT_NULL
(nullo)	N / A

NVL è utile per confrontare due valori che possono contenere valori NULL:

```
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	COMPARISON_WITH_NVL
-1	0
-1	1

NVL2 per ottenere un risultato diverso se un valore è nullo o no

Se il primo parametro NON è NULL, NVL2 restituirà il secondo parametro. Altrimenti restituirà il terzo.

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

NVL2 (NULL, 'FOO', 'bar')	NVL2 (5, 'FOO', 'bar')
Bar	foo

COALESCE per restituire il primo valore non 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

In alcuni casi, l'utilizzo di COALESCE con due parametri può essere più rapido rispetto all'utilizzo di NVL quando il secondo parametro non è una costante. NVL valuterà sempre entrambi i parametri. COALESCE si fermerà al primo valore non NULL che incontra. Significa che se il primo valore non è NULL, COALESCE sarà più veloce.

Leggi Gestire valori NULL online: https://riptutorial.com/it/oracle/topic/8183/gestire-valori-null

Capitolo 15: indici

introduzione

Qui spiegherò diversi indici usando l'esempio, come l'indice aumenta le prestazioni della query, come l'indice riduce le prestazioni DML, ecc

Examples

indice b-tree

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

Per impostazione predefinita, se non menzioniamo nulla, oracle crea un indice come indice b-tree. Ma dovremmo sapere quando usarlo. L'indice B-tree memorizza i dati come formato di albero binario. Come sappiamo, index è un oggetto schema che memorizza una sorta di voce per ogni valore per la colonna indicizzata. Quindi, ogni volta che una ricerca avviene su quelle colonne, controlla nell'indice la posizione esatta di quel record per accedere velocemente. Pochi punti sull'indicizzazione:

- Per cercare la voce nell'indice, viene utilizzata una sorta di algoritmo di ricerca binaria.
- Quando **la cardinalità dei dati è elevata, l'** indice **b-tree** è perfetto da usare.
- L'indice rende lento DML, come per ogni record, ci dovrebbe essere una voce nell'indice per la colonna indicizzata.
- Quindi, se non necessario, dovremmo evitare di creare indici.

Indice bitmap

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

- L'indice bitmap viene utilizzato quando **la cardinalità dei dati è bassa.**
- Qui, **Gender** ha un valore con cardinalità bassa. I valori possono essere Maschio, Femmina e altri.
- Quindi, se creiamo un albero binario per questi 3 valori durante la ricerca avrà una traversata non necessaria.
- Nelle strutture bitmap, viene creato un array bidimensionale con una colonna per ogni riga della tabella che viene indicizzata. Ogni colonna rappresenta un valore distinto all'interno dell'indice bitmap. Questo array bidimensionale rappresenta ciascun valore all'interno dell'indice moltiplicato per il numero di righe nella tabella.
- Al momento del recupero della riga, Oracle decomprime la bitmap nei buffer dei dati RAM in modo che possa essere rapidamente scansionata per trovare i valori corrispondenti. Questi valori di corrispondenza vengono consegnati a Oracle sotto forma di un elenco ID di riga e

questi valori di ID di riga possono accedere direttamente alle informazioni richieste.

Indice basato sulle funzioni

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


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

- Indice basato sulla funzione significa, creazione dell'indice basato su una funzione.
- Se nella ricerca (clausola where), viene utilizzata frequentemente qualsiasi funzione, è preferibile creare un indice basato su tale funzione.
- Qui, nell'esempio, per la ricerca, viene utilizzata la funzione **Upper ()** . Quindi, è meglio creare un indice usando la funzione superiore.

Leggi indici online: https://riptutorial.com/it/oracle/topic/9978/indici

Capitolo 16: Lavorare con le date

Examples

Data Aritmetica

Oracle supporta i tipi di dati `DATE` (include il tempo per il secondo più prossimo) e `TIMESTAMP` (include il tempo per le frazioni di un secondo), che consentono l'aritmetica (addizione e sottrazione) in modo nativo. Per esempio:

Per ottenere il giorno successivo:

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

Per ottenere il giorno precedente:

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

Per aggiungere 5 giorni alla data corrente:

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

Per aggiungere 5 ore alla data corrente:

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

Per aggiungere 10 minuti alla data corrente:

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

Per aggiungere 7 secondi alla data corrente:

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

Per selezionare le righe in cui `hire_date` sono `hire_date` 30 giorni o più:

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

Per selezionare le righe in cui la colonna `last_updated` trova nell'ultima ora:

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

Oracle fornisce anche il tipo di dati `INTERVAL` incorporato che rappresenta un periodo di tempo (ad

es. 1,5 giorni, 36 ore, 2 mesi, ecc.). Questi possono anche essere usati con l'aritmetica con le espressioni `DATE` e `TIMESTAMP` . Per esempio:

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

Funzione Add_months

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

La funzione Add_months aggiunge amt mesi alla data p_date.

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

M
2015/03/12

Puoi anche sottrarre mesi usando un `amt` negativo

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

M
2014/11/12

Quando il mese calcolato ha meno giorni della data indicata, verrà restituito l'ultimo giorno del mese calcolato.

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

M
2015/02/28

Leggi Lavorare con le date online: https://riptutorial.com/it/oracle/topic/768/lavorare-con-le-date

Capitolo 17: Limitazione delle righe restituite da una query (impaginazione)

Examples

Ottieni le prime N righe con clausola di limitazione delle righe

La clausola `FETCH` è stata introdotta in Oracle 12c R1:

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

Un esempio senza FETCH che funziona anche nelle versioni precedenti:

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

Impaginazione in 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 ;
```

in questo modo possiamo impaginare i dati della tabella, proprio come la pagina web serch

Ottieni N numeri di record dalla tabella

Non possiamo limitare il numero di righe dal risultato usando la clausola rownum

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

Se vogliamo il primo o l'ultimo record, vogliamo la clausola order by nella query interna che darà il risultato in base all'ordine.

