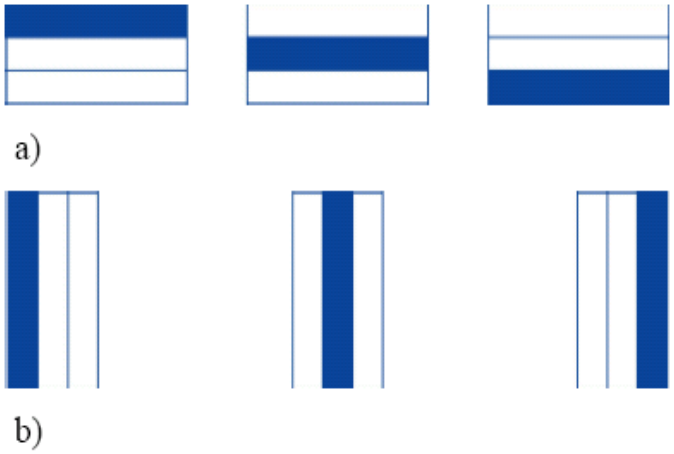
Université du Havre, UFR Sciences, Master Prof. Info.
SIRES3 (BDD), décembre 2016, enseignants : Nakechbandi M

Chapitre 2

Réplication et gestion d'intégrité référentielle dans les BDD sous Oracle.

2.1/ Rappel : La fragmentation



Fragmentations
a) horizontale et
b) verticale.

Fragmentation mixte :
a) fragments verticaux fragmentés horizontalement;
b) fragments horizontaux fragmentés verticalement.

La reconstruction

Une table Fragmentée se reconstruit à partir des fragments par des opérations d'**union** et de **jointure** :

Fragmentation Horizontale

```
CREATE VIEW V1 AS SELECT Table1.cle, Table1.attr1 FROM
Table1@site1
UNION
SELECT Table2.cle, Table2.attr1 FROM Table2@site2
```

Fragmentation Verticale

```
CREATE VIEW V1 AS SELECT Table1.cle, Table1.attr1, Table2.attr2
FROM Table1@site1, Table2@site2
WHERE Table1.cle=Table2.cle
```

2.2/ Les liens (Commande `CREATE DATABASE LINK`)

- **Paramètres d'un lien :**
  Lien à une BD distante spécifié par :
  - nom de lien
  - nom de l'utilisateur et password
  - chaîne de connexion (définie dans le fichier `tnsnames.ora`)
- **Types de lien :**
  fixed user :

```
CREATE [PUBLIC] DATABASE LINK nom_db_ en_local
    CONNECT TO scott IDENTIFIED BY tiger USING
                                'nom_bd_distante';
```

Current user :

```
CREATE DATABASE LINK nom_db_en_local
    CONNECT TO CURRENT_USER USING  'nom_bd_distante';
```

Ce type de lien suppose que compte interrogateur et compte accédé aient le même nom (dans leur base respective) et le même mot de passe au moment de la création du lien et au moment de l'utilisation du lien. Ce lien de base de données va vous permettre d'accéder à partir de votre compte en base local aux tables de votre compte sur la base distante.

- **Accès table distante :** `schéma.table@nom_db_en_local`
- **Privilège :** `grant create database link;`

Opérations sur liens

- Fermeture (Utile si forte charge) :
  `ALTER SESSION CLOSE DATABASE LINK machin;`
- Suppression :
  `DROP DATABASE LINK machin ;`
- Afficher les liens existants :
  `select owner, db_link, username from dba_db_links;`

Tables systèmes relatives aux liens :

- `DBA_DB_LINKS` (tous les liens),
- `ALL_DB_LINKS` (tous ceux accessibles par l'utilisateur),
- `USER_DB_LINKS` (tous ceux qui lui appartiennent),
- `V$DBLINK` (tous les liens ouverts par la transaction).

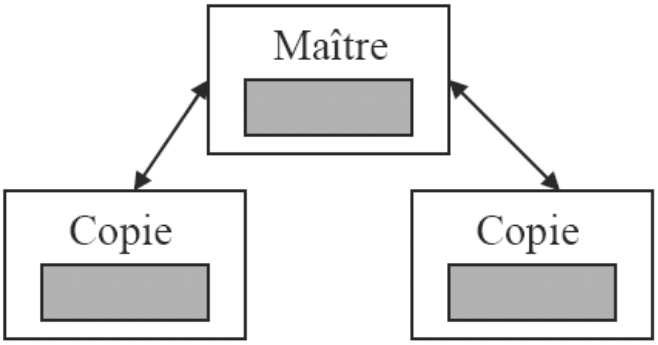
Exemples : (dans cet exemple $ est l'invite du système, SQL> est l'invite de oracle)

```
$sqlplus system/mpsystem
SQL> create user u1 identified by u1;
User created.
SQL> grant connect to u1;
Grant succeeded.
```

2.3/ **Réplication** sous Oracle

Définition : Bases de données répliquées est un ensemble de bases de données où certaines tables sont identiques, dont une appelée copie maître permet de créer les autres appelées copies esclaves.

L'objectif est de réduire la quantité de données transmises sur le réseau, et améliorer par conséquent les performances des requêtes.
lien vers une BD distante



Exemples d'utilisation :
Un bureau régional de vente possède une copie de la liste des prix des différents produits. Cette liste est maintenue au siège et est rafraîchie une fois par semaine. Fonctionnellement, l'entreprise n'a pas besoin de mettre à jour plus souvent sa liste de prix.

L'édition standard d'Oracle n'accepte qu'un seul site maître qui duplique les données vers des sites esclaves. Plusieurs options de réplication peuvent être envisagées :

Option 1 : Commande COPY

La première option consiste à répliquer régulièrement les données sur le serveur local au moyen de la commande COPY de SQL*Plus.

Exemple :
```
COPY FROM prix@regional CREATE prix_local USING SELECT * FROM prix
;
COMMIT ;
```

L'inconvénient est que les données ne peuvent pas être mises à jour.

Option 2 : Commande snapshots (instantané)

Cette option utilise des snapshots pour répliquer les données depuis une source maître vers plusieurs cibles. Les snapshots peuvent être en lecture seule (ang. read-only) ou mis à jour (ang. updateable). Avant de créer un snapshot, il faut d'abord créer un lien vers la base de données source.

Deux types de snapshots peuvent être crées : simples et complexes. Un snapshot simple ne contient pas de clause distinct, group by, connect by, de jointure multitables ou d'opérations set.

Syntaxe

```
CREATE SNAPSHOT [schema.]snapshot
    [ [PCTFREE  integer] [PCTUSED  integer]
      [INITRANS integer] [MAXTRANS integer]
      [TABLESPACE tablespace]
      [STORAGE storage_clause]
    [ USING INDEX [  PCTFREE integer | TABLESPACE tablespace
                   | INITTRANS integer | MAXTRANS integer
                   | STORAGE storage_clause ] ...
    | [CLUSTER cluster (column [, column]...)] ]
    [ REFRESH [FAST | COMPLETE | FORCE] [START WITH date] [NEXT date]]
AS subquery
```

Exemple:

Le snapshot suivant est défini de façon à extraire les données maîtres et renouveler l'opération 7 jours plus tard.

```
CREATE SNAPSHOT prix_local
REFRESH COMPLETE
START with SysDate NEXT SysDate+ 7            (chaque semaine)
AS SELECT * FROM prix@bd_regional;
```

Sur le site d'instantané(`SNAPSHOT`), Oracle crée une table local dénommée `prix_local`, contenant toutes les colonnes de la table maître prix. Oracle crée une vue dénommée `prix_local`, définie comme étant une requête sur la table distante `prix`. Il planifie aussi une tâche pour rafraîchir l'instantané.

Option 3 : Commande Vues matérialisées

A l'origine cette commande est utilisée pour sauvegarder le résultat d'une requête dans une table.

Syntaxe

```
CREATE MATERIALIZED VIEW nom
[TABLESPACE … STORAGE …]
[REFRESH FAST|COMPLETE|FORCE
START WITH sysdate
NEXT sysdate+1 // en jours…
WITH PRIMARY KEY // si possible …
USING ROLLBACK SEGMENT …]
AS
SELECT…
```

Exemple 1 :

```
CREATE MATERIALIZED VIEW Ventes-par-Mois
REFRESH COMPLETE
START WITH sysdate NEXT sysdate+1             (chaque jour)
AS
SELECT mois, SUM(montant) FROM Ventes GROUP BY mois;
```

Exemple 2 :

```
CREATE MATERIALIZED VIEW prix_local
AS SELECT * FROM prix@bd_regional;
```

Dans cette exemple la vue matérialisée est équivalente à la commande copie vue plus haut.

Exemple 3 :

```
create materialized view prix_local
refresh COMPLETE
start with sysdate
next sysdate + 1/24                                    (chaque heure)
as  SELECT * FROM prix@bd_regional;
```

Commandes complémentaires

```
DROP materialized VIEW Ventes-par-Mois;

DROP SNAPSHOT prix_local;
```

La close NEVER REFRESH empêche tout type d'actualisation de la vue matérialisée.

Rafraichissement à la demande

```
EXECUTE DBMS_REFRESH.REFRESH('VueMatérialisée');
```

Privilèges nécessaires pour la réplication

- ALTER ANY SNAPSHOT
- CREATE ANY SNAPSHOT
- DROP ANY SNAPSHOT
- CREATE DATABASE LINK
- CREATE SNAPSHOT

Exemple : `grant create snapshot to toto ;`

2.4/ Conception de la fragmentation.

2.4.1/ Introduction :

Pour la fragmentation **horizontale** les critères géographique et commerciale sont souvent utilisés.

Pour fragmenter **verticalement** on utilise une matrice A d'affinité des attributs :

Matrice $A = (a_{ij})$; a_{ij} = affinité d'attribut A_i avec l'attribut A_j , par exemple fréquences des requêtes qui accèdent aux attributs A_i et A_j ensemble.

