

Cours de « Bases de données Réparties » : CHAPITRE 4

RÉPLICATION DES DONNÉES RÉPARTIES

Présenté par : Ahmad OUTFAROUIN

Email : ah.outfarouin@gmail.com

OBJECTIFS

A LA FIN DE CE CHAPITRE, VOUS SAUREZ :



Introduction (Définition, objectifs, avantages et inconvénients).



Transparence (Localisation, fragmentation).



Types de réplication.



- La **réplication** permet de copier les informations d'une BD à une autre base.
- La réplication **peut** être accompagné d'une **transformation des données** sources.
- La réplication permet **d'augmenter** la **disponibilité** et la **fiabilité** des **bases de données**.
- Dans tous les cas la **réplication** est **une redondance d'information**.

- **L'amélioration des requête** sur les données.
- La **disponibilité des données**.
- Les **lectures** sont **exécutés** sur la **copie** la **plus proche** de l'utilisateur.
- Pour la disponibilité **on ne dépend pas du site central unique**.
- Possibilité de **faire la transaction** même si le site est indisponible, **sur une autre copie**.

- **Améliorer les performances.**
- **Utilisation de la copie la plus proche** du client => évite des transferts inutiles.
- **Augmenter la disponibilité des données.**
- Lors d'une **panne** d'un serveur, on peut **se replier sur un autre** disposant d'une copie des données.
- Avec N copies sur des serveurs différents => **Disponibilité = $1 - \text{probabilité_panne}^n$**

- **Assurer la convergence des copies.**
- **Offrir une transparence de gestion aux clients** : les clients doivent croire à l'existence d'une seule copie.
- Le SGBD doit assurer la **diffusion des mises à jour** aux copies et le **choix de la meilleure copie** lors des accès.

- Le concept d'**indépendance à la localisation** peut être traité avec des **synonymes**.

- **Exemple**

```
CREATE SYNONYM matable  
FOR monfragment@mondblink;
```

- Le fragment peut alors être utilisé sans connaître sa localisation.

- **Exemple**

- ❏ Sur le premier site:

- On suppose qu'il existe une table nommée « perso » sur le site 2.
 - Sur le site 1, on va créer un synonyme noté « perso » pour cette table sur le site 1

```
CREATE OR REPLACE PUBLIC SYNONYM perso FOR  
perso@site1TOsite2;
```

- Pour tester:

```
SELECT *FROM perso;
```


- **Exemple (Suite)**

- Sur le deuxième site:

- On suppose qu'il existe une table nommée « profession » sur le site 1.
 - Sur le site 2, on va créer un synonyme noté « profession » pour cette table sur le site 1

```
CREATE OR REPLACE PUBLIC SYNONYM profession FOR  
profession@site2TOsite1;
```

- Pour tester:

```
SELECT *FROM profession ;
```

- ⊕ Pour supprimer un synonyme:

```
DROP SYNONYM nom_du_synonyme;
```

- **Création d'un synonyme:**

```
CREATE SYNONYM Vehicule FOR  
Vehicule@lien_Base2;
```

