Ecole Supérieure de Technologie et d'Informatique à Carthage

Année Universitaire 2005-2006

Systèmes de Gestion de Bases de Données Réparties & Mécanismes de Répartition avec Oracle

Rim Moussa

M.A. à l'ISSATM –Université 7 Nov. à Carthage

Email: Rim.Moussa@issatm.rnu.tn

URL: http://ceria.dauphine.fr/Rim/SupportBDR.pdf

Partie I: Les Bases de Données Réparties

Table des Matières

Partie I : Les Bases de Données Réparties

1. Besoins, Objectifs & Définitions 6
   1.1. Problématique 6
   1.2. Buts de la répartition des bases de données 6
   1.3. SGBD réparti 7
   1.4. Objectifs définis par C.J. Date 7
   1.5. Problèmes à surmonter 8

2. Conception d'une base de données répartie 8
   2.1. Conception descendante (*top down design*) 8
   2.2. Conception ascendante (*bottom up design*) 9

3. Fragmentation 10
   3.1. Techniques de Fragmentation 10
   3.2. Définition des fragments 12

4. Schéma d'allocation 15

5. Réplication 16

6. Traitement & Optimisation de Requêtes Réparties 17
   6.1. Mise à jour de BD réparties 17
   6.2. Requêtes sur BDs réparties 18

7. Gestion des Transactions Réparties 24
   7.1. Définitions 24
   7.2. Exemple de Transactions 24
   7.3. Interférences à éviter 25
   7.4. Contrôle de concurrence 27

8. Les Architectures de Systèmes Parallèles 30
8.1. Architecture à mémoire partagée (ang. *Shared-Memory*) 30
8.2. Architecture à disque partagé (ang. *Shared-Disk ou cluster*) 30
8.3. Architecture à mémoire distribuée (ang. *Shared-Nothing*) 31
8.4. Architectures hybrides 32

Partie II : Mécanismes de Répartition avec Oracle

9. Introduction à Oracle: objets & architecture 34
9.1. Structures logiques de la BD 35
9.2. Structures BD physiques 36
9.3. Structures de mémoire 36
9.4. Processus d'arrière-plan 37
9.5. Etapes de traitement d'un ordre SQL 39

10. Oracle en réseau 39

11. Les liens de base de données 40

12. Transparence d'emplacement 41
12.1. Vues 41
12.2. Synonymes 42
12.3. Procédures 42

13. Mise au point des requêtes distribuées 43
13.1. Collocated Inline Views 43
13.2. Optimisation basée sur le calcul des coûts 43
13.3. Statistiques 44
13.4. Hints 44
13.5. Analyse du plan d'exécution 44

14. Réplication des données 45
14.1. Commande COPY 45
14.2. Snapshots 46
14.3. Vues matérialisées 47

15. Administration de grandes bases de données 47
15.1. Partitions 48
15.2. Gestion de Clusters 50

16. Oracle Parallel Query 51

1. Besoins, Objectifs & Définitions

1.1. Problématique

Les pressions pour la distribution :

- Il devient impératif de décentraliser l'information (cas des multinationales),
- Augmentation du volume de l'information (14 fois de 1990 à 2000),
- Augmentation du volume des transactions (10 fois dans les 5 prochaines années).

→ Besoin de serveurs de BDs qui fournissent un bon temps de réponse sur des gros volumes de données.

Prédiction d'accroissement:

Vitesse des microprocesseurs : 50% par an,

Capacité des DRAM : 4 fois tous les 4 ans,

Débit des disques : 2 fois sur les 10 dernières années.

→ Goulot d'étranglement sur les E/Ss.

Pour améliorer le débit des E/Ss :

- Partitionnement des données,
- Accès parallèle aux données,
- Utiliser plusieurs nœuds (avec un bon coût/ performance), et les faire communiquer par un réseau.

Les BDRs se sont développées, grâce au progrès technologiques réalisés au niveau de l'infrastructure réseau et des postes de travail.

1.2. Buts de la répartition des bases de données

Les bases de données réparties ont une architecture plus adaptée à l'organisation des entreprises décentralisées.

- Plus de fiabilité : les bases de données réparties ont souvent des données répliquées. La panne d'un site n'est pas très importante pour l'utilisateur, qui s'adressera à autre site.

- Meilleures performances : réduire le trafic sur le réseau est une possibilité d'accroître les performances. Le but de la répartition des données est de les rapprocher de l'endroit où elles sont accédées. Répartir une base de données sur plusieurs sites permet de répartir la charge sur les processeurs et sur les entrées/ sorties.

