eBook Gratuit

APPRENEZ

# Oracle Database

eBook gratuit non affilié créé à partir des **contributeurs de Stack Overflow.**

#oracle

# Table des matières

# À propos

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

# Chapitre 1: Démarrer avec Oracle Database

## Remarques

Oracle est un système de gestion de base de données relationnelle (SGBDR) conçu à l'origine par Larry Ellison, Bob Miner et Ed Oates à la fin des années 70. Il était destiné à être compatible avec IBM System R.

## Versions

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

## Examples

### Bonjour le monde

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

Dans le style SQL d'Oracle, "dual n'est qu'une table de convienence" . Il était à l' origine destiné à

doubler les lignes via un JOIN, mais contient désormais une ligne avec une valeur `DUMMY` de «X».

### Bonjour le monde! de table

## Créer un tableau simple

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

## Insérer des valeurs (vous pouvez omettre les colonnes cibles si vous fournissez des valeurs pour toutes les colonnes)

```
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');
```

## N'oubliez pas de vous engager, car Oracle utilise des *transactions*

```
commit;
```

## Sélectionnez vos données:

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

### Requête SQL

Énumérez les employés gagnant plus de 50000 $ nés ce siècle. Indiquez leur nom, leur date de naissance et leur salaire, classés alphabétiquement par nom.

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

Indiquez le nombre d'employés dans chaque département avec au moins 5 employés. Énumérez d'abord les plus grands ministères.

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

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

## Bonjour tout le monde de 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;
```

Lire Démarrer avec Oracle Database en ligne: https://riptutorial.com/fr/oracle/topic/558/demarrer-avec-oracle-database

# Chapitre 2: Affacturage récursif des sous-requêtes à l'aide de la clause WITH (AKA Common Table Expressions)

## Remarques

La factorisation récursive des sous-requêtes est disponible dans Oracle 11g R2.

## Examples

### Un générateur d'entier simple

**Requête** :

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

**Sortie** :

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

### Fractionnement d'une chaîne délimitée

**Données d'échantillon** :

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

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

**Requête** :

```
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;
```

**Sortie** :

```
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
```

Lire Affacturage récursif des sous-requêtes à l'aide de la clause WITH (AKA Common Table Expressions) en ligne: https://riptutorial.com/fr/oracle/topic/3506/affacturage-recursif-des-sous-requetes-a-l-aide-de-la-clause-with--aka-common-table-expressions-

# Chapitre 3: Astuces

## Paramètres

| Paramètres | Détails |
|---|---|
| Degré de parallélisme (DOP) | C'est le nombre de connexions / processus parallèles que vous souhaitez que votre requête ouvre. C'est généralement 2, 4, 8, 16 etc. |
| Nom de la table | Le nom de la table sur laquelle l'indicateur parallèle sera appliqué. |

## Examples

### Conseil parallèle

Les indicateurs parallèles au niveau des instructions sont les plus simples:

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

Les indications parallèles au niveau de l'objet donnent plus de contrôle mais sont plus sujettes aux erreurs. Les développeurs oublient souvent d'utiliser l'alias au lieu du nom de l'objet, ou oublient d'inclure des objets.

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

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

Disons qu'une requête prend 100 secondes pour s'exécuter sans utiliser de conseil parallèle. Si nous modifions DOP à 2 pour la même requête, la même requête avec indication parallèle prendra *idéalement* 50 secondes. De même, l'utilisation de DOP 4 prend 25 secondes.

En pratique, l'exécution parallèle dépend de nombreux autres facteurs et ne s'adapte pas de manière linéaire. Cela est particulièrement vrai pour les petits temps d'exécution où la charge parallèle peut être supérieure aux gains de l'exécution sur plusieurs serveurs parallèles.

### USE_NL

Utilisez des boucles imbriquées.

Utilisation: `use_nl(AB)`

Cet indice demande au moteur d'utiliser la méthode de boucle imbriquée pour joindre les tables A et B. Il s'agit d'une comparaison ligne par ligne. L'indice ne force pas l'ordre de la jointure, demande simplement NL.

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

## APPEND HINT

"Utiliser la méthode DIRECT PATH pour insérer de nouvelles lignes".

L'indicateur `APPEND` indique au moteur d'utiliser le chargement direct du chemin . Cela signifie que le moteur n'utilisera pas un insert conventionnel utilisant des structures de mémoire et des verrous standard, mais écrira directement les données dans le tablespace. Crée toujours de nouveaux blocs qui sont ajoutés au segment de la table. Ce sera plus rapide, mais a quelques limitations:

- Vous ne pouvez pas lire la table que vous avez ajoutée dans la même session jusqu'à ce que vous validiez ou annuliez la transaction.
- S'il existe des déclencheurs définis sur la table, Oracle n'utilisera pas le chemin direct (c'est une autre histoire pour les chargements sqlldr).
- autres

Exemple.

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

## USE_HASH

Indique au moteur d'utiliser la méthode de hachage pour joindre des tables dans l'argument.

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

Comme expliqué dans de nombreux endroits , "dans une jointure HASH, Oracle accède à une table (généralement le plus petit des résultats joints) et construit une table de hachage en mémoire sur la clé de jointure. Elle analyse ensuite l'autre table de la jointure un) et sonde la table de hachage pour y rechercher les correspondances. "

Il est préférable d'utiliser la méthode Nested Loops lorsque les tables sont grandes, qu'aucun index n'est disponible, etc.

**Remarque** : L'indicateur ne force pas l'ordre de la jointure, demande simplement la méthode HASH JOIN.

Exemple d'utilisation:

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

## PLEIN

L'indication FULL indique à Oracle d'effectuer une analyse complète de la table sur une table spécifique, peu importe si un index peut être utilisé.

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

Sans indices, l'index est utilisé:

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

FULL Hint force une analyse complète:

```
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 de résultat

Oracle ( *11g et plus* ) permet aux requêtes SQL d'être mises en cache dans le SGA et réutilisées pour améliorer les performances. Il interroge les données du cache plutôt que de la base de données. L'exécution ultérieure de la même requête est plus rapide car les données sont maintenant extraites du cache.

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

Sortie -

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

Elapsed: 00:00:02.20
```

Si j'exécute à nouveau la même requête maintenant, le temps d'exécution diminuera car les données sont désormais extraites du cache qui a été défini lors de la première exécution.

Sortie -

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

Elapsed: 00:00:00.10
```

Notez que le temps écoulé est passé de **2,20 secondes** à **0,10 seconde** .

> Le cache de résultats contient le cache jusqu'à ce que les données de la base de données soient mises à jour / modifiées / supprimées. Toute modification libère le cache.

Lire Astuces en ligne: https://riptutorial.com/fr/oracle/topic/1490/astuces

# Chapitre 4: Bloc PL / SQL anonyme

## Remarques

Comme ils ne sont pas nommés, les blocs anonymes ne peuvent pas être référencés par d'autres unités de programme.

## Examples

### Un exemple de bloc anonyme

```
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;
/
```

Lire Bloc PL / SQL anonyme en ligne: https://riptutorial.com/fr/oracle/topic/6451/bloc-pl---sql-anonyme

# Chapitre 5: contraintes

## Examples

### Mettre à jour les clés étrangères avec une nouvelle valeur dans Oracle

Supposons que vous ayez une table et que vous souhaitiez en changer un. vous pouvez utiliser le scrpit suivant. L'ID primaire est "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;
```

### Désactiver toutes les clés étrangères associées dans Oracle

Supposons que vous ayez la table T1 et qu'elle soit en relation avec de nombreuses tables et que son nom de contrainte de clé primaire soit "pk_t1", vous voulez désactiver ces clés étrangères que vous pouvez utiliser:

```
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;
```

Lire contraintes en ligne: https://riptutorial.com/fr/oracle/topic/6040/contraintes

# Chapitre 6: Créer un contexte

## Syntaxe

- CREATE [OU REPLACE] CONTEXT namespace UTILISATION du package [schema.];
- CREATE [OU REPLACE] CONTEXT namespace UTILISATION du package [schema.] INITIALIZED EXTERNALLY;
- CREATE [OU REPLACE] CONTEXT namespace UTILISATION du package [schema.] INITIALIZED GLOBALLY;
- CREATE [OU REPLACE] CONTEXT namespace UTILISATION du package [schema.] ACCESS GLOBALLY;

## Paramètres

| Paramètre | Détails |
|---|---|
| `OR REPLACE` | Redéfinir un espace de nom de contexte existant |
| espace de noms | Nom du contexte - c'est l'espace de noms pour les appels à `SYS_CONTEXT` |
| schéma | Propriétaire du forfait |
| paquet | Package de base de données qui définit ou réinitialise les attributs de contexte. Remarque: le package de base de données ne doit pas nécessairement exister pour créer le contexte. |
| `INITIALIZED` | Spécifiez une entité autre que Oracle Database qui peut définir le contexte. |
| `EXTERNALLY` | Autorise l'interface OCI à initialiser le contexte. |
| `GLOBALLY` | Autoriser l'annuaire LDAP à initialiser le contexte lors de l'établissement de la session. |
| `ACCESSED GLOBALLY` | Autoriser le contexte à être accessible dans toute l'instance - plusieurs sessions peuvent partager les valeurs d'attribut tant qu'elles ont le même ID client. |

## Remarques

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

## Examples

## Créer un contexte

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

Cela crée un contexte qui ne peut être défini que par les routines du package de base de données `my_pkg` , par exemple:

```
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;
```

Maintenant, si une session fait cela:

```
my_pkg.set_ctx;
```

Il peut maintenant récupérer la valeur de la clé:

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

Value
```

Lire Créer un contexte en ligne: https://riptutorial.com/fr/oracle/topic/2088/creer-un-contexte

# Chapitre 7: Délimitation de mots-clés ou de caractères spéciaux

## Examples

### Délimiter le nom de la table ou de la colonne avec des caractères spéciaux

Sélectionnez * à partir de l'adresse du cabinet;

Sélectionnez * dans "adresse du cabinet";

### Délimitation du nom de la table ou de la colonne, qui est également un mot réservé

Supposons que vous ayez une table nommée table ou que vous souhaitiez créer une table avec un nom qui est également un mot-clé. Vous devez inclure la table des noms dans une paire de guillemets doubles "table"

Sélectionnez * dans la table; La requête ci-dessus échouera avec une erreur de syntaxe, où la requête s'exécutera comme ci-dessous.

