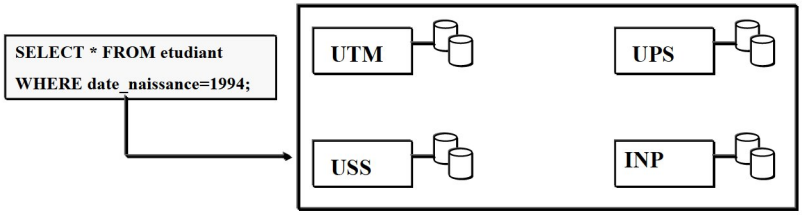
Bases de Données Réparties

Fragmentation et Duplication

BDR : Définition

- Ensemble de bases de données gérées par des sites différents et apparaissant à l'utilisateur comme une base unique
- Les 4 universités Toulousaines :



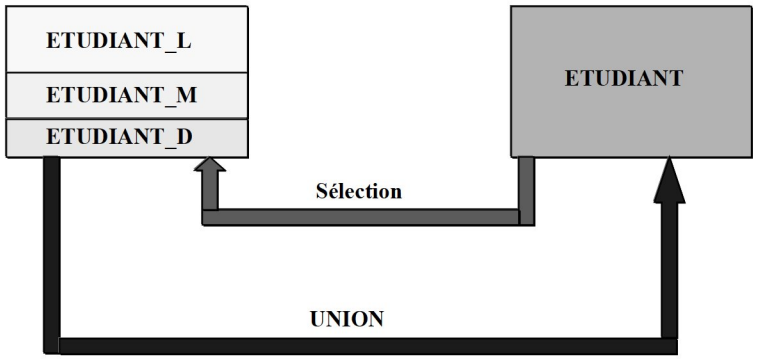
Comparatif Client-Serveur et BD répartie

Plusieurs bases vues par le client	Une base logique vue par le client
Plusieurs connexions	Une seule connexion
Localisation explicite des bases ('connect string' ou DSN)	Indépendance à la localisation
Règles de localisation dans l'application	Règles de localisation dans le dictionnaire
1 ordre SQL → 1 seule BD	1 ordre SQL → plusieurs BD
N transactions mono-base	1 transaction logique
Synchronisation des transactions dans l'application	Synchronisation automatique des n sous-transactions
Plusieurs COMMIT mono-base	1 COMMIT généralisé
CLIENT – SERVEUR	BD REPARTIE

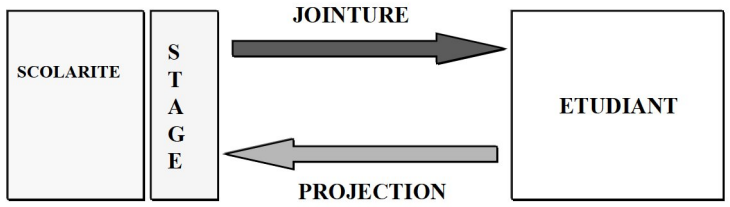
Règles de fragmentation

- Trois types de fragmentation
  - Horizontal, Vertical et Mixte
- Opérateurs SQL utilisés
  - projection et restriction pour fragmenter
  - union et jointure pour reconstituer

Fragmentation horizontale

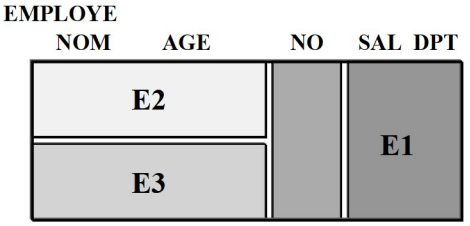


Fragmentation verticale



- Duplication de la clé primaire sur chaque fragment

Fragmentation mixte



E1 (NO, SAL, DPT)
E2 (NOM, AGE, NO) AGE<=30
E3 (NOM , AGE, NO) AGE>30

- Prise en compte des règles de fragmentation pour optimiser les requêtes :
  - SELECT …. FROM e3 WHERE …. AND age <28
  - SELECT e3.* FROM e2,e3 WHERE e2.no=e3.no

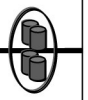
Fragmentation : construction de la BDR

- Construction descendante : la base centralisée est distribuée sur plusieurs sites (BDD)
  - C'est le cas le plus fréquent
  - La base centralisée devient trop importante : elle est distribuée avec des règles
- Construction ascendante : les diverses bases locales sont restructurées et assemblées, la base est répartie sur plusieurs sites (BDR)
  - Cas du rachat ou du regroupement d'entreprises

Faire une bonne fragmentation

- Horizontale
  - Fragments (prédicats) disjoints
  - Fréquence d'accès uniforme (complétude) aux fragments
- Verticale
  - Regroupement des attributs accédés souvent ensemble
  - Calcul de "l'affinité" entre deux attributs

Duplications multiples (indépendance à la localisation)