- Faciliter l'accroissement: l'accroissement se fait par l'ajout de machines sur le réseau.

1.3. SGBD réparti

Une base de données centralisée est gérée par un seul SGBD, est stockée dans sa totalité à un emplacement physique unique et ses divers traitements sont confiés à une seule et même unité de traitement. Par opposition, une base de données distribuée[1] est gérée par plusieurs processeurs, sites ou SGBD.
Un système de bases de données réparties ne doit donc en aucun cas être confondu avec un système dans lequel les bases de données sont accessibles à distance. Il ne doit non plus être confondu avec une multibase ou une BD fédérée.
Dans une multibase, plusieurs BDs interopérent avec une application via un langage commun et sans modèle commun.
Dans une BD fédérée, plusieurs BDs hétérogènes sont accédées comme une seule via une vue commune.

Du point de vue organisationnel nous distinguons deux architectures :

1. *Architecture Client-Serveur* : les serveurs, ont pour rôle de servir les clients. Par servir, on désigne la réalisation d'une tâche demandée par le client.

2. *Architecture Pair-à-Pair* (Peer-to-Peer, P2P) : par ce terme on désigne un type de communication pour lequel toutes les machines ont une importance équivalente.

1.4. Objectifs définis par C.J. Date

Les principaux objectifs sont:

1. Transparence pour l'utilisateur

2. Autonomie de chaque site

---

[1] Ce terme générique englobe les BD réparties, les BD fédérées et les BD parallèles

3. Absence de site privilégié
4. Continuité de service
5. Transparence vis à vis de la localisation des données
6. Transparence vis à vis de la fragmentation
7. Transparence vis à vis de la réplication
8. Traitement des requêtes distribuées
9. Indépendance vis à vis du matériel
10. Indépendance vis à vis du système d'exploitation
11. Indépendance vis à vis du réseau
12. indépendance vis à vis du SGBD

1.5. Problèmes à surmonter

1. Coût : la distribution entraîne des coûts supplémentaires en terme de communication, et en gestion des communications (–hardware et software à installer pour gérer les communications et la distribution).
2. Problème de concurrence.
3. Sécurité : la sécurité est un problème plus complexe dans le cas des bases de données réparties que dans le cas des bases de données centralisées.

2. Conception d'une base de données répartie

La définition du schéma de répartition est la partie la plus délicate de la phase de conception d'une BDR car il n'existe pas de méthode miracle pour trouver la solution optimale. L'administrateur doit donc prendre des décisions en fonction de critères techniques et organisationnels avec pour objectif de minimiser le nombre de transferts entre sites, les temps de transfert, le volume de données transférées, les temps moyens de traitement des requêtes, le nombre de copies de fragments, etc...

2.1. Conception descendante (*top down design*)

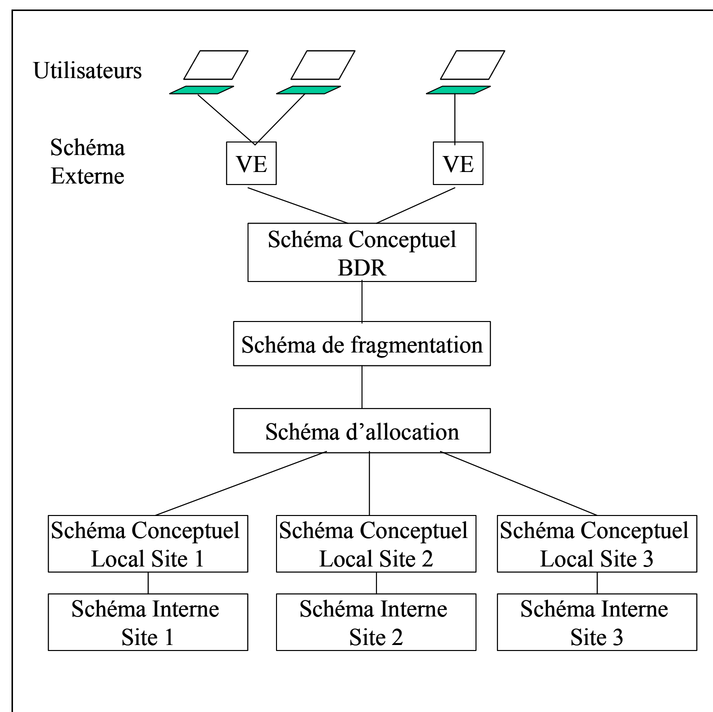
On commence par définir un schéma conceptuel global de la base de données répartie, puis on distribue sur les différents sites en des schémas conceptuels locaux.