- **Suppression d'un synonyme:**

```
DROP SYNONYM Vehicule;
```

- **Synonyme d'une séquence distante :**

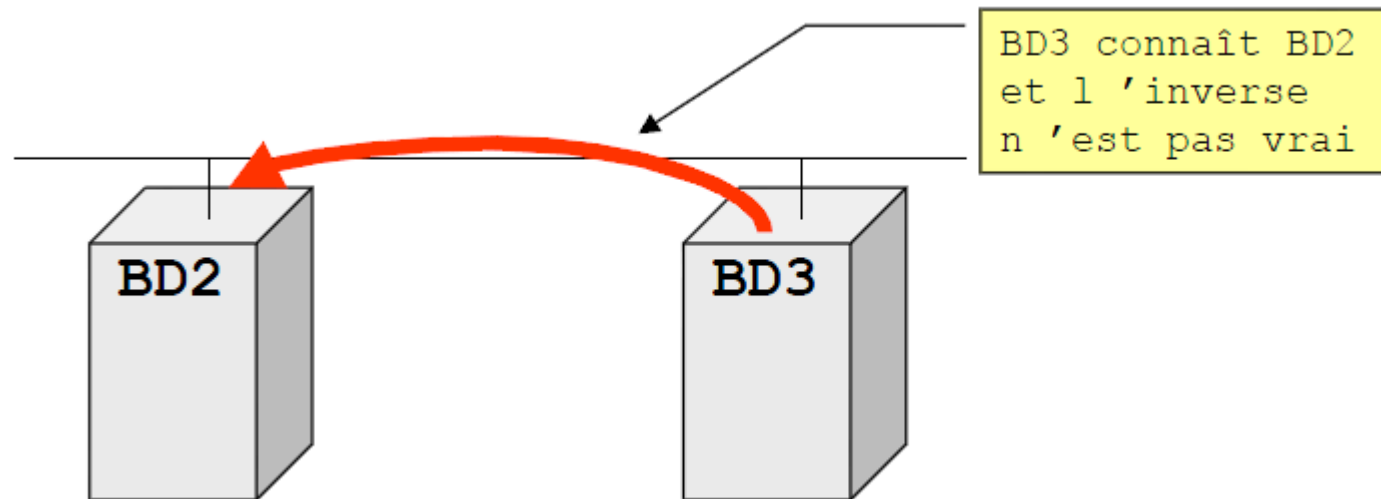
```
CREATE SYNONYM Sequence_Vehicule  
FOR Sequence_Vehicule @lien_Base2;
```

- **Database Link**

- Les bases de données doivent se connaître via leurs **méta-données**.
- Ses **liens logiques** sont appelés **database links**.
- Un **database link** comme : **Un chemin de communication unidirectionnel d'une base de données à une autre.**
- C'est uniquement un chemin, il n'implique aucune redondance.

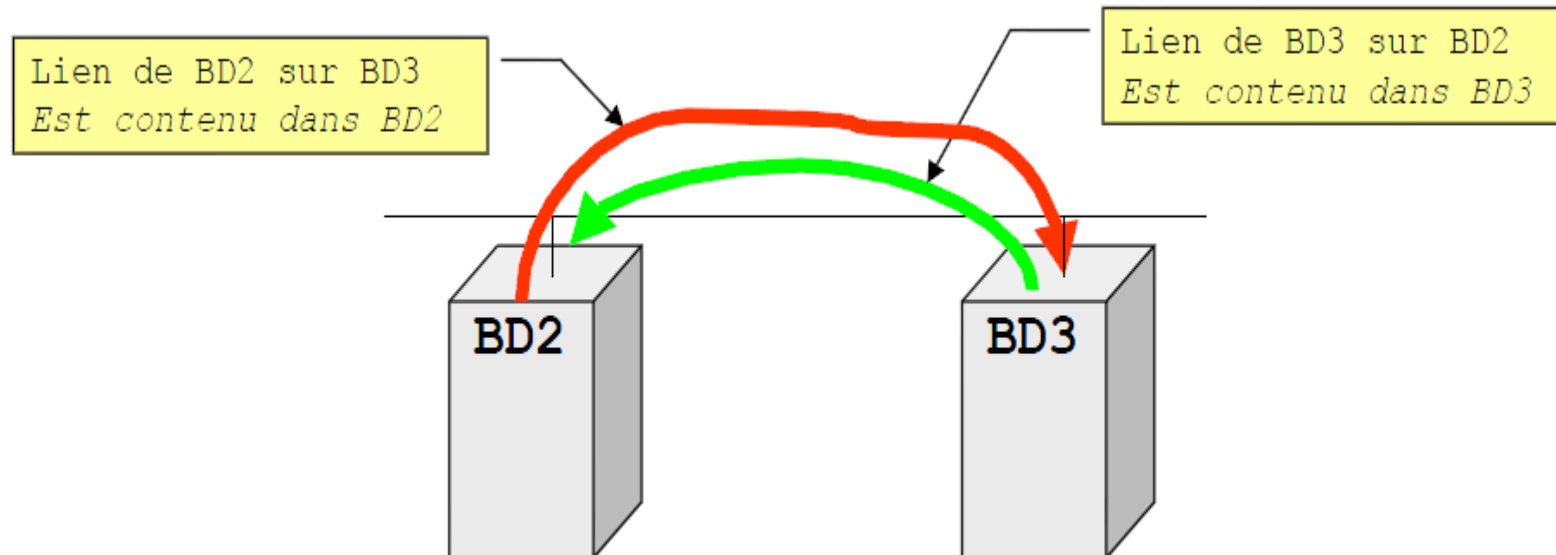
- **Database Link**

- Un database link est unidirectionnel. Il est **contenu dans les méta-donnée de la BD.**
- **Le chemin inverse n'est pas possible.**



- **Database Link** (Connections réciproques)

- Si les deux bases de données doivent se «connaître» mutuellement, **il faut créer deux databaselink**.