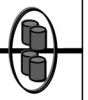


- Duplication des données sur des sites distants
- Gain de temps pour les accès en lecture (interdit en modification) → augmentation des performances
- Mises à jour par rafraîchissements périodiques à partir du site qui possède le fragment initial
- Inconvénient : pas de mise à jour immédiate
- Intéressant pour les données stables

Duplication : le principe des clichés ou "snapshots"

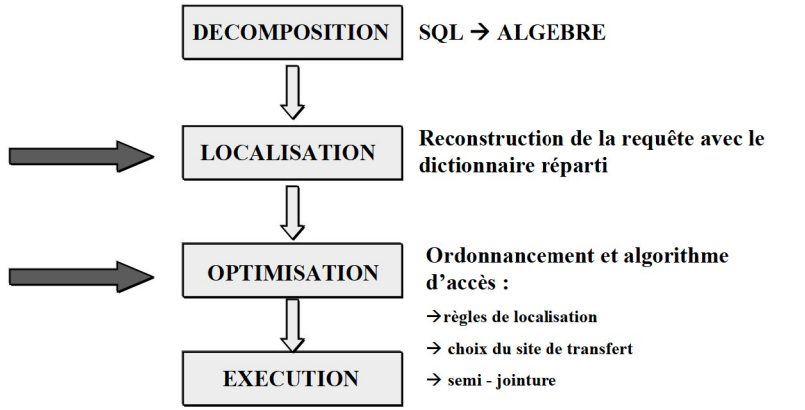
- Duplication en "cohérence faible"
- Rafraîchissements périodiques
- Plusieurs solutions techniques
  - "estampillage" des lignes modifiées avec la date et heure de sa dernière MAJ
  - Utilisation d'une "table différentielle" contenant uniquement les lignes modifiées
- Oracle a choisi la deuxième technique avec un "journal" de transactions sur le fragment initial (voir plus loin)

Requêtes distribuées



- Évaluation des requêtes distribuées ou réparties → plan d'exécution réparti
- Le SGBD comprend que la requête traite avec des objets physiques distants
- La requête est reconstruite en tenant compte de la localisation des objets
- Les opérateurs de restriction (sélection, projection) qui réduisent la taille sont appliqués au plus tôt
- Utilisation du parallélisme

Évaluation d'une requête répartie



Optimisation d'une requête répartie : la semi - jointure

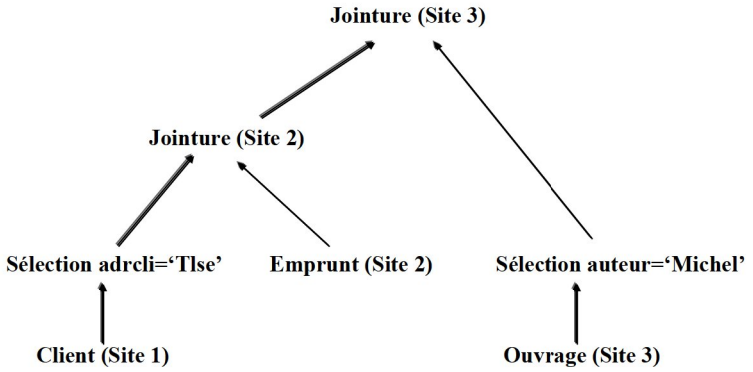
Site 1 : Client (IdCli, nomcli, ….)

Site 2 : Emprunt (IdCli,IdOuv, ….)

Site 3 : Ouvrage (IdOuv, titre, ….)

```
SELECT c.nomcli, o.titre FROM client
c, emprunt e, ouvrage o WHERE
c.idcli=e.idcli AND e.idouv=o.idouv AND
c.adrcli='Tlse' AND o.auteur='Michel';
```

Requête répartie avec jointures



Création des fragments dans les bases distantes

- SQL-ANSI propose un ordre CREATE FRAGMENT permettant de créer les tables distantes
- Cet ordre permettra de conserver les règles de fragmentation dans le dictionnaire réparti
- Cet ordre n'est pas encore implémenté
- Oracle propose l'ordre COPY
- COPY est un ordre SQL+ permettant de Duplicationr un fragment d'une base vers une autre en utilisant les 'connect string'
- COPY possède 2 variables d'environnement configurables
  - COPYCOMMIT : intervalle du nombre de lignes transférées entre 2 COMMIT (0 : COMMIT à la fin)
  - ARRAYSIZE : nombre de lignes transférées par chaque FETCH

Commande COPY

- **Syntaxe de la commande**

```
COPY FROM spécification_base1     -
       TO spécification_base2     -
{APPEND|CREATE|REPLACE|INSERT}    -
fragment [(colonnes)]             -
USING SELECT …….
```