Sélectionnez * dans "table"

Lire Délimitation de mots-clés ou de caractères spéciaux en ligne: https://riptutorial.com/fr/oracle/topic/6553/delimitation-de-mots-cles-ou-de-caracteres-speciaux

# Chapitre 8: Dictionnaire de données

## Remarques

Les vues du dictionnaire de données, également appelées vues de catalogue, vous permettent de surveiller l'état de la base de données en temps réel:

Les vues préfixées par `USER_` , `ALL_` et `DBA_` , affichent des informations sur les objets de schéma que vous `USER_` ( `USER_` ), accessibles par vous ( `ALL_` ) ou accessibles par un utilisateur disposant du privilège SYSDBA ( `DBA_` ). Par exemple, la vue `ALL_TABLES` affiche toutes les tables sur lesquelles vous disposez de privilèges.

Les vues `V$` affichent des informations relatives aux performances.

Les vues `_PRIVS` affichent des informations de privilège pour différentes combinaisons d'utilisateurs, de rôles et d'objets.

Documentation Oracle: Affichages de catalogue / Vues du dictionnaire de données

## Examples

### Source de texte des objets stockés

`USER_SOURCE` décrit la source de texte des objets stockés appartenant à l'utilisateur actuel. Cette vue n'affiche pas la colonne `OWNER` .

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

`ALL_SOURCE` décrit la source de texte des objets stockés accessibles à l'utilisateur actuel.

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

`DBA_SOURCE` décrit la source de texte de tous les objets stockés dans la base de données.

```
select * from dba_source
```

### Obtenir la liste de toutes les tables dans Oracle

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

`ALL_TAB_COLUMNS` décrit les colonnes des tables, des vues et des clusters accessibles à l'utilisateur actuel. `COLS` est un synonyme de `USER_TAB_COLUMNS` .

```
select *
```

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

## Informations de privilège

Tous les rôles accordés à l'utilisateur.

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

Privilèges accordés à l'utilisateur:

1. privilèges système

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

2. subventions d'objet

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

**Autorisations accordées aux rôles.**

Rôles accordés à d'autres rôles.

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

1. privilèges système

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

2. subventions d'objet

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

## Version Oracle

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

### Décrit tous les objets de la base de données.

```
select *
from dba_objects
```

### Pour afficher toutes les vues du dictionnaire de données auxquelles vous avez accès

```
select * from dict
```

Lire Dictionnaire de données en ligne: https://riptutorial.com/fr/oracle/topic/7347/dictionnaire-de-donnees

# Chapitre 9: Différentes façons de mettre à jour les enregistrements

## Syntaxe

- UPDATE nom-table [[AS] nom de corrélation] SET nom-colonne = valeur [, nom-colonne = valeur}] * [clause WHERE]
- UPDATE nom-table SET nom-colonne = Valeur [, nom-colonne = valeur] * OENT CURRENT DE

## Examples

### Mettre à jour la syntaxe avec un exemple

**Mise à jour normale**

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

### Mise à jour à l'aide de la vue en ligne

**Utilisation de la vue en ligne (si elle est considérée comme pouvant être mise à jour par Oracle)**

**Remarque** : Si vous rencontrez une erreur de ligne non préservée, ajoutez un index pour résoudre le même problème afin de le rendre compatible avec les mises à jour.

```
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;
```

## Mettre à jour à l'aide de la fusion

### Utiliser la fusion

```
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';
```

## Fusionner avec des exemples de données

```
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);
```

Lire Différentes façons de mettre à jour les enregistrements en ligne:
https://riptutorial.com/fr/oracle/topic/4193/differentes-facons-de-mettre-a-jour-les-enregistrements

# Chapitre 10: Erreur de journalisation

## Examples

### Erreur lors de l'écriture dans la base de données

Créez la table des erreurs Oracle ERR $ _EXAMPLE pour la table EXEMPLE existante:

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

Opération d'écriture avec SQL:

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

Lire Erreur de journalisation en ligne: https://riptutorial.com/fr/oracle/topic/3505/erreur-de-journalisation

# Chapitre 11: Fonctions de fenêtre

## Syntaxe

- Ratio_To_Report (expr) OVER (query_partition_clause)

## Examples

### Ratio_To_Report

Indique le rapport entre la valeur actuelle des lignes et toutes les valeurs de la fenêtre.

```
--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
```

Lire Fonctions de fenêtre en ligne: https://riptutorial.com/fr/oracle/topic/6669/fonctions-de-fenetre

# Chapitre 12: Fonctions statistiques

## Examples

### Calcul de la médiane d'un ensemble de valeurs

La fonction MEDIAN depuis Oracle 10g est une fonction d'agrégation facile à utiliser:

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

Il retourne la médiane des valeurs

Fonctionne également sur les valeurs `DATETIME` .

> Le résultat de MEDIAN est calculé en classant d'abord les lignes. En utilisant N comme nombre de lignes dans le groupe, Oracle calcule le numéro de ligne (RN) concerné avec la formule RN = (1 + (0.5 * (N-1)). Le résultat final de la fonction d'agrégation est calculé par linéaire interpolation entre les valeurs des lignes aux numéros de ligne CRN = CEILING (RN) et FRN = FLOOR (RN).

Depuis Oracle 9i, vous pouvez utiliser PERCENTILE_CONT, qui fonctionne de la même manière que la fonction MEDIAN avec une valeur par défaut de 0,5

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

### VARIANCE

La variance mesure à quel point un nombre donné est étalé de sa moyenne. Du point de vue pratique, c'est la distance au carré de sa moyenne (centre) - plus le nombre est grand, plus le point est éloigné.

L'exemple suivant renverrait la variance des valeurs de salaire

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

### STDDEV

STDDEV renvoie l'écart type d'échantillon de expr, un ensemble de nombres. Vous pouvez l'utiliser comme fonction agrégée et analytique. Il diffère de STDDEV_SAMP dans le sens où STDDEV renvoie zéro lorsqu'il ne possède qu'une ligne de données en entrée, alors que STDDEV_SAMP renvoie null.

Oracle Database calcule l'écart type en tant que racine carrée de la variance définie pour la

fonction d'agrégation VARIANCE.

Cette fonction prend comme argument tout type de données numérique ou tout type de données non numérique pouvant être implicitement converti en un type de données numérique. La fonction renvoie le même type de données que le type de données numérique de l'argument.

Si vous spécifiez DISTINCT, vous ne pouvez spécifier que la requête_partition_clause du paramètre analytic_clause. Les commandes order_by_clause et windowing_clause ne sont pas autorisées.

L'exemple suivant renvoie l'écart type des salaires dans la table exemple **hr.employees** :

Où h est Schéma et employés est un nom de table.

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

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

La requête dans l'exemple suivant renvoie l'écart-type cumulé des salaires dans le service 80 dans l'exemple de table hr.employees, ordonné par 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
```

Lire Fonctions statistiques en ligne: https://riptutorial.com/fr/oracle/topic/2283/fonctions-statistiques

# Chapitre 13: Fractionnement de chaînes délimitées

## Examples

### Fractionnement de chaînes à l'aide d'une clause d'affacturage sous-requête récursive

**Données d'échantillon** :

```
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
```

**Requête** :

```
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;
```

**Sortie** :

```
        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
```

## Fractionnement de chaînes à l'aide d'une fonction PL / SQL

**Fonction 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;
/
```

**Données d'échantillon** :

```
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
```

**Requête** :

```
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
```

**Sortie** :

```
        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
```

## Fractionnement de chaînes à l'aide d'une expression de table corrélée

**Données d'échantillon** :

```
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
```

**Requête** :

```
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;
```

**Sortie** :

```
        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
```

## Fractionnement de chaînes à l'aide d'une requête hiérarchique

**Données d'échantillon** :

```
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
```

**Requête** :

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

**Sortie** :

```
        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
```

## Fractionnement de chaînes à l'aide des expressions XMLTable et FLWOR

Cette solution utilise la fonction `ora:tokenize` XQuery disponible à partir d'Oracle 11.