La répartition se fait donc en deux étapes, en première étape la fragmentation, et en deuxième étape l'allocation de ces fragments aux sites.

L'approche *top down* est intéressante quand on part du néant. Si les BDs existent déjà la méthode *bottom up* est utilisée.

2.2. Conception ascendante (*bottom up design*)

L'approche se base sur le fait que la répartition est déjà faite, mais il faut réussir à intégrer les différentes BDs existantes en une seule BD globale. En d'autres termes, les schémas conceptuels locaux existent et il faut réussir à les unifier dans un schéma conceptuel global.



Architecture d'une BDR.

La répartition d'une base de donnée intervient dans les trois niveaux de son architecture en plus de la répartition physique des données :

Niveau externe: les vues sont distribuées sur les *sites utilisateurs*.

Niveau conceptuel: le schéma conceptuel des données est associé, par l'intermédiaire du schéma de répartition (lui même décomposé en un schéma de fragmentation et un schéma d'allocation), aux schémas locaux qui sont réparties sur plusieurs sites, les *sites physiques*.

Niveau interne: le schéma interne global n'a pas d'existence réelle mais fait place à des schémas internes locaux répartis sur différents sites.

3. Fragmentation

La fragmentation est le processus de décomposition d'une base de donnée en un ensemble de sous-bases de données. Cette décomposition doit être sans perte d'information.

La fragmentation peut être coûteuse s'il existe des applications qui possèdent des besoins opposés.

Les **règles de fragmentation** sont les suivantes :

1. La **complétude** : pour toute donnée d'une relation *R*, il existe un fragment R_i de la relation *R* qui possède cette donnée.
2. La **reconstruction** : pour toute relation décomposée en un ensemble de fragments *Ri*, il existe une opération de reconstruction.

3.1. Techniques de Fragmentation

Il existe plusieurs techniques de fragmentation, définies par l'unité de fragmentation.

3.1.1. Répartition des classes d'objet

Cette technique consiste en la répartition de classes (relation en relationnel, classe en Orienté-objet) qui peuvent être réparties sur différents fragments. Toutes les occurrences d'une même classe appartiennent ainsi au même fragment.

- L'opération de partitionnement est la définition de sous-schémas.
- L'opération de recomposition est la réunion de sous-schémas.

Dans l'exemple suivant la base de données relationnelle peut être fragmentée en {Compte, Client} et {Agence}

Relation Compte

No client	Agence	Type compte	Somme
174723	Lausanne	courant	123345
177498	Genève	courant	34564
201639	Lausanne	courant	45102
201639	Lausanne	dépôt	325100
203446	Genève	courant	274882

Relation Agence

Agence	Adresse
Lausanne	Rue du lac, 3. 1002 Lausanne
Genève	Avenue du Mont Blanc, 21. 1200 Genève

Relation Client

No client	Nom client	Prènom	Age
174723	Villard	Jean	29
177498	Cattell	Blaise	38
201639	Tsellis	Alan	51
203446	Kowalsky	Valdimir	36

3.1.2. Répartition des occurrences (fragmentation horizontale)

Les occurrences d'une même classe peuvent être réparties dans des fragments différents.

- L'opérateur de partitionnement est la *sélection* (σ)
- L'opérateur de recomposition est l'*union* (∪)

Dans l'exemple précédent, la relation *Compte* peut être fractionnée en *Compte1* et *Compte2* avec la fragmentation suivante

$$\text{Compte1} = \sigma_{[\text{TypeCompte = 'courant'}]} \text{Compte}$$

et $$\text{Compte2} = \sigma_{[\text{TypeCompte = 'dépôt'}]} \text{Compte}$$

La recomposition de *Compte* est *Compte1* ∪ *Compte2*

3.1.3. Répartition des attributs (fragmentation verticale)

Toutes les valeurs des occurrences pour un même attribut se trouvent dans le même fragment. Une fragmentation verticale est utile pour distribuer les parties des données sur le site où chacune de ces parties est utilisée.

- L'opérateur de partitionnement est la projection (π)
- L'opérateur de recomposition est la jointure

Soit le partitionnement de la relation précédente *Client* en deux relations :

$Cli1 = \pi_{[NoClient,\ NomClient]}\ Client$

et $Cli2 = \pi_{[Noclient,\ Prénom,\ Age]}\ Client$

NoClient	NomClient
174 723	Villard
177 498	Cattell
201 639	Tsellis
203 446	Kowalsky

Relation Cli1

NoClient	Prénom	Age
174 723	Jean	29
177 498	Blaise	38
201 639	Alan	51
203 446	Vladimir	36

Relation Cli2

La relation d'origine est obtenue avec la recomposition suivante : Client = Cli1 * Cli2

3.1.4. Répartition des valeurs (fragmentation hybride)

C'est la combinaison des deux fragmentations précédentes, horizontale et verticale. Les occurrences et les attributs peuvent donc être répartis dans des partitions différentes.