Caractère ligne suite

- APPEND : [CREATE] + INSERT
- CREATE : CREATE + INSERT
- REPLACE : [DROP] + CREATE + INSERT
- INSERT : INSERT

Exemples de COPY

- **Création ou remplacement du fragment**

```
COPY FROM michel/michel@base1       -
       TO michel/michel@base2       -
       REPLACE enseignants_info          -
       USING SELECT * FROM enseignants   -
              WHERE ufr='info'
```

- **Création d'un fragment initial avec restriction verticale**

```
COPY FROM michel/michel@base1       -
       TO michel/michel@base2       -
       CREATE etudiant_scol(ine,nom,adr) -
       USING SELECT inet,nomet,adret     -
       FROM etudiant
```

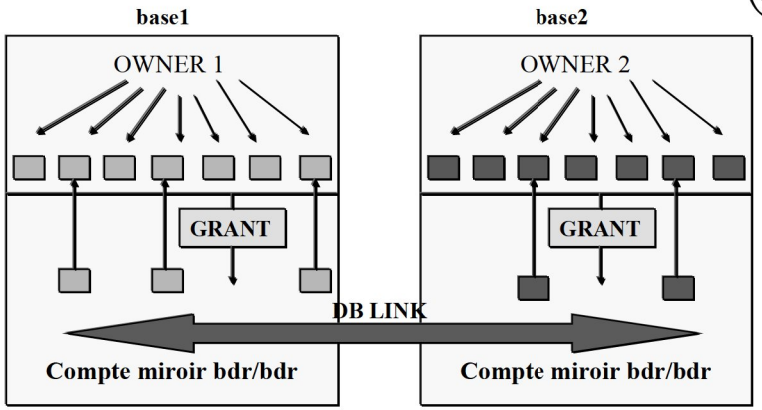
Contraintes des fragments

- La commande COPY n'exporte pas les contraintes (sauf NOT NULL)
- Il faut les recréer
  - Clés primaires
  - Clés étrangères
  - Contraintes autres
- Problème pour les clés étrangères distantes
  - Impossible d'utiliser les DB LINKS (voir plus loin)
  - Créer deux TRIGGERS :
    - Sur le fragment fils : le père doit exister
    - Sur le fragment père : suppression impossible si des fils sont présents

Bases réparties : travail de compte à compte

- Ne pas travailler avec les véritables comptes propriétaires des données
- Chaque site distant doit créer un compte ayant accès aux objets répartis locaux
- Ces comptes 'miroir' sont crées par le DBA et reçoivent les droits d'accès par les propriétaires des données réparties
- Chaque responsable local de la BDR ne connaît que le password des comptes 'miroir' distants

Organisation de l'ensemble



Lien inter-bases : Database Link

- Lien défini par un utilisateur pour relier deux bases
- Connaissance du user/password du compte miroir distant
- Utilisation du 'connect string' du serveur local pour accéder à l'instance distante

```
CREATE DATABASE LINK dbl_base2
CONNECT TO bdr IDENTIFIED by bdr
USING 'base2';
```

Nom du 'connect string'

User/password

Manipulation des DataBase Link

- Suppression d'un lien

```
DROP DATABASE LINK dbl_base2;
```

- Dictionnaire de données : USER_DB_LINKS

```
SQL> col DB_LINK format a8
SQL> col USERNAME format a8
SQL> col  PASSWORD format a8
SQL> col HOST format a8
SQL> col CREATED format a8
SQL> select * from user_db_links;

DB_LINK  USERNAME PASSWORD HOST     CREATED
-------- -------- -------- -------- --------
DB_ORA   MICHEL   MICHEL   oracle   14/01/03
```

Utilisation des DataBase Link

- Sélection d'un fragment distant

```
SQL> select * from etudiant@DB_ORA;

       INE NOM                  DIPLO       CYCLE
---------- -------------------- ----- ----------
       100 étudiant 100         miag3           2
       200 étudiant 200         stri3           2
       300 étudiant 300         miag3           2
       400 étudiant 400         stri2           2
```

- Manipulation distante

```
SQL> update etudiant@DB_ORA SET cycle=3;

 4 ligne(s) mise(s) à jour.
```

Indépendance à la localisation les SYNONYMS

- Création d'un synonyme

```
CREATE SYNONYM etudiant FOR etudiant@db_ora;
```

- Suppression d'un synonyme

```
DROP SYNONYM etudiant;
```

Constructions des objets virtuels

- Reconstitution d'un objet fragmenté : VIEW

```
CREATE VIEW etudiant (ine,nom,adr,cycle) AS
SELECT inet,nomet,adret,'L' FROM etud_licence
UNION
SELECT inet,nomet,adret,'M' FROM etud_mastere;
```

- Reconstitution d'un objet non éclaté : SYNONYM

```
CREATE SYNONYM sequence_client
       FOR sequence_client@db_link;
```

Manipulation d'une base répartie les Procédures stockées

- Les procédures stockées ou packages se comportent comme de véritables méthodes
- Les données réparties sont encapsulées et ne sont pas accessibles directement aux développeurs clients
- Les règles de fragmentation sont dans les procédures
- La localisation des données est transparente aux utilisateurs : appel des procédures sans connaissance de la base
- La transaction est traitée dans la procédure (COMMIT / ROLLBACK)