**Données d'échantillon** :

```
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
```

**Requête** :

```
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;
```

**Sortie** :

```
        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
```

## Fractionnement de chaînes à l'aide de CROSS APPLY (Oracle 12c)

**Données d'échantillon** :

```
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
```

**Requête** :

```
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;
```

**Sortie** :

```
        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
```

## Fractionnement de chaînes délimitées à l'aide de XMLTable

**Données d'échantillon** :

```
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
```

**Requête** :

```
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;
```

*(Remarque: le caractère `#` est ajouté pour faciliter l'extraction des valeurs `NULL` ; il est ensuite supprimé à l'aide de `SUBSTR( item, 2 )` . Si les valeurs `NULL` ne sont pas requises, vous pouvez simplifier la requête et l'omettre.)*

**Sortie** :

```
        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
```

Lire Fractionnement de chaînes délimitées en ligne:
https://riptutorial.com/fr/oracle/topic/1968/fractionnement-de-chaines-delimitees

# Chapitre 14: Gestion des valeurs NULL

## Introduction

Une colonne est NULL lorsqu'elle n'a pas de valeur, quel que soit le type de données de cette colonne. Une colonne ne devrait jamais être comparée à NULL en utilisant cette syntaxe `a = NULL` car le résultat serait INCONNU. Utilisez plutôt `a IS NULL` ou `a IS NOT NULL` condition `a IS NOT NULL` . NULL n'est pas égal à NULL. Pour comparer deux expressions où null peut se produire, utilisez l'une des fonctions décrites ci-dessous. Tous les opérateurs sauf la concaténation renvoient NULL si l'un de leurs opérandes est NULL. Par exemple, le résultat de `3 * NULL + 5` est nul.

## Remarques

NULL ne peut pas apparaître dans les colonnes restreintes par une contrainte PRIMARY KEY ou NOT NULL. (L'exception est une nouvelle contrainte avec la clause NOVALIDATE)

## Examples

### Les colonnes de tout type de données peuvent contenir des valeurs NULL

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

| NUM_COLUMN | VARCHAR2_COLUMN |
|---|---|
| 1 | foo |
| (nul) | (nul) |

### Les chaînes vides sont NULL

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

| UNE | B |
|---|---|
| 1 | (nul) |

### Les opérations contenant NULL sont NULL, sauf la concaténation

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

| 3 * NULL + 5 | 'HELLO' || NULL || 'WORLD' |
|---|---|
| (nul) | Bonjour le monde |

### NVL pour remplacer la valeur nulle

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

| COLUMN_WITH_NULL | COLUMN_WITHOUT_NULL |
|---|---|
| (nul) | N / A |

NVL est utile pour comparer deux valeurs pouvant contenir des valeurs 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 pour obtenir un résultat différent si une valeur est nulle ou non

Si le premier paramètre est NOT NULL, NVL2 renvoie le deuxième paramètre. Sinon, il retournera le troisième.

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

| NVL2 (NULL, 'FOO', 'BAR') | NVL2 (5, «FOO», «BAR») |
|---|---|
| Bar | Foo |

### COALESCE pour retourner la première valeur 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 |

Dans certains cas, l'utilisation de COALESCE avec deux paramètres peut être plus rapide qu'avec NVL lorsque le second paramètre n'est pas une constante. NVL évaluera toujours les deux paramètres. COALESCE s'arrêtera à la première valeur non NULL rencontrée. Cela signifie que si la première valeur est non-NULL, COALESCE sera plus rapide.

Lire Gestion des valeurs NULL en ligne: https://riptutorial.com/fr/oracle/topic/8183/gestion-des-valeurs-null

# Chapitre 15: Index

## Introduction

Ici, je vais expliquer différents indices en utilisant l'exemple, comment l'index augmente les performances des requêtes, comment l'index diminue les performances DML, etc.

## Examples

### index b-tree

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

Par défaut, si nous ne mentionnons rien, oracle crée un index en tant qu'index b-tree. Mais nous devrions savoir quand l'utiliser. L'index B-tree stocke les données sous forme d'arborescence binaire. Comme nous le savons, index est un objet de schéma qui stocke une sorte d'entrée pour chaque valeur de la colonne indexée. Ainsi, chaque fois qu'une recherche est effectuée sur ces colonnes, elle vérifie dans l'index l'emplacement exact de cet enregistrement pour y accéder rapidement. Quelques points sur l'indexation:

- Pour rechercher une entrée dans l'index, une sorte d'algorithme de recherche binaire est utilisé.
- Lorsque **la cardinalité des données est élevée, l'** index **b-tree** est parfait.
- Index rend DML lente, pour chaque enregistrement, il doit y avoir une entrée dans l'index pour la colonne indexée.
- Donc, si ce n'est pas nécessaire, nous devrions éviter de créer un index.

### Index de bitmap

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

- L'index bitmap est utilisé lorsque **la cardinalité des données est faible.**
- Ici, le **genre** a une valeur avec une faible cardinalité. Les valeurs peuvent être masculines, féminines et autres.
- Donc, si nous créons un arbre binaire pour ces 3 valeurs lors de la recherche, cela aura un cheminement inutile.
- Dans les structures bitmap, un tableau à deux dimensions est créé avec une colonne pour chaque ligne de la table indexée. Chaque colonne représente une valeur distincte dans l'index bitmap. Ce tableau à deux dimensions représente chaque valeur de l'index multipliée par le nombre de lignes de la table.
- Au moment de la récupération des lignes, Oracle décompresse le bitmap dans les tampons de données RAM afin de pouvoir effectuer rapidement une analyse pour rechercher les

valeurs correspondantes. Ces valeurs correspondantes sont fournies à Oracle sous la forme d'une liste ID de ligne et ces valeurs ID de ligne peuvent accéder directement aux informations requises.

## Index basé sur la fonction

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


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

- Indicateur basé sur la fonction, créant un index basé sur une fonction.
- Si dans la recherche (clause where), une fonction est fréquemment utilisée, il est préférable de créer un index basé sur cette fonction.
- Ici, dans l'exemple, pour la recherche, la fonction **Upper ()** est utilisée. Il est donc préférable de créer un index en utilisant la fonction supérieure.

Lire Index en ligne: https://riptutorial.com/fr/oracle/topic/9978/index

# Chapitre 16: JOINT

## Examples

### CROSS JOIN

Un `CROSS JOIN` effectue une jointure entre deux tables qui n'utilise pas de clause de jointure explicite et génère le produit cartésien de deux tables. Un produit cartésien signifie que chaque ligne d'une table est combinée avec chaque ligne de la seconde table de la jointure. Par exemple, si `TABLEA` a 20 lignes et que `TABLEB` a 20 lignes, le résultat serait `20*20 = 400` lignes de sortie.

Exemple:

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

Cela peut aussi être écrit comme:

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

Voici un exemple de jointure croisée en SQL entre deux tables:

**Tableau d'échantillons:** TABLEA

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

**Tableau d'échantillons:** TABLEB

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

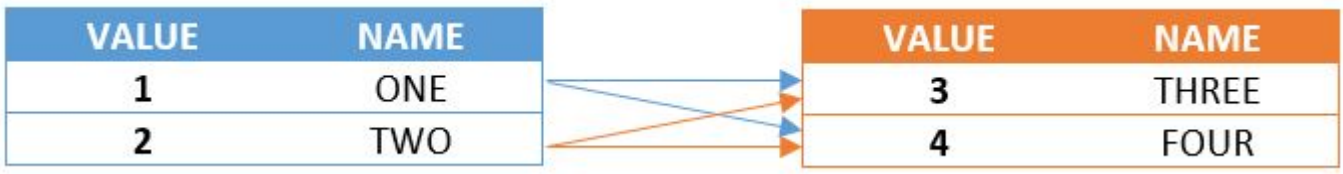
Maintenant, si vous exécutez la requête:

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

**Sortie:**

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

Voici comment les jointures se produisent entre deux tables:




En savoir plus sur Cross Join: documentation Oracle

## JOINTURE INTERNE

Un INNER JOIN est une opération JOIN qui vous permet de spécifier une clause de jointure explicite.

Syntaxe

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

Vous pouvez spécifier la clause de jointure en spécifiant ON avec une expression booléenne.

La portée des expressions dans la clause ON inclut les tables actuelles et toutes les tables des blocs de requête externes au SELECT actuel. Dans l'exemple suivant, la clause ON fait référence aux tables en cours:

```
-- 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
```

## JOINTURE EXTERNE GAUCHE

Un `LEFT OUTER JOIN` effectue une jointure entre deux tables qui nécessite une clause de jointure explicite mais n'exclut pas les lignes sans correspondance de la première table.

Exemple:

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

Même si la syntaxe ANSI est la méthode recommandée , elle rencontrera très probablement une syntaxe héritée. L'utilisation de `(+)` dans une condition détermine le côté de l'équation à considérer comme *externe* .

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

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

Voici un exemple de jointure externe gauche entre deux tables:

**Tableau d'échantillon:** EMPLOYÉ

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

**Tableau d'échantillons:** DEPT

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

Maintenant, si vous exécutez la requête:

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

**Sortie:**

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

## DROIT EXTERIEUR

Un joint `RIGHT OUTER JOIN` effectue une jointure entre deux tables qui nécessite une clause de jointure explicite mais n'exclut pas les lignes sans correspondance de la seconde table.