Last Five Record:

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

First Five Record

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

Ottieni la riga da N a M da più righe (prima di Oracle 12c)

Usa la funzione analitica 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 lo gestisce più facilmente con `OFFSET` e `FETCH` .

Saltare alcune righe e poi prenderne un po '

In Oracle 12g +

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

Nelle versioni precedenti

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

Saltare alcune righe dal risultato

In Oracle 12g +

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

Nelle versioni precedenti

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

Leggi Limitazione delle righe restituite da una query (impaginazione) online: https://riptutorial.com/it/oracle/topic/4300/limitazione-delle-righe-restituite-da-una-query--impaginazione-

Capitolo 18: Link al database

Examples

Creazione di un collegamento al database

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

Il DB remoto sarà quindi accessibile nel modo seguente:

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

Per verificare una connessione al collegamento del database senza dover conoscere nessuno dei nomi oggetto nel database collegato, utilizzare la seguente query:

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

Per specificare esplicitamente un dominio per il servizio di database collegato, il nome di dominio viene aggiunto all'istruzione `USING` . Per esempio:

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

Se non viene specificato alcun nome di dominio, Oracle utilizza il dominio del database in cui viene creato il collegamento.

Documentazione Oracle per la creazione di collegamenti al database:

- 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

Crea collegamento al database

Supponiamo di avere due database "ORA1" e "ORA2". Possiamo accedere agli oggetti di "ORA2" dal database "ORA1" utilizzando un collegamento al database.

Prerequisiti: per creare un collegamento a un database privato è necessario un privilegio `CREATE DATABASE LINK` . Per creare un collegamento a un database privato è necessario un privilegio `CREATE PUBLIC DATABASE LINK` .

* Oracle Net deve essere presente su entrambe le istanze.

Come creare un collegamento al database:

Da ORA1:

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

Collegamento al database creato.

Ora che abbiamo impostato il collegamento DB, possiamo provarlo eseguendo quanto segue da ORA1:

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

È inoltre possibile accedere agli oggetti DB di "ORA2" da "ORA1", dato l'utente `user1` ha il privilegio `SELECT` su quegli oggetti su ORA2 (come TABELLA1 di seguito):

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

Pre-requistes:

- Entrambi i database devono essere attivi e in esecuzione (aperti).
- Entrambi i listener di database devono essere attivi e in esecuzione.
- TNS deve essere configurato correttamente.
- L'utente utente1 deve essere presente nel database ORA2, la password deve essere verificata e verificata.
- L'utente utente1 deve avere almeno il privilegio `SELECT` o qualsiasi altro richiesto per accedere agli oggetti su ORA2.

Leggi Link al database online: https://riptutorial.com/it/oracle/topic/3859/link-al-database

Capitolo 19: MAF Oracle

Examples

Per ottenere valore da Binding

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

Qui "binding" indica l'espressione EL da cui il valore deve essere ottenuto.

"nome_variabile" il parametro al quale deve essere memorizzato il valore dall'associazione

Per impostare il valore sull'associazione

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

Qui "binding" indica l'espressione EL a cui il valore deve essere memorizzato.

"valore" è il valore desiderato da aggiungere alla rilegatura

Per richiamare un metodo dall'associazione

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

"binding" indica l'espressione EL da cui viene invocato un metodo

Per chiamare una funzione javaScript

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

"function" è la funzione js desiderata per essere invocata

Leggi MAF Oracle online: https://riptutorial.com/it/oracle/topic/6352/maf-oracle

Capitolo 20: Manipolazione delle stringhe

Examples

Concatenazione: operatore || o concat ()

Oracle SQL e PL / SQL `||` operatore consente di concatenare 2 o più stringhe insieme.

Esempio:

Supponendo la seguente tabella `customers` :

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

Query:

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

Produzione:

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

Oracle supporta anche la funzione standard SQL `CONCAT(str1, str2)` :

Esempio:

Query:

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

Produzione:

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

SUPERIORE

La funzione UPPER consente di convertire tutte le lettere minuscole in una stringa in maiuscolo.

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

Produzione:

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

INITCAP

La funzione `INITCAP` converte il caso di una stringa in modo che ogni parola inizi con una lettera maiuscola e tutte le lettere successive siano in minuscolo.

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

Produzione

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

INFERIORE

LOWER converte tutte le lettere maiuscole in una stringa in lettere minuscole.

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

Uscite:

testo
ciao world123!

Espressione regolare

Diciamo che vogliamo sostituire solo i numeri con 2 cifre: l'espressione regolare li troverà con `(\d\d)`

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

Risultati in:

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

Se voglio scambiare parti del testo, uso `\1` , `\2` , `\3` per chiamare le stringhe corrispondenti:

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

SUBSTR

`SUBSTR` recupera parte di una stringa indicando la posizione iniziale e il numero di caratteri da estrarre

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

ritorna:

```
bcd
```

Per contare dalla fine della stringa, `SUBSTR` accetta un numero negativo come secondo parametro, ad es

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

ritorna:

```
de
```

Per ottenere l'ultimo carattere in una stringa: `SUBSTR(mystring,-1,1)`

LTRIM / RTRIM

`LTRIM` e `RTRIM` rimuovere caratteri dall'inizio o fine (rispettivamente) di una stringa. Un set di uno o più caratteri può essere fornito (di default è uno spazio) da rimuovere.