	A1	**A2**	**A3**	**A4**
A1	45	0	45	0
A2		80	5	75
A3			53	3
A4				78

Par regroupement des attributs de cette matrice d'affinité on obtient :

	A1	**A3**	**A2**	**A4**
A1	45	45	0	0
A3		53	5	3
A2			80	75
A4				78

On voit bien ici qu'on a intérêt à mettre A1 et A3 ensemble dans le même fragment. Idem pour A2 et A4

2.4.2/ Calcul de matrice d'affinité.

Nous allons illustration ce calcul à partir d'un exemple :

Considérons la table : Projet (numP, nomP, budget, ville).

A1 A2 A3 A4

Sur cette table on retient les requêtes suivantes :

```
q1 : SELECT budget FROM projet WHERE numP = valeur;
q2 : SELECT nomP, budget FROM projet;
q3 : SELECT nomP FROM projet WHERE ville = valeur;
q4 : SELECT SUM(budget) FROM projet WHERE ville = valeur;
```

La table projet est consultée par trois site (s1, s2, s3), On connaît la fréquence d'accès des sites aux quatre requêtes (mesurée pendant une certaine période) :

Acc1(q1) = 15	Acc2(q1) = 20	Acc3(q1) = 10
Acc1(q2) = 5	Acc2(q2) = 0	Acc3(q2) = 0
Acc1(q3) = 25	Acc2(q3) = 25	Acc3(q3) = 25
Acc1(q4) = 3	Acc2(q4) = 0	Acc3(q4) = 0

Sur les données de cette exemple, voici les étapes permettant l'obtention de la matrice d'affinité.

1. Construire la matrice d'utilisation Ut définie par :

$$Ut(q_i, A_j) = \begin{cases} 1 \text{ si } q_i \text{ utilise } A_j \\ 0 \text{ sinon} \end{cases}$$

d'où :

Ut =

	A1	A2	A3	A4
q1	1	0	1	0
q2	0	1	1	0
q3	0	1	0	1
q4	0	0	1	1

2. La matrice d'affinité est définie par l'expression suivante :

$$Aff(A_i, A_j) = \sum_{\substack{k \text{ tq} \\ Ut(q_k, A_i)=1 \\ \wedge Ut(q_k, A_j)=1}} \sum_{s} Ref_s(q_k) Acc_s(q_k)$$

où

- $Ref_s(q_k)$, pour deux attributs (Ai, Ak) = nombre d'accès faits par une exécution de q_k (sur le site s) à A_i et A_k. Dans la suite $Ref_s(q_k) = 1$ pour tout k, s
- $Accs(q_k)$ = fréquence de qk sur le site s (définie plus haut).

Pour l'exemple précédent on aura :

$$Aff(A1, A3) = \Sigma_{K=1} \Sigma_s Acc_s(q_k)$$
$$= Acc_1(q_1) + Acc_2(q_1) + Acc_3(q_1) = 45$$

la matrice d'affinité est donc :

	A1	A2	A3	A4
A1	45	0	45	0
A2	0	80	5	75
A3	45	5	53	3

A4	0	75	3	78

2.4.3/ Fragmentation (basée sur une heuristique)

A partir de la matrice d'affinité obtenue, les étapes ci-dessus propose une fragmentation verticale en **deux tables** de la table `projet` :

- Regroupement des attributs ayant une haute affinité :

	A1	A3	A2	A4
A1	45	45	0	0
A3	45	53	5	3
A2	0	5	80	75
A4	0	3	75	78

- Trouver un point dans la matrice pour créer deux ensembles d'attributs : AttrHaut (AH) et AttrBas (AB) : AH = { A1, A3} ; AB = { A2, A4}
- Pour garantir la reconstruction de la table projet on ajoute l'attribut A1 à AH

D'où la table projet sera fragmentée en :

```
projet1( Num, Budget) ;    projet2( Num, Nom, Loc) ;
```

Remarque : Les étapes précédentes pouvant être appliqués à un fragment pour obtenir une fragmentation en trois tables.

Exercice : Soient Q = { q_1, q_2, q_3, q_4, q_5} un ensemble de requêtes, A = { A_1, A_2, A_3, A_4, A_5} un ensemble d'attributs, et S = { S_1, S_2, S_3} un ensemble de sites. La matrice (a) ci-dessous décrit l'utilisation des attributs par les requêtes et la matrice (b) décrit la fréquence d'utilisation des requêtes par les sites. Utiliser la méthode du paragraphe 2.4. pour calculer la matrice d'affinité puis déterminer une fragmentation verticale de A sur les 3 sites. On suppose ici que la clé de la table A est A_1 et ref_i (q_k) = 1 Pour tout q_k et S_i

	A_1	A_2	A_3	A_4	A_5
q_1	0	1	1	0	1
q_2	1	1	1	0	1
q_3	1	0	0	1	1
q_4	0	0	1	0	0
q_5	1	1	1	0	0

(a)

	S_1	S_2	S_3
q_1	10	20	0
q_2	5	0	10
q_3	0	35	5
q_4	0	10	0
q_5	0	15	0

(b)

Références.

1. Th. Connolly C. Begg, Systemes de gesion de base de données, Ed. Eyrolles, 2005.
2. A. Abdellatif, M. Limame, A. Zeroual, Oracle 7 Langages Architecture Administration, Ed. Eyrolles, 1995.
3. M. Tamer Ozsu, Patrick Valduriez, Principles of Distributed Database Systems, Ed. Prentice Hall,1999.

4. M. Exbrayat Bases de Données Réparties Concepts et Techniques, ULP Strasbourg - Décembre 2007.

Annexe : Exemples sur les triggers.

Exemple 1 : Une société de vente d'électroménager ayant plusieurs points de vente répartis dans toutes les grandes villes et un siège établit à GAP. Chaque site dispose d'un serveur faisant tourner le SGBD ORACLE. Le siège idem.

Tous les sites vendent les mêmes appareils, la table appareil étant centralisée au siège. Seuls Rpignon, Hjunot et Xcuerten employés du siège et responsables du service marketing ont le droit d'insertion, de mise à jour et de suppression sur la table `APPAREILS`. C'est Rpignon qui est le user propriétaire de la table. Les deux autres y accèdent par `Rpignon.APPAREIL`

Exemple :

```
connect Xcuerten/glouglou@alias_Gap ;
select * from Rpignon.appareil ;
```

Les employés des sites délocalisés ont uniquement un droit de consultation sur cette table.

Voici la structure de la table : `APPAREILS=(no_appareil, designation, Pri_unit_HT Caracteristique_technique) ;`

Les consultations de ces tables sont si fréquentes que les lignes de connection (ADSL) étaient saturées. Il a donc été décidé de recopier la table centrale sur tous les autres sites.

Mais attention, les copies de la table doivent contenir les mêmes informations que l''original.

Si RPignon modifie le prix unitaire du lave linge THOMSON SQ658 dans l'original, il est important de mettre les copies à jour. La mise à jour des copies peut se faire:

- Immédiatement ou de temps en temps (exemple: tous les soirs).
- Lorsqu'elle se fait immédiatement on parle de réplication synchrone: les copies sont synchronisées avec l''original.
- Lorsqu'elle se fait périodiquement on parle de réplication asynchrone.

Mise en Oeuvre de la réplication synchrone:

Elle nécessite l'écriture d'un trigger sur la base centrale (à GAP) pour la table `APPAREILS`.

```
CONNECT Rpignon/sonmotdepasse@alias_GAP
CREATE OR REPLACE TRIGGER maj_copies_APPAREILS
BEFORE INSERT OR UPDATE OR DELETE
ON APPAREILS
BEGIN
IF INSERTING THEN
INSERT INTO APPAREILS@LIEN_Toulouse values
( :new.no_appareil,
:new.designation,
:new.Pri_uni_HT,
:new.Caracteristique_technique  );
INSERT INTO APPAREILS@LIEN_Paris values
( :new.no_appareil,
:new.designation,
:new.Pri_uni_HT,
:new.Caracteristique_technique );
INSERT INTO APPAREILS@LIEN_Bordeaux values
( :new.no_appareil,
:new.designation,
:new.Pri_uni_HT,
:new.Caracteristique_technique );
END IF;
IF UPDATING THEN
UPDATE APPAREILS@Lien_Toulouse SET
Designation=:new.designation,
Pri_uni_HT= :new.Pri_uni_HT,
Caracteristique_technique= :new.Caracteristique_technique
WHERE no_appareil=:new.no_appareil;
UPDATE APPAREILS@Lien_Bordeaux SET
Designation=:new.designation,
Pri_uni_HT= :new.Pri_uni_HT,
Caracteristique_technique= :new.Caracteristique_technique
WHERE no_appareil=:new.no_appareil;
UPDATE APPAREILS@Lien_Paris SET
Designation=:new.designation,
Pri_uni_HT= :new.Pri_uni_HT,
Caracteristique_technique= :new.Caracteristique_technique
WHERE no_appareil=:new.no_appareil;
END IF;
IF DELETING THEN
DELETE FROM APPAREILS@Lien_Paris where no_appareil=:new.no_appareil;
DELETE FROM APPAREILS@Lien_Bordeaux where no_appareil=:new.no_appareil;
DELETE FROM APPAREILS@Lien_Toulouse where no_appareil=:new.no_appareil;END IF; END;
```

Exemple 2 : Création d'une table fragmentée verticalement

```
# Sur base1
CREATE TABLE Produit1( idprod INTEGER, nomprod VARCHAR(10),
prixachat FLOAT, CONSTRAINT idprod_pk PRIMARY KEY (idprod));
# Sur base2
CREATE TABLE Produit2(idprod INTEGER, prixvente FLOAT, CONSTRAINT idprod_pk PRIMARY
KEY (idprod));
# Création de la vue
CREATE VIEW Produit
AS SELECT Produit1.idprod, Produit1.nomprod, Produit1.prixachat,
Produit2.prixvente
FROM Produit1@BASE1.RIA, Produit2@BASE2.RIA
#Création du trigger sur les 2 bases
CREATE OR REPLACE TRIGGER Tr_prod1
INSTEAD OF INSERT on Produit
FOR EACH ROW
BEGIN
      INSERT INTO Produit1@BASE1.RIA(idprod,nomprod,prixachat)
      VALUES (:New.idprod,:New.nomprod,:New.prixachat);
      INSERT INTO Produit2@BASE2.RIA(idprod,prixvente)
      VALUES (:New.idprod,:New.prixvente);
END;
#Test
INSERT INTO Produit (idprod, nomprod, prixachat, prixvente)
VALUES ('1', 'pain', '1', '2');
```


2.3/ Gestion des contraintes distribuées :

Dans un SGBD centralisé, les ontaintes d'intgritées sont declarées principalement dans la définition des tables. Le SGBD distribué Oracle ne gère pas l'intégrité inter-bases (Domaine, Unicité, Intégrité référentielle), il propose l'utilisation des triggers (déclencheur). et des procédure PL/SQL.