Exemple:

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

Comme les lignes sans correspondance de `SCOTT.DEPT` sont incluses, mais que les lignes sans correspondance de `SCOTT.EMP` ne le sont pas, ce qui précède est équivalent à l'instruction suivante utilisant `LEFT OUTER JOIN` .

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

Voici un exemple de jointure externe droite entre deux tables:

**Tableau d'échantillon:** EMPLOYÉ

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

**Tableau d'échantillons:** DEPT

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

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

Maintenant, si vous exécutez la requête:

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

**Sortie:**

```
+-----------+---------+---------+--------------+
|    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'équivalent de la syntaxe Oracle (+) pour la requête est:

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

## FULL OUTER JOIN

Un `FULL OUTER JOIN` effectue une jointure entre deux tables qui nécessite une clause de jointure explicite mais n'exclut pas les lignes sans correspondance dans les deux tables. En d'autres termes, il renvoie toutes les lignes de chaque table.

Exemple:

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

Voici un exemple de jointure externe complète entre deux tables:

**Tableau d'échantillon:** EMPLOYÉ

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

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

**Tableau d'échantillons:** DEPT

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

Maintenant, si vous exécutez la requête:

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

**Sortie**

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

Ici, les colonnes qui ne correspondent pas ont été conservées NULL.

## ANTIJOIN

Un antijoin renvoie des lignes du côté gauche du prédicat pour lesquelles il n'y a pas de lignes correspondantes du côté droit du prédicat. Il renvoie les lignes qui ne correspondent pas (PAS à) à la sous-requête du côté droit.

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

Voici un exemple d'Anti Join entre deux tables:

**Tableau d'échantillon:** EMPLOYÉ

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

**Tableau d'échantillons:** DEPT

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

Maintenant, si vous exécutez la requête:

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

**Sortie:**

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

La sortie montre que seules les lignes de la table EMPLOYEE, dont DEPTNO, n'étaient pas présentes dans la table DEPT.

## SEMIJOIN

Une requête de semi-jointure peut être utilisée, par exemple, pour trouver tous les départements avec au moins un employé dont le salaire dépasse 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;
```

Ceci est plus efficace que les alternatives de jointure intégrale, étant donné que la jointure interne sur les employés, puis la clause précisant que le salaire doit être supérieur à 2500, peuvent retourner le même service à plusieurs reprises. Dites si le service des incendies a `n` employés tous avec le salaire 3000, `select * from departments, employees` avec les jointures nécessaires sur nos identifiants et notre clause where renverrait le service des pompiers `n` fois.

## JOINDRE

L'opération `JOIN` effectue une jointure entre deux tables, à l'exclusion des lignes sans correspondance de la première table. A partir d'Oracle 9i, la fonction `JOIN` est équivalente en fonction à `INNER JOIN` . Cette opération nécessite une clause de jointure explicite, par opposition aux opérateurs `CROSS JOIN` et `NATURAL JOIN` .

Exemple:

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

Documentation Oracle:

- 10g
- 11g
- 12g

## NATURAL JOIN

NATURAL JOIN ne nécessite aucune condition de jointure explicite; il en construit un basé sur tous les champs portant le même nom dans les tables jointes.

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

La jointure se fera sur les champs ID et DESCR, communs aux deux tables:

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

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

Les colonnes portant des noms différents ne seront pas utilisées dans la condition 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
```

Si les tables jointes n'ont pas de colonnes communes, un JOIN sans conditions sera effectué:

```
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
```

Lire JOINT en ligne: https://riptutorial.com/fr/oracle/topic/4192/joint

# Chapitre 17: Liens de base de données

## Examples

### Création d'un lien de base de données

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

La base de données distante sera alors accessible de la manière suivante:

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

Pour tester une connexion de liaison de base de données sans avoir à connaître aucun des noms d'objet de la base de données liée, utilisez la requête suivante:

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

Pour spécifier explicitement un domaine pour le service de base de données lié, le nom de domaine est ajouté à l'instruction `USING` . Par exemple:

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

Si aucun nom de domaine n'est explicitement spécifié, Oracle utilise le domaine de la base de données dans laquelle le lien est créé.

Documentation Oracle pour la création de liens de bases de données:

- 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

### Créer un lien de base de données

Supposons que nous ayons deux bases de données "ORA1" et "ORA2". Nous pouvons accéder aux objets de "ORA2" à partir de la base de données "ORA1" en utilisant un lien de base de données.

Conditions préalables: Pour créer un lien de base de données privée, vous devez disposer d'un privilège `CREATE DATABASE LINK` . Pour créer un lien de base de données privée, vous devez disposer d'un privilège `CREATE PUBLIC DATABASE LINK` .

* Oracle Net doit être présent sur les deux instances.

Comment créer un lien de base de données:

De ORA1:

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

Lien de base de données créé.

Maintenant que le lien de la base de données est configuré, nous pouvons le prouver en exécutant les opérations suivantes depuis ORA1:

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

Vous pouvez également accéder aux objets de base de données de "ORA2" à partir de "ORA1", étant donné que l'utilisateur `user1` a le privilège `SELECT` sur ces objets sur ORA2 (comme TABLE1 ci-dessous):

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

Pré-requistes:

- Les deux bases de données doivent être opérationnelles (ouvertes).
- Les deux écouteurs de base de données doivent être opérationnels.
- TNS doit être configuré correctement.
- L'utilisateur user1 doit être présent dans la base de données ORA2, le mot de passe doit être vérifié et vérifié.
- L'utilisateur user1 doit avoir au moins le privilège `SELECT` ou tout autre moyen requis pour accéder aux objets sur ORA2.

Lire Liens de base de données en ligne: https://riptutorial.com/fr/oracle/topic/3859/liens-de-base-de-donnees

# Chapitre 18: Limiter les lignes renvoyées par une requête (Pagination)

## Examples

### Récupère les N premières lignes avec la clause de limitation de ligne

La clause `FETCH` a été introduite dans Oracle 12c R1:

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

Un exemple sans FETCH qui fonctionne aussi dans les versions précédentes:

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

### Pagination en SQL

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

de cette façon, nous pouvons paginer les données de la table, tout comme la page Web serch

### Obtenir N nombres d'enregistrements de la table

Nous pouvons limiter le nombre de lignes du résultat en utilisant la clause rownum

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

Si nous voulons le premier ou le dernier enregistrement, nous voulons une clause order by dans la requête interne qui donnera un résultat basé sur la commande.

**Last Five Record:**

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

**Premier cinq records**

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

## Récupère les lignes N à M à partir de nombreuses lignes (avant Oracle 12c)

Utilisez la fonction analytique 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 gère cela plus facilement avec `OFFSET` et `FETCH` .

## Sauter quelques lignes puis prendre quelques

Dans Oracle 12g +

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

Dans les versions antérieures

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

## Ignorer certaines lignes du résultat

Dans Oracle 12g +

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

Dans les versions antérieures

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

Lire Limiter les lignes renvoyées par une requête (Pagination) en ligne: https://riptutorial.com/fr/oracle/topic/4300/limiter-les-lignes-renvoyees-par-une-requete--pagination-

# Chapitre 19: Manipulation de cordes

## Examples

### Concaténation: Opérateur || ou fonction concat ()

Oracle SQL et PL / SQL `||` L'opérateur vous permet de concaténer deux chaînes ou plus.

**Exemple:**

En supposant la table `customers` suivante:

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

Question:

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

Sortie:

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

---

Oracle prend également en charge la fonction standard SQL `CONCAT(str1, str2)` :

**Exemple:**

Question:

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

Sortie:

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

### PLUS HAUT

La fonction UPPER vous permet de convertir toutes les lettres minuscules d'une chaîne en majuscules.

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

Sortie:

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

## INITCAP

La fonction `INITCAP` convertit la casse d'une chaîne de sorte que chaque mot commence par une majuscule et que toutes les lettres suivantes soient en minuscule.

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

Sortie

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

## INFÉRIEUR

LOWER convertit toutes les majuscules d'une chaîne en minuscules.

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

Les sorties:

| texte |
| --- |
| bonjour world123! |

## Expression régulière

Disons que nous voulons remplacer uniquement les nombres à 2 chiffres: l'expression régulière les trouvera avec `(\d\d)`

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

Résulte en:

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

Si je veux échanger des parties du texte, j'utilise `\1` , `\2` , `\3` pour appeler les chaînes correspondantes:

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

## SUBSTR

`SUBSTR` récupère une partie d'une chaîne en indiquant la position de départ et le nombre de caractères à extraire

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

résultats:

```
bcd
```

Pour compter à partir de la fin de la chaîne, `SUBSTR` accepte un nombre négatif comme second paramètre, par exemple

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

résultats:

```
de
```

Pour obtenir le dernier caractère d'une chaîne: `SUBSTR(mystring,-1,1)`

## LTRIM / RTRIM

`LTRIM` et `RTRIM` suppriment les caractères du début ou de la fin (respectivement) d'une chaîne. Un ensemble d'un ou plusieurs caractères peut être fourni (par défaut un espace) à supprimer.

Par exemple,

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

Résultats:

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

Lire Manipulation de cordes en ligne: https://riptutorial.com/fr/oracle/topic/1518/manipulation-de-cordes

# Chapitre 20: Mise à jour avec des jointures

## Introduction

Contrairement aux malentendus généralisés (y compris sur SO), Oracle autorise les mises à jour via des jointures. Cependant, il existe des exigences (assez logiques). Nous illustrons ce qui ne fonctionne pas et ce que fait un exemple simple. Une autre façon d'obtenir la même chose est la déclaration MERGE.

## Examples

### Exemples: ce qui fonctionne et ce qui ne fonctionne pas

```
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.
```

Imaginez ce qui arriverait si nous avions la valeur `1` dans la colonne `src.id` plus d'une fois, avec des valeurs différentes pour `src.val` . Evidemment, la mise à jour n'aurait aucun sens (dans N'IMPORTE QUELLE base de données, c'est une question logique). Maintenant, **nous** savons qu'il n'y a pas de doublons dans `src.id` , mais le moteur Oracle ne le sait pas - alors il se plaint. Peut-être est-ce la raison pour laquelle tant de praticiens pensent qu'Oracle "n'a pas mis à jour les jointures"?

Oracle s'attend à ce que `src.id` soit unique et qu'Oracle le sache au préalable. Facilement réparé! Notez que la même chose fonctionne avec les clés composites (sur plusieurs colonnes), si la correspondance pour la mise à jour doit utiliser plusieurs colonnes. En pratique, `src.id` peut être

PK et `tgt.id` peut être FK pointant vers cette PK, mais cela n'est pas pertinent pour les mises à jour avec jointure; ce qui *est* pertinent est la contrainte unique.

```
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
```

Le même résultat pourrait être obtenu avec une instruction MERGE (qui mérite son propre article Documentation), et je préfère personnellement MERGE dans ces cas-là, mais la raison n'est pas que "Oracle ne fait pas de mises à jour avec des jointures". Comme le montre cet exemple, Oracle *fait* faire des mises à jour avec des jointures.

Lire Mise à jour avec des jointures en ligne: https://riptutorial.com/fr/oracle/topic/8061/mise-a-jour-avec-des-jointures

# Chapitre 21: Oracle Advanced Queuing (AQ)

## Remarques

- N'utilisez jamais DDL ou DML sur des tables créées par `dbms_aqadm.create_queue_table` . Utilisez uniquement dbms_aqadm et dbms_aq pour travailler avec ces tables. Oracle peut créer plusieurs tables, index, etc. que vous ne connaissez pas. L'exécution manuelle de DDL ou DML sur la table peut vous conduire à un scénario dans lequel le support Oracle aura besoin de supprimer et de recréer la table et les files d'attente pour résoudre le problème.
- Il est fortement recommandé de ne pas utiliser `dbms_aq.forever` pour une option d'attente. Cela a causé des problèmes dans le passé, car Oracle peut commencer à planifier un nombre excessif de tâches de travail pour traiter les files d'attente inutiles (voir Oracle Doc ID 2001165.1).
- Il est recommandé de ne pas définir le paramètre AQ_TM_PROCESSES dans la version 10.1 ou ultérieure. Surtout, évitez de définir cette valeur sur zéro, car cela désactivera le travail en arrière-plan QMON nécessaire pour gérer les files d'attente. Vous pouvez réinitialiser cette valeur à la valeur par défaut Oracle à l'aide de la commande suivante et en redémarrant la base de données. `alter system reset aq_tm_processes scope=spfile sid='*';`

## Examples

### Simple producteur / consommateur

## Vue d'ensemble

Créez une file d'attente à laquelle vous pouvez envoyer un message. Oracle notifiera à notre procédure stockée qu'un message a été mis en file d'attente et doit être traité. Nous allons également ajouter des sous-programmes que nous pouvons utiliser en cas d'urgence pour empêcher le déqeuing des messages, autoriser une nouvelle mise en file d'attente et exécuter un simple traitement par lots pour traiter tous les messages.

Ces exemples ont été testés sur Oracle Database 12c Enterprise Edition version 12.1.0.2.0 - Production 64 bits.

## Créer une file d'attente

Nous allons créer un type de message, une table de files d'attente pouvant contenir les messages et une file d'attente. Les messages dans la file d'attente seront désaffectés en priorité, puis leur heure de mise en file d'attente. Si quelque chose se passe mal en travaillant le message et que la dequeue est annulée, AQ mettra le message à la disposition de la file d'attente 3600 secondes plus tard. Il le fera 48 fois avant de le déplacer dans une file d'exceptions.

```
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.
```

Maintenant que nous avons un endroit pour mettre les messages, créons un package pour gérer et travailler les messages dans la file d'attente.

```
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;
```

Ensuite, indiquez à AQ que lorsqu'un message est mis en file d'attente sur MESSAGE_Q (et qu'il est validé), notifiez notre procédure. AQ va commencer un travail dans sa propre session pour gérer cela.

```
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;
```

# Démarrer une file d'attente et envoyer un message

```
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;
```

Lire Oracle Advanced Queuing (AQ) en ligne: https://riptutorial.com/fr/oracle/topic/4362/oracle-advanced-queuing--aq-

# Chapitre 22: Oracle MAF

## Examples

### Pour obtenir la valeur de la liaison

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

Ici, "binding" indique l'expression EL à partir de laquelle la valeur doit être obtenue.

"nom_variable" paramètre auquel la valeur de la liaison doit être stockée

### Pour définir la valeur à la liaison

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

Ici, "binding" indique l'expression EL à laquelle la valeur doit être stockée.

"value" est la valeur désirée à ajouter à la liaison

### Pour appeler une méthode de liaison

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

"binding" indique l'expression EL à partir de laquelle une méthode à appeler

### Appeler une fonction javaScript

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

"fonction" est la fonction js désirée à invoquer

Lire Oracle MAF en ligne: https://riptutorial.com/fr/oracle/topic/6352/oracle-maf

# Chapitre 23: Partitionnement de table

## Introduction

Le partitionnement est une fonctionnalité permettant de diviser des tables et des index en morceaux plus petits. Il est utilisé pour améliorer les performances et pour gérer les petites pièces individuellement. La clé de partition est une colonne ou un ensemble de colonnes définissant la partition que chaque ligne va stocker. Vue d'ensemble du partitionnement dans la documentation officielle Oracle

## Remarques

Le partitionnement est une option de coût supplémentaire et uniquement disponible pour Enterprise Edition.

## Examples

### Partage de hachage

Cela crée une table partitionnée par hachage, dans cet exemple sur l'ID du magasin.

```
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;
```

Vous devez utiliser une puissance de 2 pour le nombre de partitions de hachage, afin d'obtenir une distribution uniforme de la taille de la partition.

### Partitionnement de gamme

Cela crée une table partitionnée par plages, dans cet exemple sur les valeurs de commande.

```
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)
);
```

## Sélectionnez des partitions existantes

Vérifier les partitions existantes sur le schéma

```
SELECT * FROM user_tab_partitions;
```

## Liste de partitionnement

Cela crée une table partitionnée par des listes, dans cet exemple sur l'ID de magasin.

```
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)
);
```

## Drop partition

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

## Sélectionnez les données d'une partition

Sélectionnez les données d'une partition

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

## Tronquer une partition

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

## Renommer une partition

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

## Déplacer la partition vers un autre tablespace

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

## Ajouter une nouvelle partition

```
ALTER TABLE table_name
ADD PARTITION new_partition VALUES LESS THAN(400);
```

### Partition fractionnée

Divise une partition en deux partitions avec une autre limite haute.

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

### Fusionner les partitions

Fusionner deux partitions en une seule

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

### Echanger une partition

Échangez / convertissez une partition en une table non partitionnée et inversement. Cela facilite un "déplacement" rapide des données entre les segments de données (par opposition à faire quelque chose comme "insert ... select" ou "create table ... as select") car l'opération est DDL (l'opération d'échange de partition est une donnée). mise à jour du dictionnaire sans déplacer les données réelles) et non DML (grande surcharge d'annulation / restauration).

Exemples les plus élémentaires:

1. Convertir une table non partitionnée (table "B") en une partition (de la table "A"):

La table "A" ne contient pas de données dans la partition "OLD_VALUES" et la table "B" contient des données

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

Résultat: les données sont "déplacées" de la table "B" (ne contient pas de données après l'opération) pour partitionner "OLD_VALUES"

2. Convertir une partition en une table non partitionnée:

La table "A" contient des données dans la partition "OLD_VALUES" et la table "B" ne contient pas de données

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

Résultat: les données sont "déplacées" de la partition "OLD_VALUES" (ne contient pas de données après l'opération) vers la table "B"

Remarque: il existe plusieurs options, fonctionnalités et restrictions supplémentaires pour cette opération.

Plus d'informations peuvent être trouvées sur ce lien ---> " https://docs.oracle.com/cd/E11882_01/server.112/e25523/part_admin002.htm#i1107555 " (section "Échanger des partitions")

Lire Partitionnement de table en ligne: https://riptutorial.com/fr/oracle/topic/3955/partitionnement-de-table

# Chapitre 24: Pompe de données

## Introduction

Voici les étapes pour créer une importation / exportation de pompe de données:

## Examples

### Surveiller les travaux DataPump

Les jobs de données peuvent être surveillés à l'aide de

**1. vues du dictionnaire de données:**