Per esempio,

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

Ritorna:

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

Leggi Manipolazione delle stringhe online:
https://riptutorial.com/it/oracle/topic/1518/manipolazione-delle-stringhe

Capitolo 21: Oracle Advanced Queuing (AQ)

Osservazioni

- Non utilizzare mai DDL o DML contro tabelle create da `dbms_aqadm.create_queue_table` . Utilizzare solo dbms_aqadm e dbms_aq per lavorare con queste tabelle. Oracle potrebbe fare diverse tabelle di supporto, indici, ecc. Di cui non sarai a conoscenza. L'esecuzione manuale di DDL o DML sulla tabella può portare a uno scenario in cui il supporto Oracle richiede la rimozione e la ricreazione della tabella e delle code per risolvere la situazione.
- Si consiglia vivamente di non utilizzare `dbms_aq.forever` per l'opzione di attesa. Ciò ha causato problemi in passato in quanto Oracle potrebbe iniziare a pianificare un numero eccessivo di lavori di lavoro per far funzionare le code non necessarie (vedere Oracle Doc ID 2001165.1).
- Si consiglia di non impostare il parametro AQ_TM_PROCESSES nella versione 10.1 e successive. Evitare in particolare di azzerare questo dato che disabiliterà il lavoro in background QMON necessario per mantenere le code. È possibile ripristinare questo valore su Oracle predefinito utilizzando il seguente comando e riavviando il database. `alter system reset aq_tm_processes scope=spfile sid='*';`

Examples

Semplice produttore / consumatore

Panoramica

Crea una coda alla quale possiamo inviare un messaggio. Oracle notificherà alla nostra stored procedure che un messaggio è stato accodato e deve essere lavorato. Aggiungiamo anche alcuni sottoprogrammi che possiamo utilizzare in caso di emergenza per impedire che i messaggi vengano squalificati, consentire nuovamente la rimozione della coda e avviare un semplice processo batch per gestire tutti i messaggi.

Questi esempi sono stati testati su Oracle Database 12c Enterprise Edition versione 12.1.0.2.0 - Produzione a 64 bit.

Crea coda

Creeremo un tipo di messaggio, una tabella della coda che può contenere i messaggi e una coda. I messaggi nella coda verranno eliminati prima dalla priorità, quindi sarà il loro tempo di accodamento. Se qualcosa va storto, lavorando il messaggio e il dequeue viene eseguito il rollback, AQ renderà il messaggio disponibile per la rimozione della coda dopo 3600 secondi. Lo farà 48 volte prima di spostarlo come una coda di eccezione.

```
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.
```

Ora che abbiamo un posto dove mettere i messaggi, creiamo un pacchetto per gestire e lavorare i messaggi in coda.

```
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;
```

Dì quindi ad AQ che quando un messaggio viene messo in coda a MESSAGE_Q (e impegnato) notifica alla nostra procedura che ha del lavoro da fare. AQ avvierà un lavoro nella propria sessione per gestirlo.

```
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;
```

Avvia coda e invia un messaggio

```
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_1,
                    payload              => message_1,
                    msgid                => message_id_1);
   commit;
end;
```

Leggi Oracle Advanced Queuing (AQ) online: https://riptutorial.com/it/oracle/topic/4362/oracle-advanced-queuing--aq-

Capitolo 22: Partizionamento delle tabelle

introduzione

Il partizionamento è una funzionalità per dividere tabelle e indici in parti più piccole. È usato per migliorare le prestazioni e gestire i pezzi più piccoli individualmente. La chiave di partizione è una colonna o un insieme di colonne che definisce in quale partizione verrà memorizzata ciascuna riga. Panoramica del partizionamento nella documentazione ufficiale di Oracle

Osservazioni

Il partizionamento è un'opzione di costo aggiuntivo e disponibile solo per l'edizione Enterprise.

Examples

Hash partizionamento

Questo crea una tabella partizionata da hash, in questo esempio sull'id del negozio.

```
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;
```

È necessario utilizzare una potenza pari a 2 per il numero di partizioni hash, in modo da ottenere una distribuzione uniforme nella dimensione della partizione.

Range partitioning

Questo crea una tabella partizionata da intervalli, in questo esempio sui valori dell'ordine.

```
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)
);
```

Seleziona le partizioni esistenti

Controllare le partizioni esistenti su Schema

```
SELECT * FROM user_tab_partitions;
```

Elenco partizionamento

Questo crea una tabella partizionata da liste, in questo esempio sull'id del negozio.

```
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)
);
```

Rilascia la partizione

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

Seleziona i dati da una partizione

Seleziona i dati da una partizione

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

Tronca una partizione

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

Rinominare una partizione

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

Sposta la partizione in un tablespace diverso

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

Aggiungi nuova partizione

```
ALTER TABLE table_name
```

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

Divisione partizione

Divide alcune partizioni in due partizioni con un altro limite elevato.