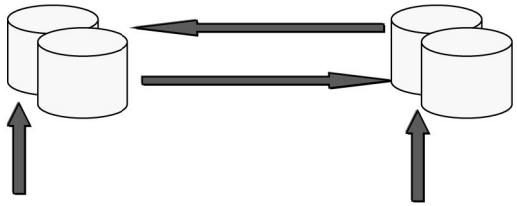
Manipulation d'une base répartie les Triggers INSTEAD OF

- Ces triggers s'appliquent sur des vues
- Les développeurs 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 développeurs 'serveur' connaissent les règles de distribution
- Ces triggers 'lèvent' éventuellement des erreurs applicatives (raise_application_error)

Exemple de Trigger INSTEAD OF

```
CREATE  TRIGGER insert_etudiant
INSTEAD OF INSERT ON etudiant FOR EACH ROW
BEGIN
IF :NEW.cycle='L' THEN
INSERT INTO etudiant_licence@db_l VALUES
(:NEW.ine,:NEW.nom,:NEW.adresse);
INSERT INTO stage@db_s VALUES
(:NEW.ine,:NEW.nomstage,:NEW.adstage);
ELSIF :NEW.cycle='M' THEN
 ..... Idem pour M et D ........
ELSE RAISE_APPLICATION_ERROR
(-20455,'Entrer M, L ou D');
END IF;
END;
/
```

Duplication et Réplication des données réparties



Distribution , Duplication et Réplication

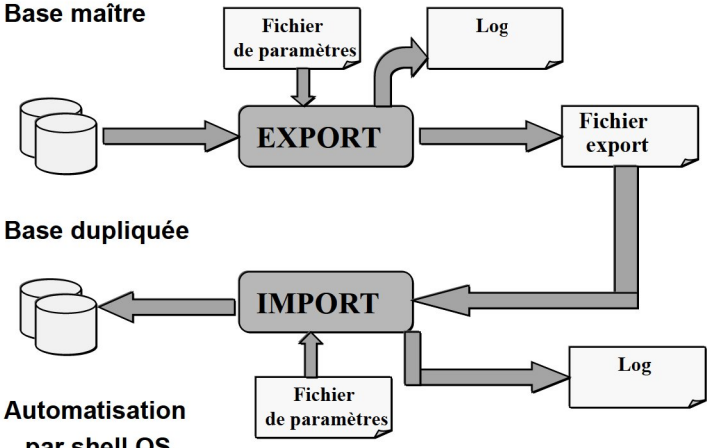
- Distribution
  - BD distribuée ou répartie
  - Sans redondance
- Duplication
  - Duplication locale d'un fragment éloigné maître
  - Fragment local en lecture seule
  - Notion de cliché ou snapshot (materialized view)
  - Duplication synchrone (maj instantannée) ou asynchrone (maj en différé)
- Réplication
  - Pas de fragment maître
  - Duplications en miroir
  - Réplication synchrone (emploi de jetons) ou asynchrone (problèmes de cohérence)

Duplication de données : différentes possibilités

- Duplication d'une base entière
  - EXPORT – IMPORT programmé
- Duplication d'une table
  - Create ou Copy
- Duplication synchrone : trigger ou trigger instead of
- Duplication asynchrone programmée par programmateur
- Duplication asynchrone assurée par Oracle : les snapshot ou vues matérialisées

Exportation et Importation d'une base

- Base maître
- Base dupliquée
- Automatisation par shell OS



Duplication d'une table distante

- Processus simple, rapide et fiable (PUSH ou PULL)
- Duplication complète (sans les contraintes)

```
CREATE TABLE copie AS SELECT * FROM
          maître@dblink;
```

- Duplication d'un fragment

```
CREATE TABLE Duplication AS
SELECT col1, col3, col5  FROM
          maître@dblink
     WHERE prédicat_de_resriction;
```

Duplication d'une table distante (2)

- Table locale existante

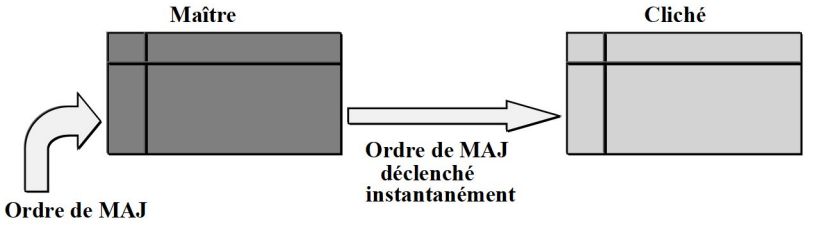
```
DELETE FROM copie;
INSERT INTO copie
      SELECT * FROM maître@dblink;
COMMIT;
```