2.3.1/ Procédures. Les unités de programmes **PL/SQL**, peuvent servir à :

1. **Référer à des données distantes.**

   Exemple : Considérons la procédure LicencierEmployé sur un SGBD `base1`:

```
CREATE PROCEDURE LicencierEmployé (enum NUMBER) AS
BEGIN
  DELETE FROM Employé@base2
  WHERE NumEmp = enum
END;
```

   SQL_base1> `execute LicencierEmployé(1250);`

   Nous pouvons également créer un synonyme pour le lien à la table Employé.

```
CREATE SYNONYM Emp FOR Employé@base2;
```

   La procédure LicencierEmployé devient :

```
CREATE PROCEDURE LicencierEmployé (enum NUMBER) AS
BEGIN  DELETE FROM Emp WHERE NumEmp = enum;  END;
```

2. **Appeler des procédures distantes.** Partant d'un 3ème SGBD (`base3`) on peut (à distance) appeler la procédure précédente :

   SQL_base3> `execute LicencierEmployé(1250)@base1;`

2.3.2/ Les triggers : Les triggers, (en français déclencheurs) permettant de garantir les contraintes d'intégrité référentielle, c'est-à-dire notamment de s'assurer qu'un tuple utilisé à partir d'une autre table existe réellement. Syntaxe relative à la création des triggers :

```
CREATE [OR REPLACE] TRIGGER trigger_name
     timing
          event1 [OR event2 OR event3]
               ON table_name
```

```
trigger_body
```

- `Trigger_name` : Nom du trigger.
- `Timing` : Indique le moment où le trigger se déclenchera par rapport à l'événement déclencheur.
  Pour la table :
  - BEFORE (avant) : Exécution du corps du trigger avant le déclenchement de l'événement LMD sur la table.
  - AFTER (après) : Exécution du corps du trigger après le déclenchement de l'événement LMD sur la table.

  Pour la vue :
  - INSTEAD OF : Exécution de manière invisible en tâche de fond, et effectue directement sur les tables sous-jacentes l'action codée dans le corps du trigger.
- `Event` : Identifie l'opération de manipulation de données à la base du déclenchement du trigger : `DELETE, INSERT, UPDATE` [OF liste colonne].
- `table/ view_name` : Spécifie la table associée au trigger.
- `trigger body` : Il s'agit du corps de trigger définissant l'action exécutée par le trigger. Il commence par DECLARE ou BEGIN et se termine par END ou un appel vers une procédure PL/SQL. Syntaxe simplifiée :

```
CREATE TRIGGER nom_trigger
BEFORE | AFTER
INSERT | DELETE | UPDATE [ OF liste_colonne]
ON nom_table
FOR EACH STATEMENT | ROW
[WHEN condition]
BEGIN  code_trigger  END
```

Opération sur les déclencheurs :

- Suppression : `DROP TRIGGER nom_trigger ON nom_table [CASCADE | RESTRICT]`
- Désactivation d'un trigger : `ALTER TRIGGER <nom_trigger> DISABLE;`
- Réactivation d'un trigger : `ALTER TRIGGER <nom_trigger> ENABLE;`
- Tous les triggers d'une table : `ALTER TABLE <nom_table> {ENABLE|DISABLE} ALL TRIGGERS;`
- Afficher les triggers (Consulter le dictionnaire de données) :
  `desc USER_TRIGGERS ;`
  `Select TRIGGER_NAME, STATUS from  USER_TRIGGERS ;`

Exemples sur les triggers

Exemple 1 : Créer deux copies identique. On considère ici la table :

```
emprunteur(num_emp, nom_emp, prenom_emp,
                    ville_emp, inscription_emp, interdit_jusqu_a);
```

dont une copie se trouve dans une BD_locale et une autre identique dans BD_distante. Le triger ci-dessous assure l'insertion dans la copie distante à chaque fois il y a une insertion dans la copie locale

```
CREATE TRIGGER copies_identique_emprunteur
After insert on emprunteur
For each row
BEGIN
  insert into emprunteur@BD_distante values
 (:new.num_emp, :new.nom_emp,:new.prenom_emp, :new.ville_emp,
    :new.inscription_emp, :new.interdit_jusqu_a);
```

```
END;/
```

Utilisables également pour `update` et `delete (:New et :Old)` pour Suppression `(:Old)`

Exemple 2 : Voici une autre solution améliorée.

```
CREATE OR REPLACE TRIGGER insert_emprunteur
AFTER INSERT ON EMPRUNTEUR
FOR EACH ROW  DECLARE num2 NUMBER;
BEGIN
select count(num_emp) into num2 from emprunteur@BD_distante where
num_emp = :new.num_emp;
if (num2 = 0) then l'emprunteur -- n'existe donc pas sur le site 2
insert into emprunteur@BD_distante values
    (:new.num_emp, :new.nom_emp,:new.prenom_emp, :new.ville_emp,
                    :new.inscription_emp, :new.interdit_jusqu_a);
end if;
END;/
```

Exemple 3 : Le trigger ci-dessous sert à supprime dans la table emprunteur distante les changements suite à une suppression dans la table emprunteur local

```
CREATE OR REPLACE TRIGGER delete_emprunteur
AFTER DELETE ON emprunteur
FOR EACH ROW
BEGIN
IF deleting THEN  -- Suppression emprunteur
 delete from emprunteur@BD_distante where num_emp = :old. num_emp;
END;/
```

Exemple 4 : Gestion d'intégrité référentielle.

On considérer deux bases de données samu et prefecture (vue en TD). La BD `samu` contennant une table `accident(nacc, date_acc,, route, nvec,… )` faisant référence à une table `voiture(nvec, marc, type, …)` dans la base `prefecture`.

Le trigrer suivant vérifie avant la suppression d'un enregistrement dans la table local `voiture`, que le numéro de voiture `nvec` de l'enregistrement concerné n'est référencé par aucune table distante, en l'occurrence par la table `accident`.

```
CREATE TRIGGER voiture_nvec
Befor delete on voiture
For each row
Declare  Unicite exeption ; nveclocal number;
BEGIN
  nveclocal :=0 ;
  Select count(nvec) into nvelocal form
                      accident@samulink nvec=OLD.nvec;
  If (nveclocal != 0) then rais unicite; endif
  Exception
  When unicite then
       rais_application_erreur(20009,'supression dans la
       table voiture : numero de vehicule référencé dans
       la table accident') ;
END;/
```

Exemple 5 : Gestion d'une table fragmentée verticalement

 # Sur base1 : `CREATE TABLE Produit1( idprod INTEGER, nomprod VARCHAR(10), prixachat FLOAT, CONSTRAINT idprod_pk PRIMARY KEY (idprod));`

 # Sur base2 : `CREATE TABLE Produit2(idprod INTEGER, prixvente FLOAT, CONSTRAINT idprod_pk PRIMARY KEY (idprod));`

 # Sur un site de vent base3 : Création de la vue

```
CREATE VIEW Produit
AS SELECT Produit1.idprod, Produit1.nomprod, Produit1.prixachat,
Produit2.prixvente
FROM Produit1@BASE1.RIA, Produit2@BASE2.RIA
where Produit1.idprod = Produit2.idprod ;
```

```
#Création du trigger sur les 2 bases
CREATE OR REPLACE TRIGGER Tr_prod1
INSTEAD OF INSERT on Produit
FOR EACH ROW
BEGIN
     INSERT INTO Produit1@BASE1.RIA(idprod,nomprod,prixachat)
     VALUES (:New.idprod,:New.nomprod,:New.prixachat);
     INSERT INTO Produit2@BASE2.RIA(idprod,prixvente)
     VALUES (:New.idprod,:New.prixvente);
END;
```

```
#Test
INSERT INTO Produit (idprod, nomprod, prixachat, prixvente)
VALUES ('1', 'pain', '1', '2');
```

Exemple 6. Etude de cas. Présentation : Une société de vente d'électroménager ayant plusieurs points de vente répartis et un siège établit à Paris. Chaque site dispose d'un serveur faisant tourner le SGBD ORACLE, la table appareil étant centralisée au siège. Seuls RP, HJ et XC employés du siège et responsables du service marketing ont le droit d'insertion, de mise à jour et de suppression sur la table `APPAREILS`. C'est RP qui est le user propriétaire de la table. Les deux autres y accèdent par `RP.APPAREIL`.

```
connect XC/glouglou@alias_paris ;
select * from RP.appareil ;
```

Les employés des sites délocalisés ont uniquement un droit de consultation sur cette table.
Voici la structure de la table :
`APPAREILS=(no appareil, designation, Pri_unit_HT, Caracteristique) ;`

Les consultations de ces tables sont si fréquentes que les lignes de connection (ADSL) étaient saturées. Il a donc été décidé de recopier la table centrale sur tous les autres sites.
Mais attention, les copies de la table doivent contenir les mêmes informations que l''original.
Si RP modifie le prix unitaire du lave linge THOMSON SQ658 dans l'original, il est important de mettre les copies à jour.

Solution :

- 1ere solution copie par snapshot ou vue matérialisé (**Réplication Synchrone**) : La mise à jour des copies peut se faire de temps en temps, par exemple tous les soirs (certainement ici ce n'est pas la bonne solution).
- La mise à jour des copies peut se faire Immédiatement on parle de **réplication asynchrone**.