- **Database Link**

- Un database link est **un pointeur qui définit un lien unidirectionnel d'une base de données d'oracle à une autre.**
- Il est défini dans les **méta-données (dictionnaire).**
 - Syntaxe Oracle :

```
CREATE DATABASE LINK lien_base2  
CONNECT TO Nom_util_distantIDENTIFIED BY mot_passe_util_distant  
USING 'base2';
```



Nom du lien

User/Password du
compte miroir
distant

Chaîne de
connexion du
serveur distant

- **Database Link**

- Utilisation d'un lien : Sur la base de données 'Base1'

Select * from Vehicule@lien_base2;

-- liste la table distante Vehicule

- Suppression d'un lien

DROP DATABASE LINK lien_base2;

• EXERCICE 1

On dispose de la même structure de la BD sur les deux serveurs Serveur1 (site 192.168.1.101) et Serveur2 (192.168.1.102).

Villes(c_ville, nom_ville , c_region);

Agences(n_agence, nom_ag, adresse, c_ville);

Clients(n_compte, nom, prenom, profession, date_naiss, n_agence);

Retraits(n_cheque, montant, date_retr, n_compte, beneficiaire) ;

Versements(n_transaction, montant, date_vers, n_compte);

Créer la BD sur Serveur1.

Créer les deux liens de base de données **I12** et **I21** entre le premier et le deuxième serveur **dans les deux sens**.

- ✚ Un des principaux objectifs de bases de données réparties est la transparence à la fragmentation.
 - ▣ Ainsi, même fragmentés, les enregistrements doivent apparaître comme sur un seul site.
 - ▣ Pour ceci, on utilise les vues : View
- ➔ Les utilisateurs pourront consulter la base, ou modifier la base (avec certaines restrictions) à travers la vue, c'est-à-dire manipuler la table résultat du SELECT comme si c'était une table réelle.
- ✚ La syntaxe pour créer une vue est la suivante :

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

Exemple

- Création d'une vue constituant une restriction de la table emp aux employés du département 10.

```
CREATE VIEW emp10 AS SELECT * FROM emp WHERE n_dept = 10;
```

➔ INSERT INTO emp10 VALUES (15, 'Mohamed', 25, 25);

Equivalent à : INSERT INTO emp VALUES (15, 'Mohamed', 25, 25);

⊕ Exemple :

```
CREATE VIEW emp10 AS SELECT * FROM emp WHERE n_dept = 10 WITH  
CHECK OPTION ;
```

Les insertion et modification suivantes ne sont pas autorisées:

```
INSERT INTO emp10 VALUES (15, 'Mohamed', 45, 25);
```

```
UPDATE emp10 SET n_dept = 25;
```

- ✦ Pour supprimer une vue
`DROP VIEW nom_vue;`
- ✦ Pour renommer une vue :
`RENAME ancien_nom TO nouveau_nom`
- ✦ Certaines vues peuvent être l'objet de mise à jour par les instructions `INSERT`, `UPDATE`, `DELETE`.
- ⚙ Déclencheurs **INSTEAD OF**.
 - ➔ Les triggers `INSTEAD OF` prennent la main et font les mises à jour sur les fragments distants.

Les Triggers **INSTEAD OF**

- Ces triggers **s'appliquent sur les vues**.
- Les clients connaissent les **objets virtuels** et exécutent les ordres du LMD.
- Les triggers **INSTEAD OF** prennent la main et **font les mises à jour sur les fragments distants**.