- Marquage des lignes transférées

```
INSERT INTO copie
      SELECT * FROM maître@dblink
      WHERE jeton='pas transféré'
            FOR UPDATE;
UPDATE maître SET jeton='transféré'
      WHERE jeton='pas transféré';
COMMIT;
```

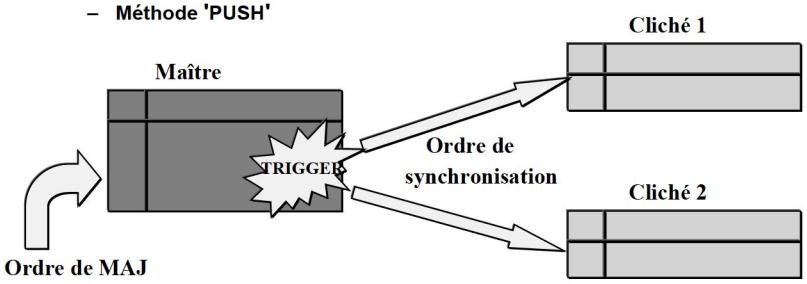
Duplication Synchrone

- Mise à jour instantanée de la Duplication pour toute modification de la table maître
- La duplication synchrone fait partie de la transaction



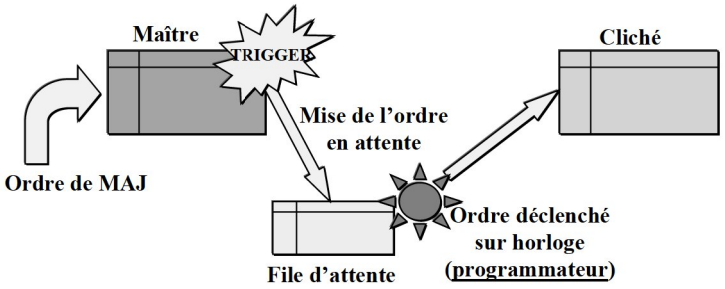
Duplication Synchrone : mise en œuvre avec les trigger

- Trigger de type 'before' qui répercute l'ordre exécuté sur la table maître dans la table cliché
  - La transaction est gérée par le client : exceptions possibles
  - Insertion, mise à jour et suppression de données
  - Plusieurs clichés possibles
  - Méthode 'PUSH'



Duplication Asynchrone par programmateur

- La mise à jour du cliché est différée
- Utilisation d'un programmateur et d'une file d'attente



Exemple de mise en attente d'un ordre

- Un trigger de type 'before' est posé sur la table maître
- Important de conserver la séquence des ordres de mise à jour
- Utilisation du dblink

```
CREATE TRIGGER mise_attente BEFORE INSERT
ON maître FOR EACH ROW
vordre VARCHAR(200);
BEGIN
     vordre:='INSERT INTO cliché@dblink
            VALUES ……..';
INSERT INTO attente(numéro,ordre)
     VALUES (seq_ordre.NEXTVAL,vordre);
END;
```


Exemple de mise en attente d'ordres

- Ordres de mise à jour dans maître

```
INSERT INTO maitre VALUES('ligne1');
INSERT INTO maitre VALUES('ligne2');
INSERT INTO maitre VALUES('ligne3');
DELETE FROM maitre WHERE nom='ligne2';
UPDATE maitre SET nom='ligne33'
 WHERE nom='ligne3';
```

- Table Maître

```
SQL> SELECT * FROM maitre;
NOM
----------
ligne1
ligne33
```

Exemple de mise en attente d'ordres (2)

- Table attente

```
SQL> select * from attente;

NUMERO ORDRE
------ ---------------------------------------------
     1 insert into cliché@db_ora values('ligne1')
     2 insert into cliché@db_ora values('ligne2')
     3 insert into cliché@db_ora values('ligne3')
     4 delete from cliché@db_ora where nom = 'ligne2'
     5 update cliché@db_ora set nom = 'ligne33'
         where nom = 'ligne3'
```


Notion de Programmateur

- Service de déclenchement d'un processus par ordre d'une horloge
- Logiciel installé extérieur (OS) ou interne à la base de données
- La table d'attente peut être sur le site maître ou de Duplication
- Le programmateur peut être sur le site maître ou de Duplication
- Programmateur interne à la BD : JOBS

Programmateur avec Oracle

- Processus (Unix) ou Service de type SNP
  - Vérifier l'état du service sur NT (activé)
- Le processus ouvre une session dans la base à intervalle régulier (programmé) et consulte les tâches
- Exécution en tâche de fond
- Utilisation multiple : sauvegarde, export, duplication, ….
- Exécution asynchrone

Mise en service du programmateur (DBA)

- Vérification du nombre de jobs autorisés

```
SQL> SELECT name,value FROM v$parameter
  2  WHERE name LIKE '%job%';
NAME                           VALUE
------------------------------ ---------
job_queue_processes            10
```

