TP1

Exercice 1 : Propriétés des bases réparties

Sites de bases de données

1) Sur chaque site, créer un utilisateur avec les droits suffisants.
   - Sur le site1 : utilisateur bdr1. Mot de passe : bdr1

```
CREATE USER BDRE1 IDENTIFIED BY bdr1;
GRANT ALL PRIVILEGES TO BDRE1;
-- oubien
GRANT connect, resource, create view, create database link, create snapshot TO bdr1;
```

   - Sur le site2 : utilisateur bdr2. Mot de passe : bdr2

```
CREATE USER BDRE2 IDENTIFIED BY bdr2;
GRANT ALL PRIVILEGES TO BDRE2;
```

Liens de bases de données

Pour interroger une base distante,il faut créer un lien de base de données.
Un lien de base de données est un chemin unidirectionnel d'un serveur à unautre.
Un client connecté à une BDA, peut utiliser un lien stocké dans la BD A pour accéder à la BD distante B, mais les utilisateurs connectés à B ne peuvent pas utiliser le même lien pour accéder aux données sur A. Lorsqu'un lien est référencé par une instruction SQL, Oracle ouvre une session dans la base distante et y exécute l'instruction.

2) Créer dans le site 1 un lien vers l'autre site en utilisant la syntaxe :

```
CREATE [SHARED|PUBLIC|PRIVATE] DATABASE LINK Nom_du_Lien
CONNECT TO { CURRENT_USER | User IDENTIFIED BY password} USING connect_string;
```

Exemple :

```
create public database link site1tosite2 connect to bdre2 identified by "2222" using
'192.168.56.2:1521/xe';
```

3) Tester le bon fonctionnement du lien

4) Faire de même pour le site 2

5) Consulter la liste des liens déjà crées (utiliser la table dba_db_links du dictionnaire de données)

Transparence de localisation

Pour assurer une propriété de la répartition des bases de données qui est la transparence vis-à-vis de la localisation, Oracle utilise des synonymes dont le rôle est de masquer le nom du lien de base de

données. La syntaxe de création est :

```
CREATE OR REPLACE [PUBLIC] SYNONYM nom_du_synonyme
FOR [schéma.]nom-objet[@Nom_du_Lien] ;
```

6) Créer une table table_s2 sur le site 2

7) Sur le site1, on va créer un synonyme noté table_s2 pour cette table sur le site1

```
create or replace public synonym table_s2 for table_s2@site1tosite2;
```

Transparence de Fragmentation

La commande COPY de SQL*Plus permet de copier des données entre deux SGBD
En général on exécute cette commande sur la machine où réside la base de données.
La syntaxe est la suivante :

```
COPY {FROM database| TO database| FROM databaseTO database}
{APPEND|CREATE|INSERT|REPLACE} destination_table[(column, column, column, ...)]
USING query
```

database a la syntaxe suivante : username[/password]@connect_identifier

- APPEND : si la table n'existe pas(CREATE+INSERT) sinon (INSERT)
- CREATE : si la table n'existe pas (CREATE+INSERT) sinon (erreur)
- REPLACE : si la table n'existe pas (CREATE+INSERT) sinon (DROP+CREATE+INSERT)
- INSERT : si la table n'existe pas (ERREUR) sinon (INSERT)

La commande COPY n'exporte pas les contraintes (sauf NOT NULL)

8) Créer un exemple de fragment sur bdr2 en utilisant la commande Copy

Une des propriétés des bases de données réparties est la transparence à la fragmentation.
Ainsi,mêmefragmentés, les enregistrements doivent apparaître comme sur un seul site. Pour ceci, on utilise les vues :

```
CREATE VIEW nom_vue[(nom_col1,...)]
AS SELECT ...
[WITH CHECK OPTION];
```

Certaines vues peuvent être l'objet de mise à jour par les instructions INSERT, UPDATE, DELETE
Dans le cas contraire, on fait appel aux déclencheurs INSTEAD OF

9) Tester le déclencheur suivant

```
CREATE TRIGGER tr_emp
INSTEAD OF INSERT ON emp
FOR EACH ROW
BEGIN
        INSERT INTO copie_empVALUES (:new.NO, :new.nom, :new.n_dept,:new.age);
END;
```

Transparence de duplication

La première option consiste à dupliquer régulièrement les données sur le serveur local.
Duplication synchrone : diffuser immédiatement les modifications apportées aux données sources vers les copies

- Utilisation des TRIGGER

Duplication asynchrone : diffuser les modifications apportées aux données sources vers les copies à des intervalles prédéfinis.