• L'opération de partitionnement est une combinaison de projections et de sélections.

• L'opération de recomposition est une combinaison de jointures et d'unions.

La relation *Client* est obtenue avec : *(Cli3 ∪ Cli5) * Cli4 * Cli6*

Relation *Cli3* $\pi_{[NoClient,\ NomClient]}\ (\sigma_{[Age < 38]}Client)$

Relation *Cli5* $\pi_{[NoClient,\ NomClient]}\ (\sigma_{[Age \geq 38]}Client)$

Relation *Cli4* $\pi_{[NoClient,\ Prénom]}Client$

Relation *Cli6* $\pi_{[NoClient,\ Age]}Client$

3.1.5. Répartition des Réseaux connexes d'occurrences (frag. horizontale dérivée)

Dans l'exemple bancaire précédent, on peut fragmenter les agences avec leurs clients avec leurs comptes. On obtient alors deux réseaux d'occurrences liées. Le premier est relatif à Genève, et le deuxième est relatif à Lausanne.

3.2. Définition des fragments

Le principe est de baser la fragmentation sur l'ensemble des requêtes d'interrogation ou de mise à jour prédéfinies. Il faut extraire de ces requêtes toutes les conditions de sélections, les attributs projetés et les jointures. Les opérations de sélection sont à la

base des fragmentations horizontales, les opérations de projection sont à la base des fragmentations verticales.

Pour éviter le risque d'émietter la base de données, on peut restreindre le nombre de requêtes prises en considération. On ne s'intéresse alors qu'aux requêtes les plus fréquentes ou les plus sensibles (celles pour lesquelles le temps de réponse maximum est borné).

3.2.1. Définition des fragments horizontaux d'une classe

Soient c_1, c_2, ..., c_n les conditions de sélection qui ont été extraites des requêtes. Comme les fragments horizontaux doivent être exclusifs, on produit l'ensemble des 2^n conjonctions de condition où chaque condition élémentaire est prise dans sa forme positive ou dans sa forme négative :

$$CC = \{ \bigwedge_{i=1,n} C_i^* \quad \text{ou } C_i^* \text{ est soit } c_i \text{ soit } \neg c_i \}.$$

On ôte de cet ensemble les conjonctions de condition qui sont toujours fausses, et on simplifie les autres.

3.2.2. Définition des fragments verticaux d'une classe

Les fragments verticaux sont exclusifs, sauf en ce qui concerne le (ou les) attribut(s) de jointure qui sont communs à tous les fragments, et ce pour que la décomposition soit sans perte d'information.

On procède comme suit,

1. Extraire toutes les expressions de projection concernées par les requêtes.
2. Ajouter à chacune d'elles le(s) attribut(s) de jointure si elle ne les possède pas.
3. Générer le complément de chaque expression (c'est à dire les autres attributs) en ajoutant le (ou les) attribut(s) de jointure.

L'étape suivante consiste, pour chaque fragment $\mathbf{F}_i$ produit par la fragmentation horizontale, à rechercher toutes les requêtes qui concernent ce fragment et à prendre les expressions de projection de ces requêtes.

A partir des *n* expressions de projection qui concernent $\mathbf{F}_i$, il faut produire l'ensemble des 2^n intersections des expressions de projection.

Exercice 1

Objectif : Fragmenter la relation *Compte* (NoClient, Agence, TypeCompte, Somme).

Proposer un schéma de fragmentation horizontale, puis verticale en tenant compte des requêtes suivantes :

R1 = $\pi_{[\text{NoClient, Agence}]}\ (\sigma_{[(\text{TypeCompte = 'courant'}) \wedge (\text{Somme > 100 000})]}\ \text{Compte})$

R2 = $\sigma_{[\text{Agence = 'Lausanne'}]}\ \text{Compte}$

R3 = $\pi_{[\text{NoClient, Somme}]}\ (\sigma_{[(\text{Agence = 'Genève'}) \wedge (\text{TypeCompte = 'courant'})]}\ \text{Compte})$

Exercice 2

Rémunération (Titre, salaire)

Employé (numE, nomE, Titre)

Projet (numP, nomP, budget, ville)

Affectation (numE, numP, responsabilité, durée)

EMP

ENO	ENAME	TITLE
E1	J. Doe	Elect. Eng
E2	M. Smith	Syst. Anal.
E3	A. Lee	Mech. Eng.
E4	J. Miller	Programmer
E5	B. Casey	Syst. Anal.
E6	L. Chu	Elect. Eng.
E7	R. Davis	Mech. Eng.
E8	J. Jones	Syst. Anal.