- Si = 0, modifier le paramètre (init.ora, pfile) ou la commande système :

```
ALTER SYSTEM SET job_queue_processes=10;
-- 1000 au maximum
```

- Droits d'exécution du package

```
CONNECT system
GRANT EXECUTE ON dbms_job TO user;
```

Package DBMS_JOB

- Ce package permet de manipuler des taches ou JOBS
- Il contient des procédures :
  - REMOVE
  - CHANGE
  - WHAT
  - RUN
  - SUBMIT

Procédure SUBMIT

- Spécification

```
PROCEDURE submit
      (  job       OUT BINARY_INTEGER,
         what       IN  VARCHAR2,
         next_date  IN  DATE DEFAULT sysdate,
         interval   IN  VARCHAR2 DEFAULT 'null',
         no_parse   IN  BOOLEAN DEFAULT FALSE,
         instance   IN  BINARY_INTEGER DEFAULT 0,
         force      IN  BOOLEAN DEFAULT FALSE );

-- Submit a new job.Chooses JOB from the
      -- sequence sys.jobseq.
-- For example,
--   variable x number;
--   execute dbms_job.submit(:x,'pack.proc(''arg1'');'
            -- ,sysdate,'sysdate+1');
```


Autres Procédures

- REMOVE

```
PROCEDURE remove( job  IN  BINARY_INTEGER );
-- Remove an existing job from the job queue.
-- This currently does not stop a running job.
--   execute dbms_job.remove(14144);
```

- WHAT

```
PROCEDURE what( job IN  BINARY_INTEGER,
                what      IN  VARCHAR2 );
  -- Change what an existing job does, and
       --replace its environment
```

- NEXT_DATE

```
PROCEDURE next_date ( job IN  BINARY_INTEGER,
                      next_date IN  DATE );
-- Change when an existing job will next execute
```


Exemple de manipulation de job

- Procédure d'insertion d'une ligne avec horaire

```
drop table testjob;
create table testjob (t varchar(50));
create or replace procedure test_job is
heure varchar(20);
begin
  heure:=to_char(sysdate,'HH24-MI-SS');
; insert into testjob values('ajout : '||heure)
end;
/
execute test_job;

SQL> select * from testjob;
T
-------------------------
ajout : 14-23-51
```

Exemple de manipulation de job (2)

```
variable numjob number;
execute dbms_job.submit(:numjob,'test_job;',
                        sysdate,'sysdate+1/1440');
SQL>print numjob
    NUMJOB
----------
         3
SQL> select * from testjob
T
------------------
ajout : 14-44-28
ajout : 14-45-30
ajout : 14-46-31

SQL>select job,what from user_jobs;
       JOB WHAT
---------- ------------
         3 test_job;
execute dbms_job.remove(3);
```

Attention au ;

Exemple de job d'extraction de la file d'attente

- Procédure d'extraction

```
CREATE PROCEDURE exeordres IS
CURSOR c1 is SELECT ordre FROM attente
            ORDER BY numero FOR UPDATE;
BEGIN
      FOR c1rec IN c1 LOOP
      EXECUTE IMMEDIATE c1rec.ordre;
      DELETE FROM attente WHERE CURRENT OF c1;
      END LOOP;
      COMMIT;
END;
```

- Le job qui lance le programmateur

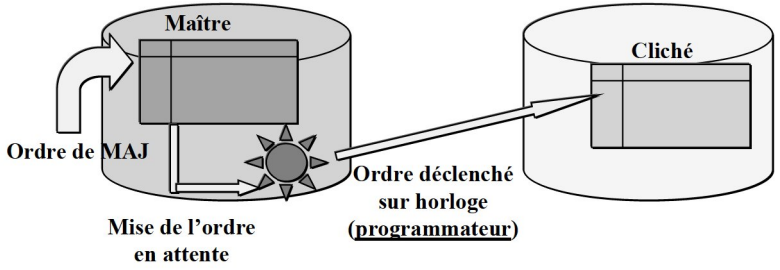
```
variable numjob number;
execute dbms_job.submit(:numjob,'exeordres;',
             sysdate,'sysdate+1');
print numjob
```


Placement du programmateur

- Gestion lourde pour la base (et l'OS)
  - Gestion des processus de fond
- Placé dans la base la moins chargée
- Deux types de propagations possibles
  - PUSH
    - Le programmateur *"pousse"* les ordres de MAJ
  - PULL
    - Le programmateur *"tire"* les ordres de MAJ