Mise en Œuvre de la réplication synchrone : Elle nécessite l'écriture d'un trigger sur la base centrale (à PARIS) pour la table `APPAREILS`.

```
Sqlplus RP/sonmotdepasse@alias_paris
```

```
CREATE OR REPLACE TRIGGER
maj_copies_APPAREILS
```

```
BEFORE INSERT OR UPDATE OR DELETE
ON APPAREILS
BEGIN
IF INSERTING THEN
INSERT INTO APPAREILS@LIEN_Toulouse
values ( :new.no_appareil,
:new.designation,
:new.Pri_uni_HT,
:new.Caracteristique_technique );

INSERT INTO APPAREILS@LIEN_Paris
values ( :new.no_appareil,
:new.designation,
:new.Pri_uni_HT,
:new.Caracteristique_technique);

INSERT INTO APPAREILS@LIEN_Bordeaux
values ( :new.no_appareil,
:new.designation,
:new.Pri_uni_HT,
:new.Caracteristique_technique);
END IF;
IF UPDATING THEN
UPDATE APPAREILS@Lien_Toulouse SET
Designation=:new.designation,
Pri_uni_HT= :new.Pri_uni_HT,
Caracteristique_technique=
:new.Caracteristique_technique
WHERE no_appareil=:new.no_appareil;

UPDATE APPAREILS@Lien_Bordeaux SET
Designation=:new.designation,
Pri_uni_HT= :new.Pri_uni_HT,
Caracteristique_technique=
:new.Caracteristique_technique
WHERE no_appareil=:new.no_appareil;

UPDATE APPAREILS@Lien_Paris SET
Designation=:new.designation,
Pri_uni_HT= :new.Pri_uni_HT,
Caracteristique_technique=
:new.Caracteristique_technique
WHERE no_appareil=:new.no_appareil;
END IF;

IF DELETING THEN
DELETE FROM APPAREILS@Lien_Paris
where no_appareil=:new.no_appareil;

DELETE FROM APPAREILS@Lien_Bordeaux
where no_appareil=:new.no_appareil;

DELETE FROM APPAREILS@Lien_Toulouse
where no_appareil=:new.no_appareil;
END IF;
END;
```

2.4/ Conception de la fragmentation.

2.4.1/ Introduction :

Pour fragmenter verticalement on utilise une matrice A d'affinité des attributs :

Matrice $A = (a_{ij})$; a_{ij} = affinité d'attribut Ai avec l'attribut Aj
ex: nb de requêtes qui accèdent Ai et Aj

	A1	**A2**	**A3**	**A4**
A1	45	0	45	0
A2		80	5	75
A3			53	3
A4				78

Par regroupement des attributs de cette matrice d'affinité on obtient :

	A1	**A3**	**A2**	**A4**
A1	45	45	0	0
A3		53	5	3
A2			80	75
A4				78

On voit bien ici qu'on a intérêt à mettre A1 et A3 ensemble dans le même fragment. Idem pour A2 et A4

2.4.2/ Calcul de matrice d'affinité.

Nous allons illustration ce calcul à partir d'un exemple :

Considérons l'exemple la table : Projet (numP, nomP, budget, ville).

A1 A2 A3 A4

Sur cette table on retient les requêtes suivantes :

```
q1 : SELECT budget FROM projet WHERE numP = valeur;
q2 : SELECT nomP, budget FROM projet;
q3 : SELECT nomP FROM projet WHERE ville = valeur;
q4 : SELECT SUM(budget) FROM projet WHERE ville = valeur;
```

La table projet est consultée par trois site (s1, s2, s3), On connaît la fréquence d'accès des sites aux quatre requêtes (mesurée pendant une certaine période) :

```
Acc1(q1) = 15    Acc2(q1) = 20    Acc3(q1) = 10
Acc1(q2) = 5     Acc2(q2) = 0     Acc3(q2) = 0
Acc1(q3) = 25    Acc2(q3) = 25    Acc3(q3) = 25
Acc1(q4) = 3     Acc2(q4) = 0     Acc3(q4) = 0
```

Sur les données de cette exemple, voici les étapes permettant l'obtention de la matrice d'affinité.

1. Construire la matrice d'utilisation Ut définie par :

$$Ut(q_i, A_j) = \begin{cases} 1 \text{ si } q_i \text{ utilise } A_j \\ 0 \text{ sinon} \end{cases}$$

d'où :

Ut =

	A1	A2	A3	A4
q1	1	0	1	0
q2	0	1	1	0
q3	0	1	0	1
q4	0	0	1	1

2. La matrice d'affinité est définie par l'expression suivante :

$$Aff(A_i, A_j) = \sum_{\substack{k\ tq \\ Ut(q_k, A_i)=1 \\ \wedge Ut(q_k, A_j)=1}} \sum_{s} Ref_s(q_k) Acc_s(q_k)$$

où

- $Ref_s(q_k)$, pour deux attributs (Ai, Ak) = nombre d'accès faits par une exécution de q_k (sur le site s) à A_i et A_k. Dans la suite $Ref_s(q_k) = 1$ pour tout k, s
- $Accs(q_k)$ = fréquence de qk sur le site s (définie plus haut).

Pour l'exemple précédent on aura :

$$Aff(\text{A1}, \text{A3}) = \Sigma_{K=1} \Sigma_s Acc_s(q_k)$$
$$= Acc_1(q_1) + Acc_2(q_1) + Acc_3(q_1) = 45$$

la matrice d'affinité est donc :

	A1	A2	A3	A4
A1	45	0	45	0
A2	0	80	5	75

A3	45	5	53	3
A4	0	75	3	78

2.4.3/ Fragmentation

A partir de la matrice d'affinité obtenue, on peut proposer une fragmentation verticale pour la table `projet` :

- Regroupement des attributs ayant une haute affinité :

	A1	A3	A2	A4
A1	45	45	0	0
A3	45	53	5	3
A2	0	5	80	75
A4	0	3	75	78

- Trouver un point dans la matrice pour créer deux ensembles d'attributs : AttrHaut (AH) et AttrBas (AB) : AH = { A1, A3} ; AB = { A2, A4}

- Pour garantir la reconstruction de la table projet on ajoute l'attribut A1 à AH

D'où la table projet sera fragmentée en :

```
projet1( Num, Budget) ;    projet2( Num, Nom, Loc) ;
```

Exercice.
Soient Q = { q_1, q_2, q_3, q_4, q_5} un ensemble de requêtes, A = { A_1, A_2, A_3, A_4, A_5} un ensemble d'attributs, et S = { S_1, S_2, S_3} un ensemble de sites. La matrice (a) ci-dessous décrit l'utilisation des attributs par les requêtes et la matrice (b) décrit la fréquence d'utilisation des requêtes par les sites. Utiliser la méthode précédente pour calculer la matrice d'affinité puis déterminer une fragmentation verticale de A sur les 3 sites. On suppose ici que la clé de la table A est A_1 et $ref_i(q_k) = 1$; pour tout q_k et S_i

	A_1	A_2	A_3	A_4	A_5
q_1	0	1	1	0	1
q_2	1	1	1	0	1
q_3	1	0	0	1	1
q_4	0	0	1	0	0
q_5	1	1	1	0	0

(a)

	S_1	S_2	S_3
q_1	10	20	0
q_2	5	0	10
q_3	0	35	5
q_4	0	10	0
q_5	0	15	0

(b)

Références.

5. Th. Connolly C. Begg, Systemes de gesion de base de données, Ed. Eyrolles, 2005.
6. A. Abdellatif, M. Limame, A. Zeroual, Oracle 7 Langages Architecture Administration, Ed. Eyrolles, 1995.

7. M. Tamer Ozsu, Patrick Valduriez, Principles of Distributed Database Systems, Ed. Prentice Hall,1999.
8. M. Exbrayat Bases de Données Réparties Concepts et Techniques, ULP Strasbourg - Décembre 2007.

Annexe : Exemples sur les triggers.

Exemple 1 : Une société de vente d'électroménager ayant plusieurs points de vente répartis dans toutes les grandes villes et un siège établit à PARIS. Chaque site dispose d'un serveur faisant tourner le SGBD ORACLE. Le siège idem.
Tous les sites vendent les mêmes appareils, la table appareil étant centralisée au siège. Seuls RP, HJ et XC employés du siège et responsables du service marketing ont le droit d'insertion, de mise à jour et de suppression sur la table `APPAREILS`. C'est RP qui est le user propriétaire de la table. Les deux autres y accèdent par `RP.APPAREIL`
Exemple :

```
connect XC/glouglou@alias_Paris ;
select * from RP.appareil ;
```

Les employés des sites délocalisés ont uniquement un droit de consultation sur cette table.
Voici la structure de la table :

```
APPAREILS=(no appareil, designation, Pri_unit_HT,
                                    Caracteristique_technique) ;
```

Les consultations de ces tables sont si fréquentes que les lignes de connection (ADSL) étaient saturées. Il a donc été décidé de recopier la table centrale sur tous les autres sites.

Mais attention, les copies de la table doivent contenir les mêmes informations que l''original.

Si RP modifie le prix unitaire du lave linge THOMSON SQ658 dans l'original, il est important de mettre les copies à jour.

La mise à jour des copies peut se faire:

- Immédiatement ou de temps en temps (exemple: tous les soirs).
- Lorsqu'elle se fait immédiatement on parle de réplication synchrone: les copies sont synchronisées avec l''original.
- Lorsqu'elle se fait périodiquement on parle de réplication asynchrone.