- Utilisation de SNAPSHOT (clichés) ou Mateliarised view(vues matérialisées)

Syntaxe :

```
CREATE SNAPSHOT nom_snapshot
[REFRESH FAST | COMPLETE | FORCE]
START WITH date_de_debut_de_synchronisation
NEXT date_de_la_prochaine_synchronisation
AS requéte_select;
```

FAST : Le mode rapide permet de faire un rafraîchissement en tenant compte seulement des mises à jour effectuées sur le site Maître.

- Un SNAPSHOT LOG doit être crée pour la table Maître afin de noter les différents changements subvenus qui seront répercutés sur le snapshot : CREATE SNAPSHOT LOG ON nom_de_la_table;

COMPLETE: à chaque rafraîchissement, toute la table est transférée.

- Ce mode est obligatoire pour les snapshots complexes
- Un snapshot est dit simple si la requête SELECT ne contient pas d'opération ensemblistes ni de clause ORDER BY. Un snashot est dit complexe dans le cas contraire

FORCE : Un rafraîchissement rapide est d'abord tenté; s'il ne marche pas le rafraîchissement complet est effectué.

Pour modifier un cliché :

```
ALTER materialized view nom_snapshot
[REFRESH FAST | COMPLETE | FORCE]
START WITH date_de_debut_de_synchronisation
NEXT date_de_la_prochaine_synchronisation;
```

10) Créer un cliché de la table emp du site 2 sur le site 1 avec un mode de rafraichissement rapide.

11) Vérifier le bon fonctionnement du cliché créé.

Exercice 2 : Fragmentation horizontale

Le schéma relationnel de la base répartie est le suivant :

- Achat(nb, nboisson, date, lieu) sur le site S1
- BuveursBig(nb, nomb, prenomb, type) qui est beaucoup plus volumineuse est placée sur le site S2
  Le domaine de l'attribut type d'un buveur est {'rare', 'petit', 'moyen', 'gros'}

Tous les buveurs de la relation BuveurBig sont fragmentés horizontalement en deux fragments selon le type du buveur :

- le fragment BuvRare contient les buveurs dont le type est 'rare',
- le fragment BuvAutre contient les autres buveurs.

L'allocation des fragments est la suivante :
sur S1 : le fragment BuvAutre
sur S2 : le fragment BuvRare

1) Exécuter les ordres SQL pour créer les fragment BuvAutre sur S1 et BuvRare sur S2.

2) Définir sur S1 la vue VueBuveurs qui représente l'union des fragments BuvRare et BuvAutre.

3) Soit la requête accédant à la vue des buveurs :

```
select count(*)
from VueBuveurs
where type = 'gros';
```

Analyser le plan d'exécution. Le plan d'exécution est-il optimal ?

4) On suppose dans la relation Achats, le plus grand numéro de buveurs est 100. Quelle est la cardinalité du résultat de la requête ?

```
select * from Achats a, BuveursBig b
where a.nb = b.nb and a.nb > 100
```

5) Comparer les temps d'exécution des 3 requêtes suivantes.

```
R1 :
select * from S2BUVEURSBIG b, ACHATS a
where a.nb = b.nb and a.nb > 100

R2:
select * from ACHATS a, S2BUVEURSBIG b
where a.nb = b.nb and a.nb > 100 ;

R3:
select * from S2BUVEURSBIG b, ACHATS a
where a.nb = b.nb and b.nb > 100;
```

Remarques :

1. Exemple pour mesurer le temps d'exécution d'une requête
   ```
   set timing on
   select  ... from ...
   where ...
   ```
2. Exemple pour analyser le plan d'exécution d'une requête :
   ```
   set timing off
   set autotrace trace
   select ... from ...
   where ....
   ```