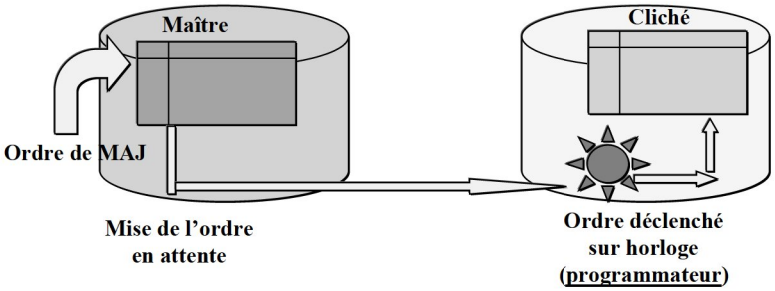
Propagation par PUSH

- La base maître pousse vers le répliqua



Propagation par PULL

- Le répliqua tire les modifications



Duplication par Oracle
Les vues matérialisées ou snapshots

- Technique d'Oracle pour la duplication asynchrone
- Un cliché ou snapshot est un fragment de données en lecture seule
- Le cliché est rafraîchi à intervalles réguliers (refresh) ou à la demande
- Les rafraichissements sont complets (complete) ou différentiels (fast)
- Le cliché (fragment dupliqué) peut être le résultat de :
  - Restriction verticale d'une table
  - Restriction horizontale
  - Une jointure de plusieurs tables

Principe général

- C'est le site du cliché qui tire les MAJ (pull)
- On crée une vue matérialisée pour créer le cliché avec les méthodes de rafraîchissement et le contenu choisis
- Pour chaque table maître qui alimente un cliché, il faut créer un journal de vue matérialisée
- Ce journal contient les mises à jour différées gérées selon deux techniques possibles
  - Par rowid (pas conseillé en cas de réorganisation de blocs)
  - Par clé primaire (utilisé par défaut) → table maître avec une clé primaire obligatoire
- Une table maître (un même journal) peut alimenter plusieurs fragments dupliqués

Mise en œuvre d'un rafraîchissement complet exécuté volontairement

- Création de la table maître sur le site maître

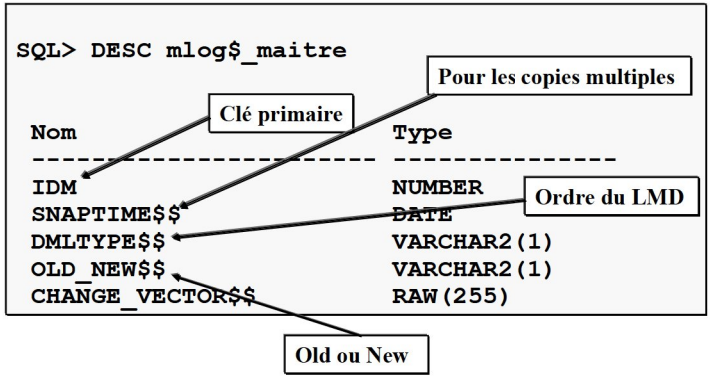
```
DROP TABLE maitre;
CREATE TABLE maitre (idm NUMBER PRIMARY KEY,
                     texte varchar(20));
```

- Création du journal de vue matérialisée sur le site maître

```
DROP MATERIALIZED VIEW LOG ON maitre;
CREATE MATERIALIZED VIEW LOG ON maitre;
```

Journal de vue matérialisée

- De nom MLOG$_nom_de_table_maître



Ordre de création de vue matérialisée (1)

- Création sans intervalles de rafraîchissements

```
CREATE MATERIALIZED VIEW copie
REFRESH [NEVER | COMPLETE | FAST | FORCE]
AS SELECT .....FROM maître@dblink .....;
```

- 4 méthodes de rafraîchissement
  - NEVER : jamais rafraîchie
  - COMPLETE : transfert complet
  - FAST : transferts différentiels
  - FORCE : FAST si possible, COMPLETE sinon

Ordre de création de vue matérialisée (2)

- Création avec intervalles de rafraîchissements

```
CREATE MATERIALIZED VIEW copie
REFRESH FAST
START WITH sysdate NEXT sysdate + 1
WITH PRIMARY KEY
AS SELECT .....FROM maître@dblink .....;
```

Début du transfert

Intervalle de rafraîchissement

Mise en œuvre : cliché par rafraîchissement manuel

- Création du cliché

```
CREATE MATERIALIZED VIEW cliché
REFRESH FAST
AS SELECT * FROM maitre@db_tuf;
```

- On utilisera la procédure refresh du package dbms_mview pour rafraîchir

```
DBMS_MVIEW.REFRESH('nom_mv','F',NULL);
```

F (fast) ou C (complète)

Mises à jour sur la table maître

- Ordres de mise à jour

```
INSERT INTO maitre VALUES(1,'ligne 1');
INSERT INTO maitre VALUES(2,'ligne 2');
INSERT INTO maitre VALUES(3,'ligne 3');
UPDATE maitre SET texte = 'LIGNE 1' WHERE idm=1;
UPDATE maitre SET texte = 'LIGNE 3' WHERE idm=3;
DELETE FROM maitre WHERE idm=2;

SQL> SELECT * FROM maitre;

       IDM TEXTE
---------- --------------------
         1 LIGNE 1
         3 LIGNE 3
```