Les Triggers INSTEAD OF

- La syntaxe est la suivante:

```
CREATE TRIGGER nom_du_declencheur  
INSTEAD OF {INSERT | UPDATE | DELETE} ON nom_de_la_vue  
FOR EACH ROW  
BEGIN  
...  
END ;
```

- Exemple

```
CREATE TRIGGER tr_emp10  
INSTEAD OF INSERT ON emp10  
FOR EACH ROW  
BEGIN  
INSERT INTO emp VALUES (:new.NO, :new.nom, :new.n_dept, :new.age);  
END ;
```

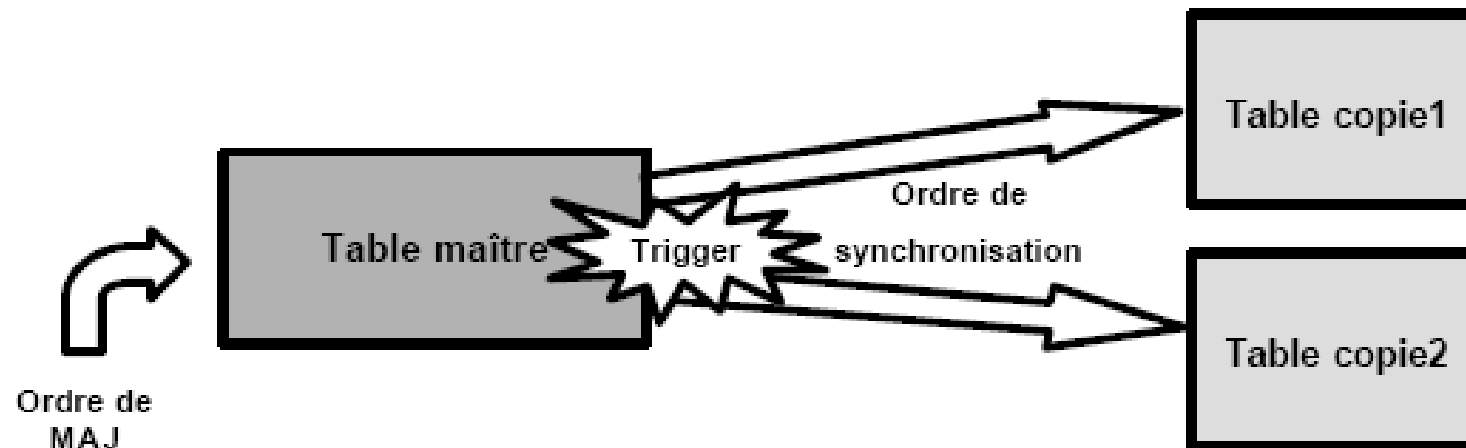
Les Triggers **INSTEAD OF** : Exemple

```
CREATE TRIGGER InsertVehicule  
INSTEAD OF INSERT ON Vehicule  
FOR EACH ROW  
BEGIN  
    IF :NEW.Type='Voiture' THEN  
        INSERT INTO vehicule.voiture@vers_BaseVoiture  
            VALUES(:NEW.NumVeh, :NEW.Marque, :NEW.Modele) ;  
    ELSIF : NEW.Type ='Bicyclette' THEN  
        INSERT INTO vehicule.bicyclette@vers_BaseBicyclette    VALUES(:NEW.NumVeh,  
:NEW.Marque, :NEW.Modele) ;  
    ELSE RAISE_APPLICATION_ERROR (-20455,'Entrer Voiture ou Bicyclette');  
    END IF;  
END;
```


- **Mise à jour instantanée** de la copie pour toute modification de la table maître.



- **Utilisation des trigger** de type ‘**before**’ qui propage la mise à jour sur la table image.



- **Trigger** de type '**before**' qui propage la mise à jour sur la table image :
 - Spécifie comment doivent évoluer les données lors d'une mise à jour.

EXEMPLE :

```
CREATE TRIGGER trigsal
BEFORE UPDATE OF sal
ON job
FOR EACH ROW
BEGIN
    IF (:new.sal < :old.sal)
    THEN raise_application_error(-20001, 'Le salaire ne peut pas
baisser');
    END IF;
END;
```

```
CREATE TABLE JOB (
    no NUMERIC,
    sal NUMERIC
)
```


SYNTHÈSE :

- Dans ce mode de réplication les **mise à jour** sur un site sont prise en compte **immédiatement** dans les autres sites.
- Permet de garder les données dans **le dernier niveau de mise à jour**.
- Quelque soit la copie accédée le système fourni des données dans **la dernière version**.
- Il est nécessaire de **gérer des transactions multi sites** qui sont coûteuses.