Mise en Oeuvre de la réplication synchrone:
Elle nécessite l'écriture d'un trigger sur la base centrale (à PARIS) pour la table `APPAREILS`.

```
CONNECT RP/sonmotdepasse@alias_PARIS
CREATE OR REPLACE TRIGGER maj_copies_APPAREILS
BEFORE INSERT OR UPDATE OR DELETE
ON APPAREILS
BEGIN
IF INSERTING THEN
INSERT INTO APPAREILS@LIEN_Toulouse values
( :new.no_appareil,
:new.designation,
:new.Pri_uni_HT,
:new.Caracteristique_technique
);
INSERT INTO APPAREILS@LIEN_Paris values
( :new.no_appareil,
:new.designation,
:new.Pri_uni_HT,
:new.Caracteristique_technique
);
```

```
INSERT INTO APPAREILS@LIEN_Bordeaux values
( :new.no_appareil,
:new.designation,
:new.Pri_uni_HT,
:new.Caracteristique_technique
);
END IF;
IF UPDATING THEN
UPDATE APPAREILS@Lien_Toulouse SET
Designation=:new.designation,
Pri_uni_HT= :new.Pri_uni_HT,
Caracteristique_technique= :new.Caracteristique_technique
WHERE no_appareil=:new.no_appareil;
UPDATE APPAREILS@Lien_Bordeaux SET
Designation=:new.designation,
Pri_uni_HT= :new.Pri_uni_HT,
Caracteristique_technique= :new.Caracteristique_technique
WHERE no_appareil=:new.no_appareil;

UPDATE APPAREILS@Lien_Paris SET
Designation=:new.designation,
Pri_uni_HT= :new.Pri_uni_HT,
Caracteristique_technique= :new.Caracteristique_technique
WHERE no_appareil=:new.no_appareil;
END IF;
IF DELETING THEN
DELETE FROM APPAREILS@Lien_Paris where no_appareil=:new.no_appareil;
DELETE FROM APPAREILS@Lien_Bordeaux where no_appareil=:new.no_appareil;
DELETE FROM APPAREILS@Lien_Toulouse where no_appareil=:new.no_appareil;
END IF;
END;
```

Exemple 2 : Création d'une table fragmentée verticalement

```
# Sur base1
CREATE TABLE Produit1( idprod INTEGER, nomprod VARCHAR(10),
prixachat FLOAT, CONSTRAINT idprod_pk PRIMARY KEY (idprod));

# Sur base2
CREATE TABLE Produit2(idprod INTEGER, prixvente FLOAT, CONSTRAINT
idprod_pk PRIMARY KEY (idprod));

# Création de la vue
CREATE VIEW Produit
AS SELECT Produit1.idprod, Produit1.nomprod, Produit1.prixachat,
Produit2.prixvente
FROM Produit1@BASE1.RIA, Produit2@BASE2.RIA

#Création du trigger sur les 2 bases
CREATE OR REPLACE TRIGGER Tr_prod1
INSTEAD OF INSERT on Produit
FOR EACH ROW
BEGIN
     INSERT INTO Produit1@BASE1.RIA(idprod,nomprod,prixachat)
     VALUES (:New.idprod,:New.nomprod,:New.prixachat);
     INSERT INTO Produit2@BASE2.RIA(idprod,prixvente)
     VALUES (:New.idprod,:New.prixvente);
END;
```

```
#Test
INSERT INTO Produit (idprod, nomprod, prixachat, prixvente)
VALUES ('1', 'pain', '1', '2');
```

Annexe 1: Les tables systèmes d'oracle

Liste des vues statiques du dictionnaire Oracle pour l'utilisateur

Nom de la vue	synonyme	Contenu
DICTIONARY	DICT	Toutes les vues du dictionnaire ,pour le développeur ou le DBA : Nom de la vue, description
USER_TABLES	TABS	mes tables : nom, tablespace, stockage, statistiques, cluster éventuel
USER_TAB_COLUMNS	COLS	Colonnes de mes tables : Nom colonne, type, longueur, obligatoire
USER_VIEWS	-	Mes vues : Nom, texte de l'ordre SQL associé, type
USER_INDEXES	IND	Mes indexs : Nom, table indexée, unicité, stockage, statistiques
USER_IND_COLUMNS	-	Nom index, nom table, nom colonne, position et longueur
USER_CLUSTERS	CLU	Mes clusters ; Nom, stockage, statistiques
USER_OBJECTS	OBJ	Mes objets : tables, vues, indexes, clusters, synonymes, procédures, fonction, package, sequence
USER_SEQUENCES	SEQ	Mes séquences : Valeur min, max, increment, cycle, cache
USER_SYNONYMS	SYN	Mes synonymes : Nom du synonyme, de l atable, propriétaire et db link éventuel
USER_USERS	-	Caractéristiques générales du user : Nom, tablespace par defaut, tablespace temporaire
USER_CONSTRAINTS	-	De mes contraintes : Nom, type, table d'accueil, statut
USER_DB_LINKS	-	De mes database links (liens base distantes) : Nom, user distant, mot de passe, serveur distant, date de creation
USER_TAB_PRIVS	-	Des privilèges donnés ou reçus : Bénéficiaire, propriétaire, créateur
USER_EXTENTS	-	Caractéristiques de stockage de mes objets : Nom du segment, de la partition, du tablespace, taille en octets et en blocs
USER_TS_QUOTAS	-	Quota d'écriture autorisé sur les tablespace : Nom du tablespace, taille max en octets et en blocs

liste des vues statiques DBA les plus utilisées

TABLE_NAME	COMMENTS
DBA_CATALOG	All database Tables, Views, Synonyms, Sequences
DBA_CLUSTERS	Description of all clusters in the database
DBA_COLL_TYPES	Description of all named collection types in the database
DBA_COL_COMMENTS	Comments on columns of all tables and views
DBA_COL_PRIVS	All grants on columns in the database
DBA_CONSTRAINTS	Constraint definitions on all tables
DBA_CONS_COLUMNS	Information about accessible columns in constraint definitions
DBA_CONS_OBJ_COLUMNS	List of types an object column or attribute is constrained to in all tables in the database
DBA_DATA_FILES	Information about database data files
DBA_DB_LINKS	All database links in the database
DBA_DEPENDENCIES	Dependencies to and from objects
DBA_DIMENSIONS	Description of the dimension objects accessible to the DBA
DBA_DMT_FREE_SPACE	Free extents in all dictionary managed tablespaces
DBA_DMT_USED_EXTENTS	All extents in the dictionary managed tablespaces
DBA_ERRORS	Current errors on all stored objects in the database

DBA_EXTENTS	Extents comprising all segments in the database
DBA_INDEXES	Description for all indexes in the database
DBA_PARTIAL_DROP_TABS	All tables with partially dropped columns in the database
DBA_PENDING_TRANSACTIONS	information about unresolved global transactions
DBA_PROCEDURES	Description of all procedures
DBA_PROFILES	Display all profiles and their limits
DBA_ROLES	All Roles which exist in the database
DBA_ROLE_PRIVS	Roles granted to users and roles
DBA_ROLLBACK_SEGS	Description of rollback segments
DBA_RULES	Rules in the databse
DBA_SEGMENTS	Storage allocated for all database segments
DBA_SEQUENCES	Description of all SEQUENCEs in the database
DBA_SNAPSHOTS	All snapshots in the database
DBA_SYNONYMS	All synonyms in the database
DBA_SYS_PRIVS	System privileges granted to users and roles
DBA_TABLES	Description of all relational tables in the database
DBA_TABLESPACES	Description of all tablespaces
DBA_TAB_COLS	Columns of user's tables, views and clusters
DBA_TAB_COLUMNS	Columns of user's tables, views and clusters
DBA_TAB_COL_STATISTICS	Columns of user's tables, views and clusters
DBA_TAB_COMMENTS	Comments on all tables and views in the database
DBA_TAB_HISTOGRAMS	Histograms on columns of all tables
DBA_TAB_MODIFICATIONS	Information regarding modifications to tables
DBA_TAB_PRIVS	All grants on objects in the database
DBA_TRIGGERS	All triggers in the database
DBA_TRIGGER_COLS	Column usage in all triggers
DBA_TS_QUOTAS	Tablespace quotas for all users
DBA_TYPES	Description of all types in the database
DBA_UNDO_EXTENTS	Extents comprising all segments in the system managed undo tablespaces
DBA_UNUSED_COL_TABS	All tables with unused columns in the database
DBA_UPDATABLE_COLUMNS	Description of dba updatable columns
DBA_USERS	Information about all users of the database
DBA_VIEWS	Description of all views in the database

Les vues dynamiques

Pour information leur nom commence par 'V_$' et appartiennent (comme les autres) a SYS. Il existe des synonymes publics pour ces vues qui commence par un 'V$'. Elles sont surtout utiles pour la surveuillance en temps réel du système ou pour faire du tuning. Quelques vues dynamiques utiles au DBA :

V$DBFILE	Les fichiers de bd
v$database	Le nom de bd
V$PARAMETER	
V$VERSION	
V$DATABASE	
V$SESSION	
v$SGA	
V$SQL	
V$SYSSTAT	

Remarques :

- Pour obtenir une liste complète des vues statiques DBA utiliser la requete suivante :
  ```
  select * from dict  where table_name like 'DBA%'
  ```
- Idem pour les aures vues

Université du Havre, UFR Sciences, Master Prof. Info , TP1, 2016/2017, (BDD)
Enseignants : Nakechbandi M (nakechb@free.fr**)**
Thème : Implémentation d'une BD Répartie

L'objectif du TP est d'implémenter une base de données fédérée sur deux machines (éventuellement virtuelles) et de configurer le paramétrage des communications distantes des bases de données Oracle. Une première machine contient **une partie** de la base, tandis qu'une seconde machine contient **le complément** de données. Un ensemble de vues sera créé afin de reconstituer une **vision unique** de la base complète.

I/ Présentation. On considère une BD CIRQUE dont le schema conceptuel :

PERSONNEL _CIRQUE (**NOM**, ROLE) ;

NUMERO_CIRQUE (**TITRE**, NATURE, RESPONSABLE) ;

ACCESSOIRE (**NOM**, COULEUR, VOLUME, RATELIER, CAMION) ;

UTILISATION (**TITRE**, **UTILISATEUR**, **ACCESSOIRE**) ;

CIRQUE est une base de données fédérée composée de deux Bases de données CIRQUE1, CIRQUE2 :

Composition de la BD CIRQUE1 :

```
PERSONNEL_CIRQUE1, NUMERO_CIRQUE1, ACCESSOIRE1, UTILISATION_CIRQUE1.

NUMERO_CIRQUE1 = NUMERO_CIRQUE where
                 NUMERO_CIRQUE.NOM = PERSONNEL _CIRQUE1.NOM

UTILISATION_CIRQUE1 = UTILISATION_CIRQUE where
                 UTILISATION_CIRQUE.NOM = PERSONNEL_CIRQUE1.NOM
```

Composition de la BD CIRQUE2 :

```
PERSONNEL_CIRQUE2, NUMERO_CIRQUE2, ACCESSOIRE2, UTILISATION_CIRQUE2.

NUMERO_CIRQUE2 = NUMERO_CIRQUE where
                 NUMERO_CIRQUE.NOM = PERSONNEL _CIRQUE2.NOM

UTILISATION_CIRQUE2  = UTILISATION_CIRQUE where
                 UTILISATION_CIRQUE.NOM = PERSONNEL_CIRQUE2.NOM
```

La table ACCESSOIRE est une table dupliquée en CIRQUE1 et en CIRQUE2.