```
   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. Etat de la base de données:**

- Notez le nom du travail à partir des journaux d'importation / exportation ou du nom du dictionnaire de données et
- Exécuter la commande **join** :
- tapez le statut dans l'invite d'importation / exportation

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

Appuyez sur **CTRL + C** pour sortir de l'invite Import / Export

### Etape 3/6: Créer un répertoire

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

### Étape 7: Commandes d'exportation

Commandes:

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

* Veuillez remplacer les données dans <> avec les valeurs appropriées selon votre environnement. Vous pouvez ajouter / modifier des paramètres selon vos besoins. Dans l'exemple ci-dessus, tous les paramètres restants sont ajoutés dans les fichiers de paramètres, comme indiqué ci-dessous: *

- Type d'exportation: **Export utilisateur**
- Exporter le schéma entier
- Détails du fichier de paramètres [dire exp.par]:

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

---

- Type d'exportation: **Exporter par l'utilisateur pour un schéma volumineux**
- Exporter l'intégralité du schéma pour les jeux de données volumineux: les fichiers de vidage d'exportation seront décomposés et compressés. Le parallélisme est utilisé ici *(Remarque: l'ajout de parallélisme augmentera la charge du processeur sur le serveur)*
- Détails du fichier de paramètres [dire exp.par]:

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

---

- Type d'exportation: **Table Export** [Jeu d'export de tables]
- Détails du fichier de paramètres [dire exp.par]:

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

---

## Étape 9: Importer des commandes

Prérequis:

- Avant d'importer l'utilisateur, il est recommandé de supprimer le schéma ou la table importés.

Commandes:

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

* Veuillez remplacer les données dans <> avec les valeurs appropriées selon votre environnement. Vous pouvez ajouter / modifier des paramètres selon vos besoins. Dans l'exemple ci-dessus, tous les paramètres restants sont ajoutés dans les fichiers de paramètres, comme indiqué ci-dessous: *

- Type d'importation: **importation utilisateur**
- Importer le schéma entier

- Détails du fichier de paramètres [dire imp.par]:

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

---

- Type d'importation: **Importation utilisateur pour un schéma volumineux**
- Importer un schéma entier pour les jeux de données volumineux: le parallélisme est utilisé ici *(Remarque: l'ajout de parallélisme augmentera la charge du processeur sur le serveur)*
- Détails du fichier de paramètres [dire imp.par]:

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

---

- Type d'importation: **importation de tableau** [Ensemble d'importation de tables]
- Détails du fichier de paramètres [dire 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. étapes de la base de données

| Serveur source [Exporter les données] | Serveur cible [Importer des données] |
|---|---|
| 1. Créez un dossier datapump qui contiendra les fichiers de vidage d'exportation | 4. Créez un dossier de datapump qui contiendra les fichiers de vidage d'importation |
| 2. Connectez-vous au schéma de base de données qui effectuera l'exportation. | 5. Connectez-vous au schéma de base de données qui effectuera l'importation. |
| 3. Créez un répertoire pointant vers l’étape 1. | 6. Créez un répertoire pointant vers l’étape 4. |
| 7. Exécuter les déclarations d'exportation. | |
| 8. Copiez / SCP les fichiers de vidage sur le serveur cible. | |
| | 9. Exécuter les instructions d'importation |

| Serveur source [Exporter les données] | Serveur cible [Importer des données] |
| --- | --- |
| | 10. vérifier les données, compiler des objets non valides et fournir des subventions connexes |

## Copier des tables entre différents schémas et espaces de table

```
 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>
```

Lire Pompe de données en ligne: https://riptutorial.com/fr/oracle/topic/9391/pompe-de-donnees

# Chapitre 25: Récupération hiérarchique avec Oracle Database 12C

## Introduction

Vous pouvez utiliser des requêtes hiérarchiques pour extraire des données en fonction d'une relation hiérarchique naturelle entre les lignes d'une table.

## Examples

### Utiliser le CONNECT BY Caluse

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

La clause `CONNECT BY` pour définir la relation entre les employés et les responsables.

### Spécifier la direction de la requête depuis le haut vers le bas

```
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;
```

Lire Récupération hiérarchique avec Oracle Database 12C en ligne:
https://riptutorial.com/fr/oracle/topic/8777/recuperation-hierarchique-avec-oracle-database-12c

# Chapitre 26: Rendez-vous

## Examples

### Génération de dates avec le composant No Time

Toutes les `DATE` ont un composant temps; cependant, il est d'usage de stocker des dates qui ne nécessitent pas d'inclure des informations de temps avec les heures / minutes / secondes définies à zéro (c'est-à-dire minuit).

Utilisez un littéral ANSI `DATE` (en utilisant le format de date ISO 8601 ):

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

Convertissez-le à partir d'un littéral de chaîne à l'aide de `TO_DATE()` :

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

*(Vous trouverez plus d'informations sur les modèles de format de date dans la documentation Oracle.)*

ou:

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

*(Si vous convertissez des termes spécifiques à une langue, tels que les noms de mois, il est `nlsparam` inclure le troisième paramètre `nlsparam` dans la fonction `TO_DATE()` et de spécifier la langue à laquelle vous devez vous attendre.)*

### Générer des dates avec un composant de temps

Convertissez-le à partir d'un littéral de chaîne à l'aide de `TO_DATE()` :

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

Ou utilisez un littéral `TIMESTAMP` :

```
CREATE TABLE date_table(
  date_value DATE
);

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

Oracle `TIMESTAMP` implicitement un `TIMESTAMP` dans une `DATE` lors de son stockage dans une colonne `DATE` d'une table; Cependant, vous pouvez explicitement `CAST()` la valeur à un `DATE` :

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

## Le format d'une date

Dans Oracle, un type de données `DATE` n'a pas de format; Lorsque Oracle envoie une `DATE` au programme client (SQL / Plus, SQL / Developer, Toad, Java, Python, etc.), il envoie 7 ou 8 octets qui représentent la date.

Une `DATE` qui n'est pas stockée dans une table (c.-à-d. `SYSDATE` par `SYSDATE` et ayant "type 13" lors de l'utilisation de la commande `DUMP()` ) a 8 octets et a la structure (les chiffres à droite sont la représentation interne de `2012-11-26 16:41:09` ):

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

Une `DATE` qui est stockée dans une table ("type 12" lorsque vous utilisez la commande `DUMP()` ) a 7 octets et a la structure (les chiffres à droite sont la représentation interne de `2012-11-26 16:41:09` ):

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

Si vous voulez que la date ait un format spécifique, vous devrez le convertir en quelque chose qui a un format (une chaîne). Le client SQL peut implicitement le faire ou vous pouvez convertir explicitement la valeur en une chaîne à l' aide de `TO_CHAR( date, format_model, nls_params )` .

## Conversion de dates en chaîne

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

*(Remarque: si un modèle de format n'est pas fourni, le paramètre de session `NLS_DATE_FORMAT` sera utilisé comme modèle de format par défaut ; il peut être différent pour chaque session, il ne faut donc pas s'y fier. Il est recommandé de toujours spécifier le modèle de format.)*

```
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 );
```

Alors:

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

Les sorties:

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

Et:

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

Les sorties:

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

## Définition du modèle de format de date par défaut

Lorsque Oracle convertit implicitement une `DATE` en chaîne ou inversement (ou lorsque `TO_CHAR()` ou `TO_DATE()` sont explicitement appelés sans modèle de format), le paramètre de session `NLS_DATE_FORMAT` sera utilisé comme modèle de format dans la conversion. Si le littéral ne correspond pas au modèle de format, une exception sera déclenchée.

Vous pouvez revoir ce paramètre en utilisant:

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

Vous pouvez définir cette valeur dans votre session en cours en utilisant:

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

*(Remarque: cela ne change pas la valeur pour les autres utilisateurs.)*

Si vous comptez sur `NLS_DATE_FORMAT` pour fournir le masque de format dans `TO_DATE()` ou `TO_CHAR()` vous ne devriez pas être surpris lorsque vos requêtes sont interrompues si cette valeur est modifiée.

## Modification du mode d'affichage des dates SQL / Plus ou SQL Developer

Lorsque SQL / Plus ou SQL Developer affichent des dates, ils effectuent une conversion implicite en chaîne à l'aide du modèle de format de date par défaut (voir l'exemple de définition du modèle de format de date par défaut).

Vous pouvez modifier l'affichage d'une date en modifiant le paramètre `NLS_DATE_FORMAT` .

## Arithmétique des dates - Différence entre les dates en jours, heures, minutes et / ou secondes

En oracle, la différence (en jours et / ou en fractions) entre deux `DATE` peut être trouvée en utilisant la soustraction:

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

Affiche le nombre de jours entre les deux dates:

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

Et:

```
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
```

Affiche la fraction de jours entre deux dates:

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

La différence en heures, minutes ou secondes peut être obtenue en multipliant ce nombre par `24` , `24*60` ou `24*60*60` respectivement.

L'exemple précédent peut être modifié pour obtenir les jours, les heures, les minutes et les secondes entre deux dates en utilisant:

```
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
);
```

*(Remarque: `TRUNC()` est utilisé plutôt que `FLOOR()` pour gérer correctement les différences négatives.)*

Les sorties:

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

L'exemple précédent peut également être résolu en convertissant la différence numérique en un intervalle à l' aide de `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
);
```

## Arithmétique des dates - Différence entre les dates en mois ou en années

La différence en mois entre deux dates peut être trouvée en utilisant le `MONTHS_BETWEEN( date1, date2 )` :

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

Les sorties:

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

Si la différence comprend des mois partiels, la fraction du mois sera calculée en fonction de **31** jours par mois:

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

Les sorties:

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

En raison de `MONTHS_BETWEEN` supposant 31 jours par mois lorsqu'il peut y avoir moins de jours par mois, cela peut entraîner des valeurs différentes pour les différences couvrant les limites entre les mois.