- La copie pour toute modification de la table maître.
- **Chaque transaction met à jour une seule copie** et la mise-à-jour des autres copies est différée (dans d'autres transactions).
- Oracle utilise la notion de **SNAPSHOT** ou **vues matérialisées**.

SNAPSHOT :

- Afin de répliquer les données d'une table à l'autre, Oracle utilise le concept de **SNAPSHOT** ou **clichés**.
- Un snapshot est **une copie conforme d'une table** située sur une base de donnée du **système distribué**. Il permet de diminuer les coûts réseau, en rendant local les données situées à distance.
- Afin d'**assurer la cohérence de données**, une **mise à jour régulière** et **automatique** est effectuée à partir du **site d'origine** ou **MASTER**.

SNAPSHOT :

- Il existe **2 types de Snapshot** :
 - en **lecture seule** (read-only)
 - ou **mis à jour** (updateable)

SNAPSHOT en lecture seule:

- Un cliché est constitué par une requête SQL
 - **Image simple** : utilisation d'une seule table, pas d'opérations ensemblistes, pas de GROUP BY.
 - **Image complexe** : autrement.
- Le cliché est rafraîchi à **intervalles réguliers** (refresh) ou **à la demande** (manuellement).
 - **Rafraîchissement rapide** (uniquement pour les images simples) : uniquement les modifications sont propagées et non toute la table.
 - **Rafraîchissement complet** : régénérer complètement l'image.

SNAPSHOT en lecture seule : Principe

- Création d'un cliché avec les méthodes de rafraîchissement et le contenu choisi.
- Pour chaque table maître qui alimente un cliché, il faut créer un **journal d'images** (SNAPSHOT LOG).
- Le **journal** contient les **mise à jour différées**.
- Une table maître (même journal) peut alimenter plusieurs fragments dupliqués.

SNAPSHOT en lecture seule : Création

```
CREATE SNAPSHOT [nom_schéma.]Nom_image  
[Spécification de stockage]  
[REFRESH [FAST|COMPLETE|FORCE]  
[START WITH Date1]  
[NEXT date2]  
AS Requête;
```

- **REFRESH** : mode est fréquence de rafraîchissement
 - **FAST** : rapide
 - **COMPLETE** : complet
 - **FORCE** : laisser le choix à Oracle
- Premier rafraîchissement : date1.
- Puis rafraîchissement toutes les date2.
- Si **REFRESH** est omis =aucun rafraîchissement.

SNAPSHOT en lecture seule : Exemple

- Création d'un cliché avec rafraîchissement tous les 10 jours :

```
CREATE SNAPSHOT Image_client  
REFRESH FAST  
START WITH SYSDATE  
NEXT SYSDATE + 10  
AS Select * from client@dblink;
```

- Création d'un journal image (créé sur la base source) :

```
CREATE SNAPSHOT LOG ON Client;
```


SNAPSHOT en lecture seule : Raffraichissements

- **Modes:**
- **Rapide:** Le mode rapide indiqué par la clause **FAST** permet de faire un rafraîchissement en tenant compte **seulement des mises à jour effectuées** sur le site Maître. Dans ce cas, un **SNAPSHOT LOG** doit être créé pour la table Maître afin de noter les différents changements survenus qui seront répercutés sur le snapshot.
- **Complet:** à chaque rafraîchissement, **toute la table est transférée**. Ce mode est obligatoire pour les snapshots complexes et est indiqué par **COMPLETE**.
- **Forcé:** Dans ce mode, un rafraîchissement rapide est d'abord tenté; s'il ne marche pas le rafraîchissement complet est effectué.

SNAPSHOT en lecture seule : Temps de rafraichissements

- Le rafraîchissement peut se déclencher de plusieurs manières:

- Sur demande : **ON DEMAND**

execute DBMS_MVIEW.REFRESH('MA_VM');

- Sur validation : **ON COMMIT**

De façon périodique : START WITH sysdate NEXT sysdate+n

SNAPSHOT de mise à jour

Les updateable Snapshots:

- Les snapshots de mise à jour peuvent être directement modifiés.
 - ➔ Dans ce cas, les données mises à jour à leur niveau sont répliquées vers le site Master lors du processus de rafraîchissement.