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

Unisci partizioni

Unisci due partizioni in una sola

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

Scambio di una partizione

Scambia / converti una partizione in una tabella non partizionata e viceversa. Questo facilita un rapido "spostamento" di dati tra i segmenti di dati (opposto a fare qualcosa come "inserire ... selezionare" o "creare tabella ... come selezionare") come l'operazione è DDL (l'operazione di scambio di partizione è un dato aggiornamento del dizionario senza spostare i dati effettivi) e non DML (overhead di annullamento / ripristino di grandi dimensioni).

La maggior parte degli esempi di base:

1. Convertire una tabella non partizionata (tabella "B") in una partizione (della tabella "A"):

La tabella "A" non contiene dati nella partizione "OLD_VALUES" e la tabella "B" contiene dati

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

Risultato: i dati vengono "spostati" dalla tabella "B" (non contiene dati dopo l'operazione) alla partizione "OLD_VALUES"

2. Convertire una partizione in una tabella non partizionata:

La tabella "A" contiene dati nella partizione "OLD_VALUES" e la tabella "B" non contiene dati

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

Risultato: i dati vengono "spostati" dalla partizione "OLD_VALUES" (non contiene dati dopo l'operazione) alla tabella "B"

Nota: ci sono alcune opzioni, caratteristiche e restrizioni aggiuntive per questa operazione

Ulteriori informazioni sono disponibili su questo link ---> "
https://docs.oracle.com/cd/E11882_01/server.112/e25523/part_admin002.htm#i1107555 "
(sezione "Scambio di partizioni")

Leggi Partizionamento delle tabelle online:
https://riptutorial.com/it/oracle/topic/3955/partizionamento-delle-tabelle

Capitolo 23: query di livello

Osservazioni

la clausola di livello è responsabile della generazione del numero N di record fittizi in base ad alcune condizioni specifiche.

Examples

Genera N Numero di record

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

Pochi usi di Query di livello

/ * Questa è una semplice query che può generare una sequenza di numeri. L'esempio seguente genera una sequenza di numeri da 1..100 * /

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

/ * La query precedente è utile in vari scenari come la generazione di una sequenza di date da una determinata data. La seguente query genera 10 date consecutive * /

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

01-GEN-17
02-Gen-17
03-Gen-17
04-Gen-17
05-GEN-17
06-GEN-17
07-GEN-17
08-Gen-17
09-GEN-17
10-GEN-17

Leggi query di livello online: https://riptutorial.com/it/oracle/topic/6548/query-di-livello

Capitolo 24: Recupero gerarchico con Oracle Database 12C

introduzione

È possibile utilizzare query gerarchiche per recuperare i dati in base a una relazione gerarchica naturale tra le righe in una tabella

Examples

Uso del collegamento CONNECT BY

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

La clausola `CONNECT BY` per definire la relazione tra dipendenti e manager.

Specificare la direzione della query dall'alto verso il basso

```
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;
```

Leggi Recupero gerarchico con Oracle Database 12C online:
https://riptutorial.com/it/oracle/topic/8777/recupero-gerarchico-con-oracle-database-12c

Capitolo 25: Registrazione errori

Examples

Registrazione errori durante la scrittura nel database

Crea la tabella del log degli errori di Oracle ERR $ _EXAMPLE per la tabella EXAMPLE esistente:

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

Crea un'operazione di scrittura con SQL:

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

Leggi Registrazione errori online: https://riptutorial.com/it/oracle/topic/3505/registrazione-errori

Capitolo 26: sequenze

Sintassi

- CREATE SEQUENCE SCHEMA.SEQUENCE {INCREMENTO DI INTEGER | INIZIA CON INTEGER | MAXVALUE INTEGER | NOMAXVALUE INTEGER | MINVALUE INTEGER | NOMINVALUE INTEGER | CYCLE INTEGER | NEGYCLE INTEGER | CACHE | NOCACHE | ORDINE | NOODER}

Parametri

Parametro	Dettagli
schema	nome dello schema
incrementa di	intervallo tra i numeri
iniziare con	il primo numero necessario
maxvalue	Valore massimo per la sequenza
nomaxvalue	Il valore massimo è predefinito
minvalue	valore minimo per la sequenza
nominvalue	il valore minimo è predefinito
ciclo	Resetta all'inizio dopo aver raggiunto questo valore
NOCYCLE	Predefinito
nascondiglio	Limite di preallocazione
nocache	Predefinito
ordine	Garantire l'ordine dei numeri
Nessun ordine	predefinito

Examples

Creazione di una sequenza: esempio

Scopo

Utilizzare l'istruzione CREATE SEQUENCE per creare una sequenza, che è un oggetto di

database da cui più utenti possono generare numeri interi univoci. È possibile utilizzare le sequenze per generare automaticamente i valori delle chiavi primarie.

Quando viene generato un numero di sequenza, la sequenza viene incrementata, indipendentemente dalla transazione che sta eseguendo o ripristinando. Se due utenti incrementano contemporaneamente la stessa sequenza, allora i numeri di sequenza che ogni utente acquisisce potrebbero avere spazi vuoti, perché i numeri di sequenza vengono generati dall'altro utente. Un utente non può mai acquisire il numero di sequenza generato da un altro utente. Dopo che un valore di sequenza è stato generato da un utente, quell'utente può continuare ad accedere a quel valore indipendentemente dal fatto che la sequenza venga incrementata da un altro utente.

I numeri di sequenza vengono generati indipendentemente dalle tabelle, quindi la stessa sequenza può essere utilizzata per una o più tabelle. È possibile che i singoli numeri di sequenza sembrino saltati, perché sono stati generati e utilizzati in una transazione che alla fine è stata sottoposta a rollback. Inoltre, un singolo utente potrebbe non rendersi conto che altri utenti stanno attingendo dalla stessa sequenza.