Le contenu de cette BD est le suivant :

PERSONNEL _CIRQUE1

NOM(40)	ROLE(20)
Clovis	Jongleur
Reine	Ecuyer
Louche	Clown
Benard	Equilibriste
Bignon	Musicien
Bordeau	Dompteur
Jerry	Clown
VASSEUR	Musicien
Fremez	Musicien
Dolores	Jongleur
Hebert	Jongleur
Sorel	Ecuyer
Sorel	Ecuyer
Loisel	Equilibriste
Jeanne	Jongleur
Sangtrespur	Dompteur

PERSONNEL _CIRQUE2

NOM(40)	ROLE(20)
VASSEUR	Joncleur
LEBRAS	Equilibriste
LEJEUNE	Ecuyer
MARMOL	Clown
OLIVIER	Equilibriste
CALMETTES	Musicien
PORCHERON	Dompteur
QUENTIN	Clown
COLIGNON	Ecuyer
DA-SILVA	Musicien

NUMERO_CIRQUE1

TITRE(30)	NATURE(20)	RESPONSABLE(20)
Les Zoupalas	Jonglerie	Clovis

Le coche infernal	Equitation	Reine
Les fauves	Clownerie	Louche
Les Smilers	Equilibre	Benard
Les Smilers	Equilibre	Clovis
les Smilers	Equilibre	VASSEUR
Les Fauves	Clownerie	Jerry
La passoire magique	Lion	Bordeau
Les Zozos	Clownerie	Jerry
Les Tartarins	Jonglerie	Demare

NUMERO_CIRQUE2

TITRE(30)	NATURE(20)	RESPONSABLE(20)
Les Fauves	Clownerie	LEBRAS
Les Zoupalas	Jonglerie	VASSEUR
Les Zoupalas	Lion	LEJEUNE
Le coche infernal	Equitation	MARMOL
Les fauves	Clownerie	OLIVIER
Les Smilers	Equilibre	CALMETTES
Les Smilers	Lion	PORCHERON
les Smilers	Equilibre	COLIGNON
Les Fauves	Clownerie	DA-SILVA

NUMERO_CIRQUE = NUMERO_CIRQUE1 Union NUMERO_CIRQUE2

ACCESSOIRE

NOM(30)	COULEUR(10)	VOLUME	RATELIER	CAMION
Ballon	Rouge	0.3	15	5
Barre	Blanc	0.6	19	5
Fouet	Marron	0.2	11	3
Bicyclette a elephant	Vert	0.4	27	8
Trompette	Rouge	0.2	2	1
Cercle	Magique Orange	0.2	1	1
Boule	Cristal	0.2	88	8
Cage a lions	Noir	10.0	0	2
Chaise longue de lion	Bleu	0.9	11	5
Peigne de chimpanze	Jaune	0.2	23	3
Etrier				

UTILISATION_CIRQUE1

TITRE(30)	UTILISATEUR(20)	ACCESSOIRE(30)
Les Zoupalas	Dolores	Ballon
Les Zoupalas	Hebert	Ballon
Les Zoupalas	Dolores	Barre
Le coche infernal	Jeanne	Bicyclette a elephant
Le coche infernal	Sorel	Fouet
Les fauves	Jerry	Trompette
Les Smilers	Benard	Cercle magique
Les Smilers	Benard	Boule

UTILISATION_CIRQUE2

TITRE(30)	UTILISATEUR(20)	ACCESSOIRE(30)
Les Smilers	VASSEUR	Bicyclette a elephant
La passoire magique	LEBRAS	Cage a lions
La passoire magique	DA-SILVA	Chaise longue de lion
Les Zoupalas	LEJEUNE	Ballon
Le coche infernal	MARMOL	Ballon
Le coche infernal	OLIVIER	Barre
Les fauves	CALMETTES	Bicyclette a elephant
Les Smilers	LEJEUNE	Fouet
Les Zoupalas	MARMOL	Trompette
Le coche infernal	OLIVIER	Cercle magique

UTILISATION_CIRQUE = UTILISATION_CIRQUE1 Union UTILISATION_CIRQUE2

II/ Travail à faire : **Ce travail est à rendre par binôme.** Rappelons que dans ce tp vous aller travailler sur **deux machines** (qui prouvent être virtuelles ou réel) sachant que la suite on va indiquer que les étapes nécessaires pour créer les machine virtuelles. …/…

Exercice A. L'objectif de cet exercice est de réaliser et configurer deux machines avec deux SGBDs Oracle. Rappelons que vous allez travailler par binôme sur deux machines réels ou virtuelles (ORD1, ORD2).

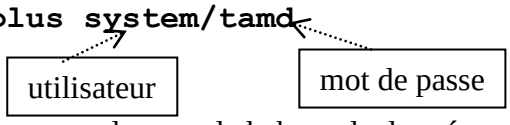
On commence par l'installation un SE Windows XP puis le logiciel de SGBD ORACLE sur la machine ORD1 puis sur la machine ORD2.

A1/ Sous linux ubuntu (sous Windows : même manipulation) :
1/ Récupérer le logiciel VM VirtualBox (https://www.virtualbox.org/) puis l'installer sur votre machine réel.
2/ Récupérer les deux fichiers iso de CD d'installation de Windows XP et de l'installation du **serveur** oracle 11_express (sur une clé usb que je fournie).
2/ Créer deux machines virtuelles window_xp avec au moins 2 à 3 gigas de mémoire ram chacune et un disque virtuel de 10 Gigas . Attribuer à ces deux machines deux noms distincts (ORD1, ORD2). Changer les adresses MAC lors de la création. (Les deux adresses mac ne doivent pas être identiques).

A2/ **Un fois windows est installé, sous les deux (virtuelles) machines Windows** : Attribuer deux noms distincts aux deux machines (par exemple win1, win2). Puis Configurer l'environnement réseaux pour que les deux machines puissent **communiquer**. Noter les **ip** attribués automatiquement à ces deux machines, les modifier en cas où elles sont identique : Tester en tapant dans une fenêtre de la commande : `ipconfig/all` puis `ping win1` (de win2 vers win1). **Indication** : Désactiver le pare-feu (Dans le pare-feu Windows).

A3/ A partir des fichiers iso de CD d'installation d'oracle (Récupérer en question A1.2) : Installer le **serveur** oracle 11_express sur win1 (ord1) puis sur win2 (ORD2) (ORD1 sera le premier serveur, ORD2 sera le deuxième serveur). Le mot de passe de l'administrateur d'Oracle sera tamd (pour tout le monde et pour tous les comptes). Ce mot de passe sera attribué aux deux comptes d'administrateur de la base de données (sys, system) qui sont créés automatiquement par Oracle. (On peut envisager un troisième serveur d'oracle !).

A4/ Vérifier que le serveur d'Oracle est bien installé en tentant une connexion. **Indication :** Sous une fenêtre de commande de ORD1 (Win1) taper : **`sqlplus system/tamd`**



A5/ Après la connexion au serveur d'Oracle, trouver et noter le nom de la base de données actuelle :
Indication : `SELECT name FROM v$database;`

Noter le nom de la base de données : Normalement "XE"

A6/ Maintenant :

- Sur la machine serveur ORD1, vérifier que le processus "listener" (le nom de ce processus est tnslsnr) est actif (est en court d'exécution) : Gestionnaire de taches (controle + alt + sup) ... tnslsnr. Vérifier également que les paramètres dans le fichier d'initialisation **`listener.ora`** sont corrects (voir le model dans l'annex2). Si non, il faut corriger ce fichier puis relancer le "lestener".
- Sur la machine client ORD2 : éditer le fichier `tnsnames.ora` puis vérifier que les paramètres d'accès à au serveur d'oracle à distance sont corrects (voir le model dans l'annexe2). Si non, il faut les corriger. *On peut passer par l'interface graphique d'Oracle : Menu démarrer → Onglet tous les programmes → oracle Client11g_home → outil de configuration → assistant de configuration oracle net → configurer une methode de resolution de nom → ...*

A7/ A partir de la machine (ORD2) tenter une connection sqlplus au serveur (ORD1) Indication **:** **`sqlplus system/tamd@BD1.`** (Pour comprendre le role du paramter **`"BD1"`** voir annexe 2).

Noter dans votre compte-rendu les difficultés rencontrées pour configuration client/serveur.

Exercice B. Les fichiers et les processus d'oracle : Certaines questions de cette partie nécessites un privilège dba. Donc ouvrir une session oracle de type dba comme **`sys`** ou **`system`** (`sqlplus system/tamd`).

Sous SQLPLUS, pour obtenir une trace de tout ce qui s'affiche à l'écran, utiliser la commande :
spool nom_de_fichier.txt ceci va faciliter la rédaction de compte rendu

B1/ Trouver le nom (ou le IP de la machine de SGBD. **indications :**

```
select distinct machine FROM v$session;
select distinct HOST_NAME from v$instance;
```

B2/ Lancer puis interpréter les requetes ci-dessous :

- `select VERSION from v$instance;`
- `select STARTUP_TIME from v$instance;`
- `select username FROM v$session;`
- `desc DBA_USERS; select username FROM DBA_USERS;`

B3/ Trouver les fichiers d'archivage. Indication : `select member FROM v$logfile;` Voir polycopié cours page 9.

B4/ Trouver les fichiers de contrôle. Indication: `SELECT name FROM v$controlfile;`

B5/ Trouver les fichiers de données de la base de données actuelle.
Indication : consulter la table système `V$DBFILE.(desc V$DBFILE;` puis `select name ... ;)`

B6/ Trouver les emplacements de fichiers de données précédent. Sont-ils accessible ? pourquoi ?