ASG

ENO	PNO	RESP	DUR
E1	P1	Manager	12
E2	P1	Analyst	24
E2	P2	Analyst	6
E3	P3	Consultant	10
E3	P4	Engineer	48
E4	P2	Programmer	18
E5	P2	Manager	24
E6	P4	Manager	48
E7	P3	Engineer	36
E8	P3	Manager	40

PROJ

PNO	PNAME	BUDGET	LOC
P1	Instrumentation	150000	Montreal
P2	Database Develop.	135000	New York
P3	CAD/CAM	250000	New York
P4	Maintenance	310000	Paris

PAY

TITLE	SAL
Elect. Eng.	40000
Syst. Anal.	34000
Mech. Eng.	27000
Programmer	24000

1. Proposer un schéma de fragmentation horizontale pour la relation *Rémunération*.
2. A partir des requêtes R1 et R2, proposer un schéma de fragmentation horizontale pour la relation *Projet*.

   R1 : SELECT nomP, budget FROM Projet WHERE ville = valeur ;

   R2 : SELECT * FROM projet WHERE budget < 200 000 ;

3. Fragmenter *Employé* selon les fragments de *Rémunération*.
4. Quels sont les choix de fragmentation de *Affectation*.

Exercice 3

Considérons la relation

Projet (numP, nomP, budget, ville).

Objectif : Trouver un schéma de fragmentation verticale pour la relation *Projet*, en tenant compte des requêtes suivantes :

R1 : SELECT budget FROM projet WHERE numP = valeur;

R2: SELECT nomP, budget FROM projet;

R3: SELECT nomP FROM projet WHERE ville = valeur;

R4: SELECT SUM(budget) FROM projet WHERE ville = valeur;

1. Construire la matrice d'utilisation *Ut*, définie comme suit :

   $Ut(Ri, Aj)$ = 1 ssi la requête *Ri* utilise l'attribut *Aj*

   $Ut(Ri, Aj)$ = 0 sinon.

2. Construire la matrice d'affinité Aff, définie comme suit :

$Aff(Ai, Aj) = \Sigma_{Ut(Rk,\ Ai)=1 \text{ et } Ut(Rk,\ Aj)=1}\ \Sigma_{\text{sites } l}\ Ref_l(R_k) * Acc_l(R_k)$

$Ref_l(R_k)$ est le nombre d'accès aux attributs *Ai* et *Aj* pour une exécution de R_k sur le site l.

$Acc_l(R_k)$ représente la fréquence d'accès à la requête R_k sur le site l.

Sachant que : $\forall R_k$, site l, $Ref_l(R_k) = 1$

$Acc_1(R_1) = 15$	$Acc_2(R_1) = 20$	$Acc_3(R_1) = 10$
$Acc_1(R_2) = 5$	$Acc_2(R_2) = 0$	$Acc_3(R_2) = 0$
$Acc_1(R_3) = 25$	$Acc_2(R_3) = 25$	$Acc_3(R_3) = 25$
$Acc_1(R_4) = 3$	$Acc_2(R_4) = 0$	$Acc_3(R_4) = 0$

3. A partir de la matrice d'affinité obtenue, proposer une fragmentation verticale pour la relation *Projet*.

4. Schéma d'allocation

L'affectation des fragments sur les sites est décidée en fonction de l'origine prévue des requêtes qui ont servi à la fragmentation. Le but est de placer les fragments sur les sites où ils sont le plus utilisés, et ce pour minimiser les transferts de données entre les sites.

L'allocation peut se faire avec réplication ou sans réplication. Sachant que la réplication favorise les performances des requêtes et la disponibilité des données, mais est coûteuse en considérant les mises à jour des fragments répliquas.

Exercice 4

Nous associons aux requêtes de l'exercice 1 : *R1, R2 et R3* les origines suivantes :

- *R1* peut être émise de Genève ou de Lausanne,

- *R2* est presque exclusivement émise de Lausanne,

- *R3* est le plus souvent émise de Genève, mais peut aussi provenir de Lausanne,
- Les mises à jour des comptes de Lausanne sont faites à partir de Lausanne,
- Les mises à jour des comptes de Genève sont faites à Genève.

Sachant que, les mises à jours sont secondaires par rapport aux requêtes dans le sens où elles peuvent être effectuées en différé. L'allocation des fragments est donc basée en priorité sur les sites des requêtes et en second sur les sites de mises à jour.