Dopo aver creato una sequenza, è possibile accedere ai relativi valori nelle istruzioni SQL con la pseudocolonna CURRVAL, che restituisce il valore corrente della sequenza o la pseudocolonna NEXTVAL, che incrementa la sequenza e restituisce il nuovo valore.

Prerequisiti

Per creare una sequenza nel proprio schema, è necessario disporre del privilegio di sistema CREATE SEQUENCE.

Per creare una sequenza nello schema di un altro utente, è necessario disporre del privilegio di sistema CREA QUALSIASI SEQUENZA.

Creazione di una sequenza: Esempio La seguente istruzione crea la sequenza customers_seq nello schema di esempio oe. Questa sequenza può essere utilizzata per fornire numeri ID cliente quando vengono aggiunte righe alla tabella clienti.

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

Il primo riferimento a customers_seq.nextval restituisce 1000. Il secondo restituisce 1001. Ogni riferimento successivo restituirà un valore 1 maggiore del riferimento precedente.

Leggi sequenze online: https://riptutorial.com/it/oracle/topic/3709/sequenze

Capitolo 27: SI UNISCE

Examples

CROSS JOIN

Un `CROSS JOIN` esegue un join tra due tabelle che non utilizza una clausola join esplicita e produce il prodotto cartesiano di due tabelle. Un prodotto cartesiano significa che ogni riga di una tabella è combinata con ciascuna riga della seconda tabella nel join. Ad esempio, se `TABLEA` ha 20 righe e `TABLEB` ha 20 righe, il risultato sarà `20*20 = 400` righe di output.

Esempio:

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

Questo può anche essere scritto come:

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

Ecco un esempio di cross join in SQL tra due tabelle:

Tabella dei campioni: TABLEA

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

Tabella di esempio: TABELLAB

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

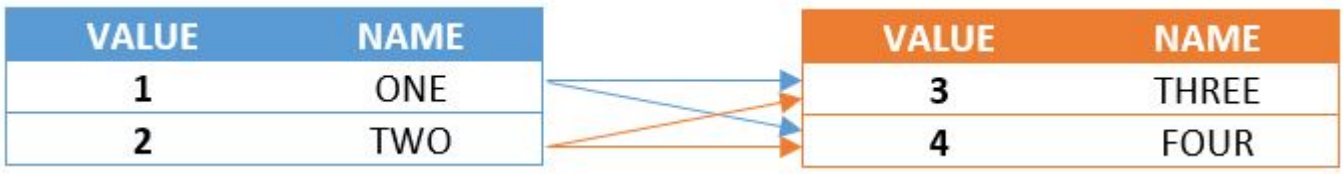
Ora, se si esegue la query:

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

Produzione:

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

Ecco come avviene l'unione incrociata tra due tabelle:



Maggiori informazioni su Cross Join: documentazione Oracle

INNER JOIN

Un INNER JOIN è un'operazione JOIN che consente di specificare una clausola join esplicita.

Sintassi

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

È possibile specificare la clausola join specificando ON con un'espressione booleana.

L'ambito delle espressioni nella clausola ON include le tabelle correnti e tutte le tabelle nei blocchi di query esterni nella SELECT corrente. Nell'esempio seguente, la clausola ON fa riferimento alle tabelle correnti:

```
-- 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
```

SINISTRA ESTERNO

A `LEFT OUTER JOIN` esegue un join tra due tabelle che richiede una clausola join esplicita ma non esclude righe non corrispondenti dalla prima tabella.

Esempio:

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

Anche se la sintassi ANSI è consigliata , è probabile che si verifichi la sintassi legacy molto spesso. L'uso di `(+)` all'interno di una condizione determina quale lato dell'equazione deve essere considerato come *esterno* .

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

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

Ecco un esempio di Left Outer Join tra due tabelle:

Tabella di esempio: IMPIEGATO

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

Tabella dei campioni: DEPT

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

Ora, se si esegue la query:

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

Produzione:

```
+-----------+---------+---------+--------------+
|    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    |         |              |
+-----------+---------+---------+--------------+
```

GIUSTO ESTERNO

A `RIGHT OUTER JOIN` esegue un join tra due tabelle che richiede una clausola join esplicita ma non esclude righe non corrispondenti dalla seconda tabella.

Esempio:

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

Poiché le righe non corrispondenti di `SCOTT.DEPT` sono incluse, ma le righe non corrispondenti di `SCOTT.EMP` non lo sono, quanto sopra è equivalente alla seguente istruzione che utilizza `LEFT OUTER JOIN` .

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

Ecco un esempio di Right Outer Join tra due tabelle:

Tabella di esempio: IMPIEGATO

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

Tabella dei campioni: DEPT

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

```
|    6    |      HR      |
+---------+--------------+
```

Ora, se si esegue la query:

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

Produzione:

```
+-----------+---------+---------+--------------+
|    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      |
+-----------+---------+---------+--------------+
```

L'equivalente di sintassi Oracle (+) per la query è:

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

FULL OUTER JOIN

Un `FULL OUTER JOIN` esegue un join tra due tabelle che richiede una clausola di join esplicita ma non esclude le righe non corrispondenti in entrambe le tabelle. In altre parole, restituisce tutte le righe in ogni tabella.