Exemple:

```
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;
```

Sortie:

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

La différence en années peut être trouvée en divisant la différence de mois par 12.

## Extraire l'année, le mois, le jour, l'heure, la minute ou la seconde composante d'une date

Les composants année, mois ou jour d'un type de données `DATE` peuvent être trouvés à l'aide de la commande `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;
```

Les sorties:

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

Les composants temps (heure, minute ou seconde) peuvent être trouvés soit par:

- Utiliser `CAST( datevalue AS TIMESTAMP )` pour convertir la `DATE` en `TIMESTAMP` puis en utilisant `EXTRACT( [ HOUR | MINUTE | SECOND ] FROM timestampvalue )` ; ou
- Utiliser `TO_CHAR( datevalue, format_model )` pour obtenir la valeur sous forme de chaîne.

Par exemple:

```
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
);
```

Les sorties:

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

## Fuseaux horaires et heure avancée

Le type de données `DATE` ne gère pas les fuseaux horaires ni les modifications de l'heure d'été.

Non plus:

- utilisez le type de données `TIMESTAMP WITH TIME ZONE` ; ou
- gérer les modifications de la logique de votre application.

Une `DATE` peut être stockée en temps universel coordonné (UTC) et convertie dans le fuseau horaire de la session en cours comme suit:

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

Si vous exécutez `ALTER SESSION SET TIME_ZONE = '+01:00';` alors la sortie est:

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

et `ALTER SESSION SET TIME_ZONE = 'PST';` alors la sortie est:

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

## Secondes de saut

Oracle ne gère pas les secondes intercalaires . Voir My Oracle Support note `2019397.2` et `730795.1` pour plus de détails.

## Obtenir le jour de la semaine

Vous pouvez utiliser `TO_CHAR( date_value, 'D' )` pour obtenir le jour de la semaine.

Cependant, cela dépend du paramètre de session `NLS_TERRITORY` :

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

Sorties `5`

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

Sorties `4`

Pour ce faire, indépendamment des paramètres `NLS` , vous pouvez tronquer la date à minuit du jour actuel (pour supprimer les fractions de jours) et soustraire la date tronquée au début de la semaine iso en cours (qui commence toujours le lundi):

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

Lire Rendez-vous en ligne: https://riptutorial.com/fr/oracle/topic/2087/rendez-vous

# Chapitre 27: requête de niveau

## Remarques

La clause level est responsable de la génération d'un nombre N de dossiers factices en fonction d'une condition spécifique.

## Examples

### Générer N Nombre d'enregistrements

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

### Quelques utilisations de la requête de niveau

/ * Ceci est une requête simple qui peut générer une séquence de nombres. L'exemple suivant génère une séquence de nombres comprise entre 1..100 * /

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

/ * La requête ci-dessus est utile dans divers scénarios, comme la génération d'une séquence de dates à partir d'une date donnée. La requête suivante génère 10 dates consécutives * /

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

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

Lire requête de niveau en ligne: https://riptutorial.com/fr/oracle/topic/6548/requete-de-niveau

# Chapitre 28: Sécurité des applications réelles

## Introduction

Oracle Real Application Security a été introduit dans Oracle 12c. Il résume de nombreux sujets de sécurité tels que le modèle de rôle d'utilisateur, le contrôle d'accès, l'application vs la base de données, la sécurité de l'utilisateur final ou le niveau de ligne et de colonne.

## Examples

### Application

Pour associer une application à quelque chose dans la base de données, il y a trois parties principales:

**Privilège d'application:** un privilège d'application décrit des privilèges tels que `SELECT` , `INSERT` , `UPDATE` , `DELETE` , ... Les privilèges d'application peuvent être résumés en tant que privilège d'agrégation.

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

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

**Utilisateur de l'application:**

Utilisateur d'application simple:

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

Utilisateur d'application de connexion directe:

```
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;
```

(optionnel:)

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

**Rôle d'application:**

Rôle d'application régulière:

```
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;
```

Rôle d'application dynamique: (activé dynamiquement en fonction de l'état d'authenatication)

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

Rôles d'application prédéfinis:

Ordinaire:

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

Dynamique: (dépend de l'état d'authentification de l'utilisateur de l'application)

- `DBMS_AUTH` : (connexion directe ou autre méthode d'authentification de base de données)
- `EXTERNAL_DBMS_AUTH` : (connexion directe ou autre méthode d'authentification de base de données et utilisateur externe)
- `DBMS_PASSWD` : (connexion directe avec mot de passe)
- `MIDTIER_AUTH` : (authentification via une application de niveau intermédiaire)
- `XSAUTHENTICATED` : (application directe ou intermédiaire)
- `XSSWITCH` : (l'utilisateur est passé de l'utilisateur proxy à l'utilisateur de l'application)

Lire Sécurité des applications réelles en ligne: https://riptutorial.com/fr/oracle/topic/10864/securite-des-applications-reelles

# Chapitre 29: Séquences

## Syntaxe

- CREATE SEQUENCE SCHEMA.SEQUENCE {INCREMENT PAR ENTIER | COMMENCEZ AVEC INTEGER | MAXVALUE INTEGER | NOMAXVALUE INTEGER | MINVALUE INTEGER | NOMINVALUE INTEGER | CYCLE INTEGER | NOCYCLE INTEGER | CACHE | NOCACHE | COMMANDE | NOODER}

## Paramètres

| Paramètre | Détails |
|---|---|
| schéma | nom du schéma |
| incrémenter de | intervalle entre les nombres |
| Commencer avec | premier numéro nécessaire |
| Valeur max | Valeur maximale pour la séquence |
| nomaxvalue | La valeur maximale est définie par défaut |
| minvalue | valeur minimale pour la séquence |
| nominvalue | la valeur minimum est par défaut |
| cycle | Réinitialiser au début après avoir atteint cette valeur |
| nocycle | Défaut |
| cache | Limite de préallocation |
| nocache | Défaut |
| commande | Garantir l'ordre des nombres |
| pas de commande | défaut |

## Examples

### Créer une séquence: exemple

Objectif

Utilisez l'instruction CREATE SEQUENCE pour créer une séquence, qui est un objet de base de

données à partir duquel plusieurs utilisateurs peuvent générer des entiers uniques. Vous pouvez utiliser des séquences pour générer automatiquement des valeurs de clé primaire.

Lorsqu'un numéro de séquence est généré, la séquence est incrémentée, indépendamment de la transaction validée ou annulée. Si deux utilisateurs incrémentent simultanément la même séquence, les numéros de séquence acquis par chaque utilisateur peuvent comporter des lacunes, car les numéros de séquence sont générés par l'autre utilisateur. Un utilisateur ne peut jamais acquérir le numéro de séquence généré par un autre utilisateur. Une fois qu'une valeur de séquence est générée par un utilisateur, cet utilisateur peut continuer à accéder à cette valeur, que la séquence soit incrémentée ou non par un autre utilisateur.

Les numéros de séquence sont générés indépendamment des tables, de sorte que la même séquence peut être utilisée pour une ou plusieurs tables. Il est possible que des numéros de séquence individuels semblent être ignorés, car ils ont été générés et utilisés dans une transaction qui a finalement été annulée. De plus, un seul utilisateur peut ne pas se rendre compte que d’autres utilisateurs utilisent la même séquence.

Une fois qu'une séquence est créée, vous pouvez accéder à ses valeurs dans les instructions SQL avec la pseudo-colonne CURRVAL, qui renvoie la valeur actuelle de la séquence ou la pseudo-colonne NEXTVAL, qui incrémente la séquence et renvoie la nouvelle valeur.

**Conditions préalables**

Pour créer une séquence dans votre propre schéma, vous devez disposer du privilège système CREATE SEQUENCE.

Pour créer une séquence dans le schéma d'un autre utilisateur, vous devez disposer du privilège système CREATE ANY SEQUENCE.

Création d'une séquence: Exemple L'instruction suivante crée la séquence customers_seq dans l'exemple de schéma oe. Cette séquence peut être utilisée pour fournir des numéros d'identification client lorsque des lignes sont ajoutées à la table clients.

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

La première référence à customers_seq.nextval renvoie 1000. La seconde renvoie 1001. Chaque référence ultérieure renverra une valeur 1 supérieure à la référence précédente.

Lire Séquences en ligne: https://riptutorial.com/fr/oracle/topic/3709/sequences

# Chapitre 30: SQL dynamique

## Introduction

SQL dynamique vous permet d'assembler un code de requête SQL dans l'environnement d'exécution. Cette technique présente certains inconvénients et doit être utilisée très soigneusement. Dans le même temps, il vous permet de mettre en œuvre une logique plus complexe. PL / SQL exige que tous les objets, utilisés dans le code, doivent exister et être valides au moment de la compilation. C'est pourquoi vous ne pouvez pas exécuter les instructions DDL directement dans PL / SQL, mais SQL dynamique vous permet de le faire.

## Remarques

Quelques remarques importantes:

1. N'utilisez jamais la concaténation de chaîne pour ajouter des valeurs à la requête, utilisez plutôt des paramètres. C'est faux:

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

   Et c'est juste:

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

   Il y a deux raisons à cela. Le premier est la sécurité. La concaténation de chaînes permet d'effectuer une injection SQL. Dans la requête ci-dessous, si une variable contient la valeur `1 or 1 = 1` , l'instruction `UPDATE` à jour toutes les lignes de la table:

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

   La deuxième raison est la performance. Oracle analysera les requêtes sans paramètres à chaque exécution, tandis que la requête avec paramètre sera analysée une seule fois dans la session.

2. Notez que lorsque le moteur de base de données exécute une instruction DDL, il exécute une validation implicite avant.

## Examples

### Sélectionnez une valeur avec SQL dynamique

Supposons qu'un utilisateur souhaite sélectionner des données provenant de différentes tables.

Une table est spécifiée par l'utilisateur.