B7/ Consulter la table `V$PARAMETER` puis analyser son contenu

Indication: describe `V$PARAMETER`, puis `SELECT … FROM V$PARAMETER`

B8/ Donner l'ensemble de processus système relatifs à oracle.

Indication : Sous Windows consulter la liste des taches (`cont+alt+sup`) (certains taches sont des threads comme : `DBWR, LGWR, SMON, SMON , PMON,...` pour les afficher lancer les equêtes : `select name from v$bgprocess;` puis `select program from v$session;` On peut combiner les deux requêtes en une seule avec jointure. On peut aussi les afficher en passant par le menu démarrer de Windows → oracle--- →outils de configuration--- → oracle assistant administration--- →oracle managed object→ computer→nom de la base de données → information sur les process... On trouve entre autre les processus (certain sont threads sous Windows) : `DBWR, LGWR, SMON, SMON , PMON...` (Sous Unix (Linux) on tape : `ps -aux | grep oracle | grep votre_nom`).

Exercice C. Création des comptes (des utilisateurs**) Oracle** . Ce travail est à faire uniquement sur la machine `ORD1`.**Indication :** On peut exécuter le scripte SQL ci-dessous :

```
create user U1 identified by U1;
grant CONNECT, RESOURCE to U1;
create user U2 identified by U2;
grant connect, resource to U2;
```

C.1/ Utiliser les scriptes disponibles sur `193.48.166.225 --> m2`
ou sur (`http://nakech.free.fr/creer_cirque1.sql`),
(`http://nakech.free.fr/creer_cirque2.sql`)
Pour créer sur chacune des deux machines la structure des quatre tables de la base en intégrant les contraintes d'intégrité (local) nécessaires et notamment les références (clé étrangère).

C.2/ Peupler CIRQUE1 et CIRQUE2

Remarque : Oracle ne gère pas automatiquement les références décrites dans les scriptes de création. La solution qu'on va adopter dans un premier temps est de **corriger les erreurs** de références entre les deux Bases. On verra plus tard comment **Oracle assure l'intégrité référentielle des BDD avec l'utilisation des «trriger».**

C.3/ Peupler les tables de la BD CIRQUE1 en exécutant le scripte insertion_cirque1.sql qui est disponible sur **http://193.48.166.225/m2/ (ou sur mon site frée)**

C.4/ Idem peupler les tables de la BD CIRQUE2 en utilisant les données décrites dans les tables plus hautes. (penser aux contraintes d'intégrités en local). **…/…**

Exercice D/ L'objectif de cet exercice est de réer et tester un lien entre deux BDs une locale l'autre distante. Rappelons que le fichier tnsnames.ora décrit les différentes BD distantes qui sont accessibles par la BD locale. Ce fichier est dans le répertoire ORACLE_HOME/network/admin

Dans la suite on suppose que les deux machines sont **alO4-07** et **alO4-06**, l'utilisateur "oracle" de la première machine est **aurelien** , et l'utilisateur "oracle" de le deuxième machine est **arnaud.**

Arnaud est un utilisateur oracle de la base 'BD_AURLIEN' sa création peut se faire par la commande suivante :

```
create user arnaud identified by mot_passe
```

Pour donner à Arnaud les droits de connection et de pouvoir créer des tables :

```
grant connect, ressource to arnaud;
```

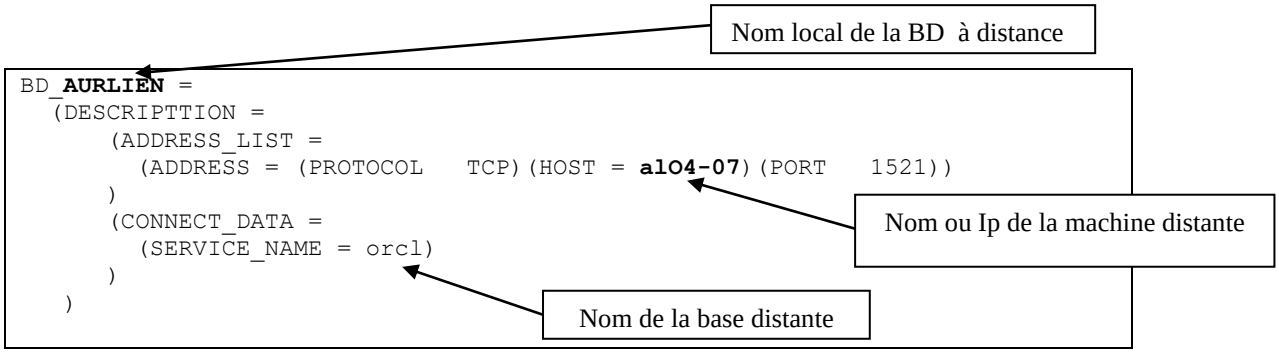
Et pour donner à Arnaud le droit d'utiliser les tables de l'utilisateur Aurlien :

```
grant select on aurlien.personnel to arnaud;
```

…

D.1/ (Travail à faire par *arnaud*) Éditer le ficher **`tnsnames.ora`** puis **ajouter l**e bloc ci-dessous **(voir annexe pour un exemple complet)** :

```
/usr/local/oracle/product/11g/network/admin/tnsnames.ora
```



Cet ajout permet à *arnaud* d'acceder à la BD de `AURLIEN` (par la creation des alias).

D.2/ Idem, aurelien doit également configurer l'accès à la BD distante de arnaud

D.3/ Créer un liens entre la BD locale et la BD distante.

Command SQL :

```
create database link <nom> connect to <utilisateur de la base cible>
          identified  by <mdp dans DB cible> using <nom de la base cible>.
```

`<nom de la base cible>` doit être défini comme alias dans le fichier **`tnsnames.ora`** décrit plus haut.

La commande de création de lien que *arnaud* doit faire est donc :

```
create database link lien_a connect to arnaud
                 indentified by mot_passe using 'BD_AURLIEN' ;
```

D.4/ Tester ce lien par les requetes SQL suivantes :

```
Desc dba_db_links;
```

```
select * from dba_db_links;
```

```
select owner, db_link, username from dba_db_links;
```

```
     select * from aurlien.personnel@lien_a;
```

D.5/ Essayer d'afficher la structure d'une table de la base distante

(`desc aurlien.personnel@lien_a )`

Exercice E/ Le but de cet exercice est de créer une vue unique permettant de d'afficher l'ensemble de données repartie (sur les deux machines) d'une table comme une table unique. **La création des vues passe par les étapes suivantes :**

E.1/ Rendre le lien qu'on vient de créer transparent par la création d'un synonyme :

```
      create synonym personnel_distant for  aurlien.personnel@lien_a;
```

E.2/ Une requête de création de vue pleuvant être donc :

```
create view tout_personnel as
select * from personnel  union  select * from personnel_distant ;
```

E.3/ Vérifier : select * from tout_personnel

E.4/ Créer et tester les autres vues.

Exercice F/ Le but de cette exercice est de faire quelques requêtes SQL pour manipuler la BDD.

F.1/ Tester une mise à jour directement sur le lien et sur la table local

```
update  aurlien.accessoire@lien_a set volume = volume + 0.1 ;
…
```

Vérifier sur la machine local et sur la machine à distance (Ne pas oublier les **commit**)

F.2/ Tester une mises à jour sur une vue totale . Votre conclusion ?

F.3/ Modifier la structure de la table accessoire en ajoutant une contrainte supplémentaire

```
alter accesoire (constran volume_ck (volume <=10 )) ;
alter aurlien.accessoire@lien_a (constran volume_ck (volume <=10 )) ;
```

On notera qu'Oracle n'autorise pas les commandes CREATE, ALTER et DROP de définition de données distantes.

F.4/ Tester maintenant des mises à jour violant les contraintes précédentes

Exercice G/ Créer quelques requêtes à base de jointures et/ou de requêtes imbriques, par **exemples :**

- G1. Afficher les personnels qui utilisent l'accessoire Ballon ;
- G2. Afficher les accessoires utilisés par les Jongleurs ;
- G3. Trouver les noms de personnels assurant au moins deux numéros
- G4. Afficher le nombre des numéros utilisant des accessoires rouges
- G5. Donner le nombre de personnels jouant chaque rôle
- G6. Donner le nombre des numéros assures par chaque personne.

CR : Dossier à rendre par Binôme par mail au `nakechb@free.fr` avec comme objet

`m2BDD_tp1_votre_nom`.

Le CR comportant :

- Toutes les requêtes utilisées avec un échantillonnage significatif d'exécution.
- Vos remarques et commentaires à chaque fois où il y a une nouvelle notion abordé (la configuration des machines virtuelles (ou réels) en réseaux, la configuration des Oracles distribués, …).

Date limite : Avant mon prochain cours le 4/1/2017.

Pour faciliter la rédaction de votre CR, le sujet est disponible sur : `http://193.48.166.225/m2`
`Ou sur : http://nakech.free.fr/m2tp1.pdf`

Annexe 1 : Création de la base de données

(Les scriptes sont disponibles à l'url : **http://nakechb.free.fr/M2_SIRES)**

```
create table personnel (nom varchar(40),role varchar(20),
               constraint personnel_pri primary key(nom));

create table numero (titre varchar(30),nature varchar(20),responsable varchar(40),
               constraint numero_pri primary key(titre),
               constraint numero_etr foreign key(responsable) references personnel(nom));

create table accessoire (nom varchar(30),couleur varchar(10),
                        volume number(4,1),ratelier number(2),camion number(1),
                        constraint accessoire_pri primary key(nom));

create table utilisation (titre varchar(30),utilisateur varchar(40),
               accessoire varchar(30),
               constraint utilisation_pri primary key(titre, utilisateur, accessoire),
               constraint utilisation_et1 foreign key(titre) references numero(titre),
               constraint utilisation_et2 foreign key(utilisateur) references personnel(nom),
               constraint utilisation_et3 foreign key(accessoire) references  accessoire(nom));
```

Saisie sur machine 1

```
insert into personnel values ('Clovis','Jongleur');
…
insert into numero values ('Les Zoupalas','Jonglerie','Clovis');
…
```

Saisie sur machine 2

```
insert into personnel values ('LEBRAS', 'Joncleur') ;
…
insert into numero values ('Les Fauves', 'Clownerie', 'LEBRAS');
```

Annexe2 : Rappel sur SQLPLUS-Oracle

Notion de transaction en base de données

Une transaction est un ensemble de modifications de la base qui forme un tout indivisible. Il faut effectuer ces modifications entièrement ou pas du tout, sous peine de laisser la base dans un état incohérent. Les Systèmes de Gestion de Bases de Données permettent aux utilisateurs de gérer leurs transactions. Ils peuvent à tout moment :

- Valider la transaction en cours par la commande COMMIT. Les modifications deviennent définitives et visibles à tous les utilisateurs.
- Annuler la transaction en cours par la commande ROLLBACK. Toutes les modifications depuis le début de la transaction sont alors défaites.