Esempio:

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

Ecco un esempio di Full Outer Join tra due tabelle:

Tabella di esempio: IMPIEGATO

```
+-----------+---------+
|    NAME   |  DEPTNO |
+-----------+---------+
```

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

Tabella dei campioni: DEPT

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

Ora, se si esegue la query:

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

Produzione

```
+-----------+---------+---------+--------------+
|    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  |
+-----------+---------+---------+--------------+
```

Qui le colonne che non corrispondono sono state mantenute NULL.

antijoin

Un antijoin restituisce righe dal lato sinistro del predicato per il quale non ci sono righe corrispondenti sul lato destro del predicato. Restituisce le righe che non riescono a far corrispondere (NOT IN) la sottoquery sul lato destro.

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

Ecco un esempio di Anti Join tra due tabelle:

Tabella di esempio: IMPIEGATO

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

Tabella dei campioni: DEPT

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

Ora, se si esegue la query:

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

Produzione:

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

L'output mostra che solo le righe della tabella EMPLOYEE, di cui DEPTNO non erano presenti nella tabella DEPT.

semijoin

Una query semijoin può essere utilizzata, ad esempio, per trovare tutti i reparti con almeno un dipendente il cui stipendio supera 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;
```

Questo è più efficiente delle alternative complete di join, poiché l'unione interna con i dipendenti e la clausola where che specifica che il salario deve essere maggiore di 2500 potrebbe restituire lo stesso dipartimento numerose volte. Diciamo che se i vigili del fuoco hanno `n` dipendenti con salario 3000, `select * from departments, employees` con il necessario join su id e la nostra clausola where restituirebbe i vigili del fuoco `n` volte.

ADERIRE

L'operazione `JOIN` esegue un join tra due tabelle, escludendo le righe non corrispondenti dalla prima tabella. Da Oracle 9i in avanti, `JOIN` è equivalente in funzione a `INNER JOIN` . Questa operazione richiede una clausola di join esplicita, a differenza degli operatori `CROSS JOIN` e `NATURAL JOIN` .

Esempio:

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

Documentazione Oracle:

- 10g
- 11g
- 12g

JOIN NATURALE

NATURAL JOIN non richiede condizioni di unione esplosive; ne crea uno basato su tutti i campi con lo stesso nome nelle tabelle unite.

```
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');
```

Il join verrà eseguito sui campi ID e DESCR, comuni a entrambe le tabelle:

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

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

Le colonne con nomi diversi non verranno utilizzate nella condizione JOIN:

```
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
```

Se le tabelle unite non hanno colonne comuni, verrà eseguito un JOIN senza condizioni:

```
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
```

Leggi SI UNISCE online: https://riptutorial.com/it/oracle/topic/4192/si-unisce

Capitolo 28: Sicurezza delle applicazioni reali

introduzione

Oracle Real Application Security è stato introdotto in Oracle 12c. Riassume molti argomenti di sicurezza come User-Role-Model, Access Control, Application vs. Database, End-User-Security o Row-and Column Level Security

Examples

Applicazione

Per associare un'applicazione a qualcosa nel database ci sono tre parti principali:

Privilegio dell'applicazione: un privilegio dell'applicazione descrive i privilegi come `SELECT` , `INSERT` , `UPDATE` , `DELETE` , ... I privilegi dell'applicazione possono essere riepilogati come un privilegio aggregato.

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

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

Utente dell'applicazione:

Utente dell'applicazione semplice:

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

Utente dell'applicazione di accesso diretto:

```
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;
```

(opzionale:)

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

Ruolo dell'applicazione:

Ruolo dell'applicazione regolare:

```
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;
```

Ruolo dell'applicazione dinamica: (viene abilitato dinamico in base allo stato di autenaticazione)

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

Ruoli dell'applicazione predefiniti:

Regolare:

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

Dinamica: (dipende dallo stato di autenticazione dell'utente dell'applicazione)

- `DBMS_AUTH` : (accesso diretto o altro metodo di autenticazione del database)
- `EXTERNAL_DBMS_AUTH` : (accesso diretto o altro metodo di autenticazione del database e utente esterno)
- `DBMS_PASSWD` : (accesso diretto con password)
- `MIDTIER_AUTH` : (autenticazione tramite l'applicazione di livello intermedio)
- `XSAUTHENTICATED` : (applicazione diretta o di livello intermedio)
- `XSSWITCH` : (utente passato da utente proxy a utente dell'applicazione)

Leggi Sicurezza delle applicazioni reali online:
https://riptutorial.com/it/oracle/topic/10864/sicurezza-delle-applicazioni-reali

Capitolo 29: SQL dinamico

introduzione

Dynamic SQL consente di assemblare un codice di query SQL nel runtime. Questa tecnica ha alcuni svantaggi e deve essere usata con molta attenzione. Allo stesso tempo, consente di implementare una logica più complessa. PL / SQL richiede che tutti gli oggetti, utilizzati nel codice, debbano esistere e che siano validi al momento della compilazione. Ecco perché non è possibile eseguire istruzioni DDL direttamente in PL / SQL, ma SQL dinamico consente di farlo.

Osservazioni

Alcune importanti osservazioni:

1. Non utilizzare mai concatenazioni di stringhe per aggiungere valori alla query, utilizzare invece i parametri. Questo è sbagliato:

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

   E questo è giusto:

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

   Ci sono due ragioni per questo. Il primo è la sicurezza. La concatenazione di stringhe consente di eseguire l'iniezione SQL. Nella query seguente, se una variabile conterrà il valore `1 or 1 = 1` , l'istruzione `UPDATE` aggiornerà tutte le righe nella tabella:

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

   La seconda ragione è la prestazione. Oracle analizzerà la query senza parametri ogni volta che viene eseguita, mentre la query con parametro verrà analizzata solo una volta nella sessione.

2. Si noti che quando il motore di database esegue un'istruzione DDL, esegue prima il commit implicito.

Examples

Seleziona il valore con SQL dinamico

Supponiamo che un utente desideri selezionare i dati da tabelle diverse. Una tabella è specificata dall'utente.