Proposer un schéma d'allocation des fragments aux sites de Genève et de Lausanne.

5. Réplication

Dans le cas où les utilisateurs n'auraient pas besoin d'accéder aux données les plus récentes, une alternative existe pour éviter le trafic qu'engendre l'accès aux données à jour. Elle consiste en l'utilisation de clichés (ang. *snapshot*). Un cliché représente un état de la base de données à un instant donné.

La pertinence d'un cliché diminue donc au fur et à mesure que le temps passe.

On peut en effet se passer de l'information exacte pour diverses raisons. Certaines informations ne subissent pas souvent de modification (comme le nom de famille, l'adresse ou le nombre d'enfants des employés) et par conséquent une copie même ancienne de ces informations est, dans sa grande majorité, exacte.

Les deux critères qui sont à prendre en compte pour définir l'intérêt d'un cliché sont d'une part l'ancienneté du cliché, et d'autre part le temps d'attente qui serait nécessaire avant d'obtenir l'information originale (à jour). Ces deux informations, l'ancienneté et le temps d'attente, peuvent être pondérées par un taux de satisfaction pour le système d'information.

Exemple : Dans un milieu bancaire les retraits d'argent par un client ne devraient être autorisés que si le compte est suffisamment approvisionné. Comme les ordres de virement et chèques émis au cours des heures écoulées ne sont pas encore connus et donc non débités, la situation réelle du compte peut être bien différente de l'état correspondant dans la base de données. Par conséquent, la décision d'autoriser un retrait d'argent peut être prise en fonction d'un cliché légèrement ancien plutôt que d'accéder à l'information originale qui ne sera que légèrement plus fiable. Cependant, une information exacte permettrait de contrecarrer des retraits d'argent successifs dans deux distributeurs à quelques minutes d'intervalle.

Exemples de Taux de satisfaction du cliché par rapport à l'original en fonction de l'âge du cliché :

5 minutes : 0.95	3 jours : 0.3	1 Heure : 0.9
1 semaine : 0.1	1 jour : 0.7	1 mois : 0.01

Exemples de Taux de satisfaction du temps d'attente pour obtenir l'original :

jusqu'à 30 sec : 1	2 min : 0.5	45 sec : 0.9
3 min : 0.1	1 min : 0.7	5 min : 0.01

Si l'on dispose d'un cliché vieux d'un jour, le temps d'attente maximum pour la réception de l'original est d'une minute. Passé ce délai, l'information contenue dans le cliché sera utilisée.

6. Traitement & Optimisation de Requêtes Réparties

Les règles d'exécution et les méthodes d'optimisation de requêtes définies pour un contexte centralisé sont toujours valables, mais il faut prendre en compte d'une part la fragmentation et la répartition des données sur différents sites et d'autre part le problème du coût des communications entre sites pour transférer les données. Le problème de la fragmentation avec ou sans duplication concerne principalement les mises à jours tandis que le problème des coûts des communications concerne surtout les requêtes.

6.1. Mise à jour de BD réparties

La principale difficulté réside dans le fait qu'une mise à jour dans une relation du schéma global se traduit par plusieurs mises à jours dans différents fragments. Il faut donc identifier les fragments concernés par l'opération de mise à jour, puis décomposer en conséquence l'opération en un ensemble d'opération de mise à jour sur ces fragments.

Insertion

Retrouver le fragment horizontal concerné en utilisant les conditions qui définissent les fragments horizontaux, puis insertion du tuple dans tous les fragments verticaux correspondants.

Suppression

Rechercher le tuple dans les fragments qui sont susceptibles de contenir le tuple concerné, et supprimer les valeurs d'attribut du tuple dans tous les fragments verticaux.

Modification

Rechercher les tuples, les modifier et les déplacer vers les bons fragments si nécessaire.

Exercice 5

Déterminer les fragments (de l'exercice 1) mis à jour par les requêtes suivantes :

1. Création d'un nouveau compte pour un client.

   INSERT INTO compte VALUES (184 177, Genève, courant, 350 000) ;

2. Ajout de 80 000 francs sur le compte courant d'un client.

   UPDATE Compte

   SET somme = somme + 80 000

   WHERE NoClient = 177 498 AND TypeCompte = 'courant' ;

6.2. Requêtes sur BDs réparties

Comme pour le traitement de requêtes en Bases de données centralisées, on produit l'arbre algébrique de la requête. Chaque feuille de l'arbre représente une relation, et chaque nœud représente une opération algébrique. On enrichit l'arbre avec les informations sur la répartition des données sur les différents sites, en particulier sur le site où chaque opération de la requête doit être exécutée.