En cours de transaction, seul l'utilisateur ayant effectué les modifications les voit. Ce mécanisme est utilisé par les systèmes de gestion de bases de données pour assurer l'intégrité de la base en cas de fin anormale d'une tâche utilisateur : il y a automatiquement ROLLBACK des transactions non terminées.

Rappel de quelques instructions SQL utile pour ce TP :

operateur UNION **:**

```
      SELECT ...
      UNION  (SELECT ...)
```

operateur MINUS **:**

```
      SELECT ...
            MINUS  (SELECT ...)
```

Mémento SQL*PLUS

- save filename.sql : sauve le contenu du buffer dans un fichier de nom filename.sql
- get filename.sql : charge le buffer avec le contenu du fichier de nom filename.sql
- start filename : charge le buffer et lance l'exécution du fichier script sql
- run : lance l'exécution du contenu du buffer
- spool filename.txt : copie la sortie écran sur le fichier filename.txt
- spool off : suspend l'opération précédente
- help commande : pour obtenir de l'aide sur la commande donnée en argument

Variables d'environnement (Mise en page SQLPLUS)

- set long 1024 : pour voir la totalité des définitions de vues
  (exemple : set long 1024 ; select * from USER_VIEWS ;)
- set pagesize 20 : formate la sortie écran par blocs de 20 lignes
- set pause on : ne visualise la sortie qu'après un 2e RC - bloc par bloc -
-
- set timing [on|off] : active ou désactive le chronomètre.

Quelques Vues de "méta-base"

(dictionnaire de données ou tables système) **décrivant les objets utiles pour ce TP :**

- desc[ribe] tablename : donne le schéma de la relation tablename
- all_catalog : table système donnant toutes les tables accessibles
- user_catalog : table système donnant les seules tables du USER
- all_objects : table système donnant tous les objets accessibles
- user_objects : table système donnant les seuls objets du USER
- user_sys_privs : table système donnant les privilèges système du USER
- user_tab_privs : table système donnant les privilèges sur les objets accessiles

- USER_TABLES : Description des tables créées par l'utilisateur.
- USER_TAB_COLUMNS : Description des colonnes de chaque table ou vue créée par l'utilisateur courant. Chaque ligne de la vue col décrit une colonne.
- USER_TAB_COMMENTS :Commentaires sur les tables et les vues créés par l'utilisateur.
- USER_TAB_GRANTS_MADE : Droits sur ses objets, tables ou vues, donnés par l'utilisateur.
- USER_USERS :Informations sur l'utilisateur courant.
- USER_VIEWS :Texte des vues créées par l'utilisateur.

Exemples :

```
desc all_db_links;
SELECT * FROM all_db_links;
```

- Pour obtenir une liste complète des vues statiques DBA utiliser la requete suivante :

```
desc  dict ;
select * from dict where table_name like 'DBA%'
```

- Synonymes **:** Les noms des vues étant assez longs, les synonymes suivants ont étés définis :
  COLS : Synonyme pour USER_TAB_COLUMNS
  MYPRIV : Synonyme pour USER_USERS
  TABS : Synonyme pour USER_TABLES
  OBJ : Synonyme pour USER_OBJECTS
  SYN : Synonyme pour USER_SYNONYMS
  cat : synonyme pour user_catalog

Exemples :

la requete : `select * from tabs;` Affiche une liste des tables de l'utilisateur.
la requete : `select * from SYN` …

Annexe 3 : Exemples de fichiers de configuration

`listener.ora` Ce ficher gère les demande d'accès à distance.

```
#   /usr/local/oracle/product/11g/network/admin/listener.ora

LISTENER =
 (DESCRIPTION_LIST =
  (DESCRIPTION =
   (ADDRESS = (PROTOCOL = IPC)(KEY = EXTPROC1521))
    (ADDRESS = (PROTOCOL = TCP)(HOST = 172.16.21.184)(PORT = 1521))
  )
 )
SID_LIST_LISTENER =
 (SID_LIST =
  (SID_DESC =
   (SID_NAME = orcl)
  )
 )
ADR_BASE_LISTENER = /usr/local/oracleADR_BASE_LISTENER = /usr/local/oracle
```

Nom ou ip de la machine serveur

Nom de la base serveur

Un autre exemple du fichier **`listener.ora`**

```
#   /usr/local/oracle/product/11g/network/admin/listener.ora

LISTENER =
  (DESCRIPTION_LIST =
    (DESCRIPTION =
      (ADDRESS_LIST =
        (ADDRESS = (PROTOCOL = IPC)(KEY = EXTPROC0))
      )
      (ADDRESS_LIST =
        (ADDRESS = (PROTOCOL = TCP)(HOST = LIH-NAKECH)(PORT = 1521))
      )  )  )

SID_LIST_LISTENER =
  (SID_LIST =
    (SID_DESC =
      (SID_NAME = PLSExtProc)
      (ORACLE_HOME = /oracle/ora92)
      (PROGRAM = extproc)
    )
    (SID_DESC =
      (GLOBAL_DBNAME = ora1)
      (ORACLE_HOME = /oracle/ora92)
      (SID_NAME = ora1)))
```

Nom ou ip de la machine serveur

Nom de la base serveur

select name from v$database;

```
set linesize 121
col name format a40
col value format a40
```

Exemple complet de 2 fichiers tnsnames.ora (site1 site2)

```
# site 1  (de Pascal)

# tnsnames.ora
# Network Configuration File: /oracle/product/11g/network/admin/tnsnames.ora
# Generated by Oracle configuration tools.
LISTENER_ORCL =
(ADDRESS = (PROTOCOL = TCP)(HOST = 172.16.21.180)(PORT = 1521))
ORCL =
(DESCRIPTION =
(ADDRESS = (PROTOCOL = TCP)(HOST = 172.16.21.180)(PORT = 1521))
(CONNECT_DATA =
(SERVER = DEDICATED)
(SERVICE_NAME = orcl)
))
ORCL_MICHEL = (DESCRIPTION =
(ADDRESS_LIST =
(ADDRESS = (PROTOCOL = TCP) (HOST = 172.16.21.186) (PORT = 1521))
)
(CONNECT_DATA =
(SERVICE_NAME = orcl)
))
```

Nom local de la BD à distance

Ip de la machine distante

Seulement la partie **en gras** est à ajouter sur le fichier tnsnames

```
# site 2  (de MICHEL)

# tnsnames.ora Network Configuration File:
/oracle/product/11g/network/admin/tnsnames.ora
# Generated by Oracle configuration tools.
LISTENER_ORCL =
(ADDRESS = (PROTOCOL = TCP)(HOST = 172.16.21. 186)(PORT = 1521))
ORCL =
(DESCRIPTION =
(ADDRESS = (PROTOCOL = TCP)(HOST = 172.16.21. 186)(PORT = 1521))
(CONNECT_DATA =
(SERVER = DEDICATED)
(SERVICE_NAME = orcl)
))
ORCL_Pascal = (DESCRIPTION =
(ADDRESS_LIST =
(ADDRESS = (PROTOCOL = TCP) (HOST = 172.16.21.180) (PORT = 1521))
)
(CONNECT_DATA =
(SERVICE_NAME = orcl)
))
```

Nom local de la BD à distance

Ip de la machine distante

Commentaire :

- Alias (ORCL_MICHEL) est entré dans le fichier de configuration tnsnames.ora du site1 (site de pascal). Il permet de référencer la base de données orcl hébergée sur la machine site2 ayant pour adresse IP 172.16.21.186. Aussi, cet alias permet de se connecter à cette base distante en utilisant le port "1521".

- De même pour que le site2 se connecte au site1, on procède de la même manière sauf il faut changer l'alias (ORCL_Pascal) et les adresses IP sur le fichier tnsnames.ora

```
CREATE OR REPLACE PROCEDURE raise_salary
(v_id in emp.empno%TYPE)
IS
BEGIN
   UPDATE emp
   SET sal = sal * 1.10
   WHERE empno = v_id;
END raise_salary;
/
```

L'exemple ci-dessus montre une procédure contenant un paramètre IN. L'exécution de cette instruction dans SQL*Plus crée la procédure RAISE_SALARY. Une fois appelée, RAISE_SALARY prend le paramètre du numéro d'employé et met à jour l'enregistrement relatif à l'employé et lui affecte une augmentation de 10 pour cent.

Pour appeler une procédure dans SQL*Plus, on utilise la commande EXECUTE :

```
SQL> EXECUTE raise_salary (7369)
```

```
CREATE OR REPLACE PROCEDURE query_emp
(v_id IN emp.empno%TYPE,
              v_name OUT emp.ename%TYPE,
              v_salary OUT emp.sal%TYPE,
               v_comm OUT emp.comm%TYPE)
IS
BEGIN
 SELECT ename, sal, comm
 INTO v_name, v_salary, v_comm
    FROM emp  WHERE empno = v_id;
END query_emp;
/
```

```
SQL> START emp_query.sql
             Procedure Procedure created created.

SQL> VARIABLE g_nameVARCHAR2(15)
SQL> VARIABLE g_sal NUMBER
SQL> VARIABLE g_commNUMBER
SQL> EXECUTE query_emp(7654,:g_name,:g_sal,:g_comm)
     PL/SQL procedure procedure successfully successfully completed

SQL> PRINT g_name g_sal g_comm
```