```
 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;
```

Chiama questa funzione come al solito:

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

Tabella da testare:

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

Inserisci valori in SQL dinamico

L'esempio seguente inserisce il valore nella tabella dell'esempio precedente:

```
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;
/
```

Aggiorna valori in SQL dinamico

Aggiorniamo la tabella dal primo esempio:

```
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;
/
```

Esegui la dichiarazione DDL

Questo codice crea la tabella:

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

Esegui blocco anonimo

Puoi eseguire un blocco anonimo. Questo esempio mostra anche come restituire il valore dall'SQL dinamico:

```
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;
/
```

Leggi SQL dinamico online: https://riptutorial.com/it/oracle/topic/10905/sql-dinamico

Capitolo 30: suggerimenti

Parametri

parametri	Dettagli
Grado di parallelismo (DOP)	È il numero di connessioni / processi paralleli a cui si desidera aprire la query. Di solito è 2, 4, 8, 16 così via.
Nome tabella	Il nome della tabella su cui verrà applicato il suggerimento parallelo.

Examples

Suggerimento parallelo

I suggerimenti paralleli a livello di istruzione sono i più semplici:

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

I suggerimenti paralleli a livello di oggetto danno più controllo, ma sono più inclini agli errori; gli sviluppatori spesso dimenticano di usare l'alias al posto del nome dell'oggetto o dimenticano di includere alcuni oggetti.

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

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

Diciamo che una query impiega 100 secondi per eseguire senza utilizzare il suggerimento parallelo. Se cambiamo DOP a 2 per la stessa query, *idealmente* la stessa query con hint parallelo richiederà 50 secondi. Allo stesso modo usare DOP come 4 richiederà 25 secondi.

In pratica, l'esecuzione parallela dipende da molti altri fattori e non si scala linearmente. Ciò è particolarmente vero per i piccoli tempi di esecuzione in cui il sovraccarico parallelo può essere maggiore dei guadagni derivanti dall'esecuzione in più server paralleli.

USE_NL

Usa cicli annidati.

Uso: `use_nl(AB)`

Questo suggerimento chiederà al motore di utilizzare il metodo loop annidato per unire le tabelle A e B. Questo è il confronto riga per riga. Il suggerimento non forza l'ordine del join, richiede solo NL.

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

APPENDICE SUGGERIMENTO

"Usa il metodo DIRECT PATH per inserire nuove righe".

Il suggerimento `APPEND` indica al motore di utilizzare il carico del percorso diretto . Ciò significa che il motore non utilizzerà un inserto convenzionale utilizzando strutture di memoria e serrature standard, ma scriverà direttamente nel tablespace i dati. Crea sempre nuovi blocchi che vengono aggiunti al segmento della tabella. Questo sarà più veloce, ma presenta alcuni limiti:

- Non è possibile leggere dalla tabella aggiunta alla stessa sessione finché non si impegna o si esegue il rollback della transazione.
- Se ci sono trigger definiti nella tabella, Oracle non utilizzerà il percorso diretto (è una storia diversa per i carichi sqlldr).
- altri

Esempio.

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

USE_HASH

Indica al motore di utilizzare il metodo hash per unire le tabelle nell'argomento.

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

Come spiegato in molti punti , "in un join Hash, Oracle accede a una tabella (in genere il più piccolo dei risultati uniti) e crea una tabella hash sulla chiave join in memoria, quindi esegue la scansione dell'altra tabella nel join (in genere il più grande uno) e controlla la tabella hash per le corrispondenze. "

È preferibile rispetto al metodo Nested Loops quando le tabelle sono grandi, non ci sono indici disponibili, ecc.

Nota : il suggerimento non forza l'ordine del join, richiede solo il metodo HASH JOIN.

Esempio di utilizzo:

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

PIENO

L'hint FULL indica a Oracle di eseguire una scansione completa della tabella su una tabella specificata, non importa se è possibile utilizzare un indice.

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

Senza suggerimenti, viene utilizzato l'indice:

```
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 |
-------------------------------------------------------------------------------
```

L'hint FULL forza una scansione 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 |
-----------------------------------------------------------------------------------
```

Cache dei risultati

Oracle (*11g e versioni successive*) consente alle query SQL di essere memorizzate nella cache nell'SGA e riutilizzate per migliorare le prestazioni. Interroga i dati dalla cache anziché dal database. L'esecuzione successiva della stessa query è più veloce perché ora i dati vengono estratti dalla cache.

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

Produzione -

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

Elapsed: 00:00:02.20
```

Se ora eseguo di nuovo la stessa query, il tempo di esecuzione si ridurrà poiché i dati vengono ora recuperati dalla cache che è stata impostata durante la prima esecuzione.

Produzione -

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

Elapsed: 00:00:00.10
```

Notare come il tempo trascorso si è ridotto da **2,20 secondi** a **0,10 secondi** .

> Cache dei risultati conserva la cache finché i dati nel database non vengono aggiornati / modificati / eliminati. Qualsiasi modifica rilascerà la cache.