La complexité d'une requête dans une base de données répartie est définie en fonction des facteurs suivants :

- Entrées/ Sorties sur les disques (*disks I/Os*), c'est le coût d'accès aux données.
- Coût CPU : c'est le coût des traitements de données pour exécuter les opérations algébriques (jointures, sélections ...).
- Communication sur le réseau : c'est le temps nécessaire pour échanger un volume de données entre des sites participant à l'exécution d'une requête.

Dans une base de données centralisée, seuls les facteurs E/Ss et CPU déterminent la complexité d'une requête.

Notons que nous faisons la distinction entre le *coût total* et le *temps de réponse global* d'une requête:

- Coût total : c'est la somme de tous les temps nécessaires à la réalisation d'une requête. Dans ce coût, les temps d'exécution sur les différents sites, les accès aux données et les temps de communication entre les différents sites qui entrent en jeu.

- Temps de réponse global : c'est le temps d'exécution d'une requête. Comme certaines opérations peuvent être effectuées en parallèle sur plusieurs sites, le temps de réponse global est généralement inférieur au coût total.

6.2.1. Transferts de données

Le temps de transmission d'un message tient compte du temps d'accès et de la durée de la transmission (volume des données / débit de transmission). Le temps d'accès est négligeable sur un réseau local, mais peut atteindre quelques secondes pour des transmissions sur de longues distances ou via satellite. Dans ces conditions, un traitement ensembliste des données s'impose. L'unité de transfert entre sites est une relation ou un fragment, et non une occurrence.

6.2.2. Traitement de requêtes réparties

Le but est d'affecter de manière optimale un site d'exécution pour chacune des opérations algébriques de l'arbre. Pour cela, on associe à chacune des feuilles le site sur lequel la relation va être puisée. Lorsqu'une relation est dupliquée, le choix du site de départ est un élément d'optimisation. On cherche ensuite à associer à chaque nœud de l'arbre le site sur lequel l'opération algébrique associée à ce nœud sera exécutée.

Généralement, il faut faire localement tous les traitements qui peuvent y être faits. Ainsi, lorsque toutes les opérandes d'une même opération algébrique sont situées sur le même site, la solution la moins coûteuse pour exécuter cette opération est le plus souvent de l'exécuter sur ce site. Ceci est notamment toujours vrai pour les opérateurs unaires qui font diminuer le volume d'informations (sélection, projection).

Pour diminuer le volume de données transmis d'un site à un autre, il faut limiter les transferts d'information aux seules informations utiles. Pour cela il faut systématiquement rajouter des projections dans l'arbre algébrique pour abandonner les attributs inutiles. Il faut aussi noter que les parties de requêtes indépendantes peuvent être exécutées en parallèle sur des sites différents et donc baisser le temps total d'exécution.

6.2.3. Optimisation dynamique des requêtes

Après avoir généré un arbre de requête, la stratégie adoptée pour l'exécution est ascendante. C'est à dire que l'affectation de chaque nœud de l'arbre à un site peut être décidée en cours d'exécution en fonction des différents volumes de données intermédiaires obtenus sur les sites. On part donc des feuilles, où l'on connaît les données (type, volume, statistiques sur la répartition des occurrences dans les domaines de valeurs...) pour remonter au niveau supérieur et prendre une décision sur le site d'affectation de l'opérateur. Et ainsi de suite pour les niveaux supérieurs jusqu'à la racine. Il faut cependant noter que cette stratégie n'est pas optimale si elle est effectuée en aveugle.

D'une manière générale, il faut tenir compte des volumes de données de chaque relation, et de leur composition. Il est en effet parfois intéressant de tenir compte du calcul effectué par un site avant que les autres sites se mettent à travailler.

6.2.4. Semi-jointure

Les BDR ont abouti à la définition d'un nouvel opérateur, la semi-jointure, qu'il est parfois intéressant d'utiliser. Il s'agit en fait d'une double jointure : le principe est d'effectuer deux petites jointures plutôt qu'une grosse; c'est à dire deux petites transmissions de données plutôt qu'une seule beaucoup plus volumineuse.

La semi-jointure réduit la taille des opérandes des relations. Elle permet de réduire la taille des données à transmettre.

Soient *R1* et *R2* deux relations se trouvant respectivement sur les sites *S1* et *S2*.

But : Evaluer $R1 \bowtie R2$ sur le site *S1*.