```
 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;
```

Appelez cette fonction comme d'habitude:

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

Tableau à tester:

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

## Insérer des valeurs dans SQL dynamique

L'exemple ci-dessous insère de la valeur dans le tableau de l'exemple précédent:

```
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;
/
```

## Mettre à jour les valeurs en SQL dynamique

Mettons à jour la table du premier exemple:

```
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;
/
```

## Exécuter l'instruction DDL

Ce code crée la table:

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

## Exécuter un bloc anonyme

Vous pouvez exécuter un bloc anonyme. Cet exemple montre également comment renvoyer une valeur à partir d'un SQL dynamique:

```
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;
/
```

Lire SQL dynamique en ligne: https://riptutorial.com/fr/oracle/topic/10905/sql-dynamique

# Chapitre 31: Table DUAL

## Remarques

`DUAL` table `DUAL` a une colonne `DUMMY` , définie pour être `VARCHAR2(1)` et une seule ligne avec une valeur `x` .

`DUAL` table `DUAL` est automatiquement créée dans le schéma `SYS` lorsque la base de données est créée. Vous pouvez y accéder depuis n'importe quel schéma.

Vous ne pouvez pas modifier la table `DUAL` .

Vous pouvez utiliser la table `DUAL` pour appeler n'importe quelle fonction à partir d'une instruction SQL. C'est utile car il n'y a qu'une seule ligne et oracle optimizer en sait tout.

## Examples

### L'exemple suivant renvoie la date et l'heure actuelles du système d'exploitation

```
select sysdate from dual
```

### L'exemple suivant génère des nombres entre start_value et end_value

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

Lire Table DUAL en ligne: https://riptutorial.com/fr/oracle/topic/7328/table-dual

# Chapitre 32: Transactions autonomes

## Remarques

Les cas d'utilisation typiques pour une transaction autonome sont.

1. Pour créer tout type de structure de journalisation, comme la structure de journalisation des erreurs expliquée dans l'exemple ci-dessus.
2. Pour l'audit des opérations DML dans des déclencheurs sur des tables, quel que soit le statut final de la transaction (COMMIT ou ROLLBACK).

## Examples

### Utilisation d'une transaction autonome pour les erreurs de journalisation

La procédure suivante est une procédure générique qui sera utilisée pour consigner toutes les erreurs dans une application dans un tableau de journal des erreurs commun.

```
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;
```

Le bloc PLSQL anonyme suivant montre comment appeler la procédure 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
```

Lire Transactions autonomes en ligne: https://riptutorial.com/fr/oracle/topic/6103/transactions-autonomes

# Chapitre 33: Travailler avec des dates

## Examples

### Date arithmétique

Oracle supporte `DATE` (inclut le temps à la seconde près) et `TIMESTAMP` (inclut le temps à la fraction de seconde) des types de données, qui permettent l'arithmétique (addition et soustraction) de manière native. Par exemple:

Pour obtenir le lendemain:

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

Pour obtenir la veille:

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

Pour ajouter 5 jours à la date actuelle:

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

Pour ajouter 5 heures à la date actuelle:

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

Pour ajouter 10 minutes à la date actuelle:

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

Pour ajouter 7 secondes à la date actuelle:

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

Pour sélectionner des lignes dont la date de `hire_date` est il y a 30 jours ou plus:

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

Pour sélectionner des lignes où la colonne `last_updated` est à la dernière heure:

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

Oracle fournit également le type de données intégré `INTERVAL` qui représente une durée (par

exemple, 1,5 jour, 36 heures, 2 mois, etc.). Celles-ci peuvent également être utilisées avec l'arithmétique avec les expressions `DATE` et `TIMESTAMP` . Par exemple:

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

## Fonction add_months

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

La fonction Add_months ajoute des mois à la date p_date.

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

| M |
| --- |
| 2015-03-12 |

Vous pouvez également soustraire des mois en utilisant un `amt` négatif

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

| M |
| --- |
| 2014-11-12 |

Lorsque le mois calculé a moins de jours que la date indiquée, le dernier jour du mois calculé sera renvoyé.

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

| M |
| --- |
| 2015-02-28 |

Lire Travailler avec des dates en ligne: https://riptutorial.com/fr/oracle/topic/768/travailler-avec-des-dates

# Crédits

| S. No | Chapitres | Contributeurs |
|---|---|---|
| 1 | Démarrer avec Oracle Database | Community, J. Chomel, Jeffrey Kemp, Jon Ericson, Kevin Montrose, Mark Stewart, Sanjay Radadiya, Steven Feuerstein, tonirush |
| 2 | Affacturage récursif des sous-requêtes à l'aide de la clause WITH (AKA Common Table Expressions) | B Samedi, MT0 |
| 3 | Astuces | Aleksej, Florin Ghita, Jon Heller, Mark Stewart, Pirate X |
| 4 | Bloc PL / SQL anonyme | Jon Heller, Skynet, Zohar Elkayam |
| 5 | contraintes | SSD |
| 6 | Créer un contexte | Jeffrey Kemp |
| 7 | Délimitation de mots-clés ou de caractères spéciaux | dev |
| 8 | Dictionnaire de données | Mark Stewart, Pancho, Slava Babin |
| 9 | Différentes façons de mettre à jour les enregistrements | nimour pristou, Nogueira Jr, SriniV |
| 10 | Erreur de journalisation | zygimantus |
| 11 | Fonctions de fenêtre | Leigh Riffel |
| 12 | Fonctions statistiques | Evgeniy K., Matas Vaitkevicius, ppeterka, Pranav Shah |
| 13 | Fractionnement de chaînes délimitées | Arkadiusz Łukasiewicz, MT0 |
| 14 | Gestion des valeurs | Dalex, JeromeFr |

| | | |
|---|---|---|
| | NULL | |
| 15 | Index | smshafiqulislam |
| 16 | JOINT | Aleksej, B Samedi, Bakhtiar Hasan, Daniel Langemann, Erkan Haspulat, Pranav Shah, Robin James, SriniV, Sumner Evans |
| 17 | Liens de base de données | carlosb, Daniel Langemann, g00dy, kasi |
| 18 | Limiter les lignes renvoyées par une requête (Pagination) | Ahmed Mohamed, Martin Schapendonk, Matas Vaitkevicius, Sanjay Radadiya, tonirush, trincot |
| 19 | Manipulation de cordes | carlosb, Eric B., Florin Ghita, Francesco Serra, J. Chomel, J.Hudler, Jeffrey Kemp, Mark Stewart, SriniV, Thunder, walen, zhliu03 |
| 20 | Mise à jour avec des jointures | mathguy |
| 21 | Oracle Advanced Queuing (AQ) | Jon Theriault |
| 22 | Oracle MAF | Anand Raj |
| 23 | Partitionnement de table | BobC, carlosb, ivanzg, JeromeFr, Kamil Islamov, Stephen Leppik, tonirush |
| 24 | Pompe de données | Vidya Thotangare |
| 25 | Récupération hiérarchique avec Oracle Database 12C | Muntasir, Vahid |
| 26 | Rendez-vous | carlosb, MT0, Roman, tonirush |
| 27 | requête de niveau | Sanjay Radadiya, TechEnthusiast |
| 28 | Sécurité des applications réelles | Ben H |
| 29 | Séquences | Pranav Shah, SriniV |
| 30 | SQL dynamique | Dmitry |
| 31 | Table DUAL | Slava Babin |
| 32 | Transactions autonomes | phonetic_man |

| | | |
|---|---|---|
| 33 | Travailler avec des dates | David Aldridge, Florin Ghita, Jeffrey Kemp, Mark Stewart, tonirush, zygimantus |