Leggi suggerimenti online: https://riptutorial.com/it/oracle/topic/1490/suggerimenti

Capitolo 31: Tavolo DUAL

Osservazioni

`DUAL` tabella `DUAL` ha una colonna `DUMMY` , definita come `VARCHAR2(1)` e una sola riga con un valore `x` .

`DUAL` tabella `DUAL` viene creata automaticamente nello schema `SYS` quando viene creato il database. Puoi accedervi da qualsiasi schema.

Non è possibile modificare la tabella `DUAL` .

È possibile utilizzare la tabella `DUAL` per chiamare qualsiasi funzione dall'istruzione SQL. È utile perché ha solo una riga e oracle optimizer sa tutto su di esso.

Examples

L'esempio seguente restituisce la data e l'ora del sistema operativo corrente

```
select sysdate from dual
```

L'esempio seguente genera numeri tra start_value e end_value

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

Leggi Tavolo DUAL online: https://riptutorial.com/it/oracle/topic/7328/tavolo-dual

Capitolo 32: Transazioni autonome

Osservazioni

I casi d'uso tipici per le transazioni autonome sono.

1. Per costruire qualsiasi tipo di framework di registrazione come il framework di registrazione degli errori spiegato nell'esempio precedente.
2. Per il controllo delle operazioni DML nei trigger su tabelle indipendentemente dallo stato finale della transazione (COMMIT o ROLLBACK).

Examples

Utilizzo della transazione autonoma per errori di registrazione

La seguente procedura è generica che verrà utilizzata per registrare tutti gli errori in un'applicazione in una tabella di log degli errori comune.

```
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;
```

Il seguente blocco PLSQL anonimo mostra come chiamare la procedura 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
```

Leggi Transazioni autonome online: https://riptutorial.com/it/oracle/topic/6103/transazioni-autonome

Capitolo 33: vincoli

Examples

Aggiorna le chiavi esterne con un nuovo valore in Oracle

Supponiamo di avere una tabella e di voler modificare uno di questi ID primario della tabella. puoi usare il seguente scrpit. l'ID principale qui è "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;
```

Disattiva tutte le chiavi esterne correlate in Oracle

Supponiamo che tu abbia la tabella T1 e che abbia una relazione con molte tabelle e che il suo nome di vincolo di chiave primaria sia "pk_t1" che vuoi disabilitare queste chiavi esterne che puoi usare:

```
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;
```

Leggi vincoli online: https://riptutorial.com/it/oracle/topic/6040/vincoli

Titoli di coda

S. No	Capitoli	Contributors
1	Iniziare con il database Oracle	Community, J. Chomel, Jeffrey Kemp, Jon Ericson, Kevin Montrose, Mark Stewart, Sanjay Radadiya, Steven Feuerstein, tonirush
2	Aggiorna con Joins	mathguy
3	Blocco anonimo di PL / SQL	Jon Heller, Skynet, Zohar Elkayam
4	Creare un contesto	Jeffrey Kemp
5	Data Pump	Vidya Thotangare
6	Date	carlosb, MT0, Roman, tonirush
7	Delimitare parole chiave o caratteri speciali	dev
8	Diversi modi per aggiornare i record	nimour pristou, Nogueira Jr, SriniV
9	Divisione di stringhe delimitate	Arkadiusz Łukasiewicz, MT0
10	Dizionario dei dati	Mark Stewart, Pancho, Slava Babin
11	Factoring ricorsivo sotto-query utilizzando la clausola WITH (espressioni di tabella comuni AKA)	B Samedi, MT0
12	Funzioni della finestra	Leigh Riffel
13	Funzioni statistiche	Evgeniy K., Matas Vaitkevicius, ppeterka, Pranav Shah
14	Gestire valori NULL	Dalex, JeromeFr
15	indici	smshafiqulislam

16	Lavorare con le date	David Aldridge, Florin Ghita, Jeffrey Kemp, Mark Stewart, tonirush, zygimantus
17	Limitazione delle righe restituite da una query (impaginazione)	Ahmed Mohamed, Martin Schapendonk, Matas Vaitkevicius, Sanjay Radadiya, tonirush, trincot
18	Link al database	carlosb, Daniel Langemann, g00dy, kasi
19	MAF Oracle	Anand Raj
20	Manipolazione delle stringhe	carlosb, Eric B., Florin Ghita, Francesco Serra, J. Chomel, J.Hudler, Jeffrey Kemp, Mark Stewart, SriniV, Thunder, walen, zhliu03
21	Oracle Advanced Queuing (AQ)	Jon Theriault
22	Partizionamento delle tabelle	BobC, carlosb, ivanzg, JeromeFr, Kamil Islamov, Stephen Leppik, tonirush
23	query di livello	Sanjay Radadiya, TechEnthusiast
24	Recupero gerarchico con Oracle Database 12C	Muntasir, Vahid
25	Registrazione errori	zygimantus
26	sequenze	Pranav Shah, SriniV
27	SI UNISCE	Aleksej, B Samedi, Bakhtiar Hasan, Daniel Langemann, Erkan Haspulat, Pranav Shah, Robin James, SriniV, Sumner Evans
28	Sicurezza delle applicazioni reali	Ben H
29	SQL dinamico	Dmitry
30	suggerimenti	Aleksej, Florin Ghita, Jon Heller, Mark Stewart, Pirate X
31	Tavolo DUAL	Slava Babin
32	Transazioni autonome	phonetic_man
33	vincoli	SSD