Syntaxe:

```
CREATE SNAPSHOT nom_snapshot  
[REFRESH          FAST | COMPLETE | FORCE]  
START WITH        date_de_debut_de_synchronisation  
NEXT              date_de_la_prochaine_synchronisation  
ENABLE QUERY REWRITE  
AS                requête_select;
```

SNAPSHOT de mise à jour

| Description | Date Expression |
|---|--|
| Now | <code>SYSDATE</code> |
| Tomorrow/ next day | <code>SYSDATE + 1</code> |
| Seven days from now | <code>SYSDATE + 7</code> |
| One hour from now | <code>SYSDATE + 1/24</code> |
| Three hours from now | <code>SYSDATE + 3/24</code> |
| An half hour from now | <code>SYSDATE + 1/48</code> |
| 10 minutes from now | <code>SYSDATE + 10/1440</code> |
| 30 seconds from now | <code>SYSDATE + 30/86400</code> |
| Tomorrow at 12 midnight | <code>TRUNC(SYSDATE + 1)</code> |
| Tomorrow at 8 AM | <code>TRUNC(SYSDATE + 1) + 8/24</code> |
| Next Monday at 12:00 noon | <code>NEXT_DAY(TRUNC(SYSDATE), 'MONDAY') + 12/24</code> |
| First day of the month at 12 midnight | <code>TRUNC(LAST_DAY(SYSDATE) + 1)</code> |
| The next Monday, Wednesday or Friday at 9 a.m | <code>TRUNC(LEAST(NEXT_DAY(sysdate, 'MONDAY'), NEXT_DAY(sysdate, 'WEDNESDAY'), NEXT_DAY(sysdate, 'FRIDAY'))) + (9/24)</code> |

Vues Matérialisées

- Une vue standard est **une table virtuelle**.
- Une vue matérialisée (VM) est **un moyen** simple de **créer une vue physique** d'une table (c'est une sorte de **réplication de la table réelle**).
- Comme son nom l'indique et à la différence d'une vue standard, les données sont dupliquées.
- On l'utilise à des fins d'**optimisation de performance**, ou pour faire des répliquions de table.
- La '**fraicheur**' des données de la VM dépend des options choisies. Le décalage entre les données de la table maître et la VM peut être nul (rafraichissement synchrone) ou d'une durée planifiée.

Vues Matérialisées : Création

```
CREATE MATERIALIZED VIEW nom_vue_m aterialisée  
REFRESH FAST  
START WITH sysdate  
NEXT sysdate+ 1  
AS SELECT * FROM Employes@db_link
```

```
CREATE MATERIALIZED VIEW MV_UneVueMaterialisee  
REFRESH FAST  
START WITH SYSDATE  
NEXT SYSDATE + 1  
AS SELECT * FROM monSchema.MaTable;
```

Exemple:

```
CREATE materialized view emp_snap_fast  
REFRESH COMPLETE START WITH Sysdate NEXT Sysdate + 1  
Enable QUERY REWRITE  
AS SELECT ID, NOM FROM emp@site1tosite2 WHERE n_dept = 10 ;
```

Vues Matérialisées : Création


- Il existe essentiellement **2 tables sur les vues matérialisées** :
 - USER_MVIEWS
 - et USER_MVIEW_LOGS.
- On peut vérifier également **les objets créés implicitement** :
SQL> select object_name, object_type from user_objects;
- il faut **tracer les mises à jour** faites sur la **table maître** dans des fichiers LOG:
create materialized view log on nom_table;
- Les logs prennent le nom de la table maître préfixé par 'MLOG\$_'.

Vues Matérialisées : Privilèges

- Celui qui crée des MV **doit avoir les privilèges suivants** :
 - CREATE ANY VIEW.
 - CREATE TABLE.
 - CREATE ANY INDEX.
 - CREATE ANY MATERIALIZED VIEW ou bien CREATE ANY SNAPSHOT.



TP1 á TP3



Déposer le compte
rendu dans la
plateforme