Contenu du journal de MV

- La date indique qu'aucun rafraîchissement n'a eu lieu à partir de ce journal (pour un éventuel autre cliché)

```
SQL> COL CHANGE_VECTOR$$ FORMAT a10
SQL> SELECT * FROM mlog$_maitre;

       IDM SNAPTIME D O CHANGE_VEC
---------- -------- - - ----------
         1 01/01/00 I N FE
         2 01/01/00 I N FE
         3 01/01/00 I N FE
         1 01/01/00 U U 04
         3 01/01/00 U U 04
         2 01/01/00 D O 00
```


Rafraîchissement manuel

- Rafraîchissement manuel complet

```
EXECUTE dbms_mview.refresh('cliché','C',null);

SQL> select * from cliché;

       IDM TEXTE
---------- --------------------
         1 LIGNE 1
         3 LIGNE 3

-- sur le site maître

SQL> SELECT * FROM mlog$_maitre;

aucune ligne sélectionnée
```

Il n'y a pas d'autres clichés

Rafraîchissement différentiel manuel avec deux clichés (1)

- Sur le site maître

```
INSERT INTO maitre VALUES(4,'ligne 4');
INSERT INTO maitre VALUES(5,'ligne 5');
COMMIT;
```

Sinon, pas de propagation

```
SQL>  SELECT * FROM mlog$_maitre;

       IDM SNAPTIME D O CHANGE_VEC
---------- -------- - - ----------
         5 01/01/00 I N FE
         4 01/01/00 I N FE
```

Rafraîchissement différentiel manuel avec deux clichés (2)

- Sur le site de copie : on crée le second cliché

```
create materialized view cliché2
refresh fast
as SELECT * FROM maitre@db_tuf;
```

- On lance le rafraîchissement manuel

```
EXECUTE dbms_mview.refresh('cliché','F',null);

EXECUTE dbms_mview.refresh('cliché2','F',null);
```


Rafraîchissement différentiel manuel avec deux clichés (3)

- Vérification de la propagation des modifications

```
SQL> SELECT * FROM cliché;
       IDM TEXTE
---------- --------------------
         1 LIGNE 1
         3 LIGNE 3
         4 ligne 4
         5 ligne 5

SQL> SELECT * FROM cliché2;
       IDM TEXTE
---------- --------------------
         1 LIGNE 1
         3 LIGNE 3
         4 ligne 4
         5 ligne 5
```


Rafraîchissement différentiel manuel avec deux clichés (4)

- Sur le site maître

```
SQL> SELECT * FROM mlog$_maitre;

aucune ligne sélectionnée

INSERT INTO maitre VALUES(6,'ligne 6');
COMMIT;

SQL> SELECT * FROM mlog$_maitre;

       IDM SNAPTIME D O CHANGE_VEC
---------- -------- - - ----------
         6 01/01/00 I N FE
```


Rafraîchissement différentiel manuel avec deux clichés (5)

- On ne rafraîchit que le premier cliché

```
EXECUTE dbms_mview.refresh('cliché','F',null);
SQL> SELECT * FROM cliché;
       IDM TEXTE
---------- --------------------
         1 LIGNE 1
         3 LIGNE 3
         4 ligne 4
         5 ligne 5
         6 ligne 6
```

- Journal du site maître

```
SQL> SELECT * FROM mlog$_maitre;
       IDM SNAPTIME D O CHANGE_VEC
---------- -------- - - ----------
         6 03/10/04 I N FE
```

Il reste la ligne 6 pour cliché2

Voir la date - heure

Rafraîchissement différentiel manuel avec deux clichés (6)

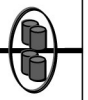
- Mise à jour de maître

```
INSERT INTO maitre VALUES (7,'ligne 7');

SQL> select * from mlog$_maitre;

       IDM SNAPTIME D O CHANGE_VEC
---------- -------- - - ----------
         7 01/01/00 I N FE
         6 03/10/04 I N FE
```


Rafraîchissement différentiel manuel avec deux clichés (7)



- Rafraîchissement de cliché2

```
EXECUTE dbms_mview.refresh('cliché2','F',null);

SQL> select * from esclave2;

       IDM TEXTE
---------- --------------------
         1 LIGNE 1
         3 LIGNE 3
         4 ligne 4
         5 ligne 5
         6 ligne 6
         7 ligne 7
```


Rafraîchissement différentiel manuel avec deux clichés (8)

- Le journal de maître

```
SQL> SELECT * FROM mlog$_maitre;
       IDM SNAPTIME D O CHANGE_VEC
---------- -------- - - ----------
         7 03/10/04 I N FE
```

- Rafraîchissement de cliché

```
EXECUTE dbms_mview.refresh('cliché','F',null);

-- sur le site maître

SQL> SELECT * FROM mlog$_maitre;
aucune ligne sélectionnée
```