L'algorithme de semi-jointure se déroule comme suit,

S1> $temp1 \leftarrow \pi_{R1 \cap R2}(R1)$

S1> envoi de *temp1* vers *S2*

S2> $temp2 \leftarrow R2 \bowtie temp1$

S2> envoi de *temp2* vers S1

S1> $R1 \bowtie temp2$ (équivalent à $R1 \bowtie R2$)

Exercice 6

Soit la BDR composée des relations suivantes :

P (NP, NOMP, MADE_IN, COULEUR, POIDS)

U (NU, VILLEU, NOMU)

F (NF, NOMF, VILLEF, ADRESSE, PAYS, COEF)

PUF (NP, NU, NF, DATE, QUANTITE)

Ville(nom, région, pays, description)

Les relations U et PUF sont sur le site A.

Les relations F et P sont sur le site B.

Le réseau reliant les deux sites A et B a les caractéristiques techniques suivantes :

- temps d'accès d'un site à un autre : 0,5 seconde
- débit de transmission : 9 600 bauds (soit ≈ 1 000 octets/ sec)

Un message est donc transféré en (0,5 + Nb-d'octets-du-message /1000) secondes.

|F| = 100 enregs, de 120 octets,

|U| = 1000 enregs de 60 octets,

|P| = 10000 enregs de 90 octets,

|PUF| = 1 000 000 enregs de 10 octets,

|Ville| = 200 enregs de 240 octets,

P contient 500 produits italiens.

U contient 100 usines de Lausanne.

Nbre de livraisons de produits à des usines situées à Lausanne : 100000.

Nbre de livraisons de produits italiens : 5000.

Les numéros NU, NP et NF sont codés sur 2 octets.

Le nom de la ville sur 30 octets.

Requête1 : "Donner les numéros et les noms des fournisseurs qui ont livré un produit italien à une usine située à Lausanne", qui provient du site $\mathbb{B}$, correspond la requête algébrique suivante :

$$R = \Pi_{[NOMF,\ NF]} (\sigma_{[VILLEU\ =\ "Lausanne"]}U * PUF * \sigma_{[MADE_IN\ =\ "Italy"]}P * F)$$

Pour chacune des stratégies suivantes, dessinez l'arbre de la requête, et calculer le temps de communication total.

1. Envoyer P et F sur le site A et exécution de la requête sur A.
2. Envoyer U et PUF sur le site B et exécution de la requête sur B.
3. Sélectionner les usines de Lausanne sur A, faire la jointure avec PUF, et la projection sur (NP, NF). Transmettre la nouvelle relation au site B pour exécution de la requête.
4. Sélectionner les numéros (NP) des produits italiens sur B. Transmettre ces numéros au site A en lui demandant qu'il renvoie les numéros des fournisseurs qui ont livré un de ces produits à une usine de Lausanne. Le site A transmet un ensemble de numéros de fournisseurs à B.

Requête2 : "on recherche sur le site A les couples (fournisseur, usine) tel que le fournisseur habite dans la ville où se trouve l'usine".

- Les 100 fournisseurs habitent dans 60 villes différentes.
- Les 1000 usines sont réparties sur 840 villes différentes.
- 5 fournisseurs habitent dans une ville où se trouve au moins une usine
- 100 usines sont situées dans une ville où habitent un ou plusieurs fournisseurs

Pour chacune des stratégies suivantes, calculer le nombre d'octets transférés entre sites

1. Si le site A envoie U à B.
2. A envoie à B les noms de villes d'usine, puis B renvoie les fournisseurs retenus.
3. B transmet à A les noms de ville différents, A lui renvoie le résultat de la

(semi-)jointure entre ces villes des fournisseurs et les villes des usines. Enfin B fait la seconde semi-jointure et envoie à A les fournisseurs retenus.

6.2.5. Autres Optimisations

Décomposition de requête

Ce processus est commun aux bases de données centralisées et réparties.

En première étape, la requête est réécrite sous forme normalisée. La restriction de la requête (les prédicats qui suivent la clause WHERE) est écrite sous la forme conjonctive, càd une forme de conjonctions de disjonctions de prédicats. La requête normalisée est analysée afin de rejeter les requêtes qu'il serait impossible de traiter, exemple rejet pour mauvais typage des prédicats. Puis, grâce aux règles d'idempotences pour les opérations logiques, les redondances sont éliminées.

Exercice 7

Réécrire les requêtes suivantes :

(R1) SELECT *

FROM square

WHERE length* length < 10 000 ;

(R2) SELECT Title

FROM Employee

WHERE (NOT (Title = "programmer") AND ((Title = "programmer") OR

(Title = "Elect. Eng.")) AND NOT(Title = "Elect. Eng.")) OR

(Name = "J.Doe");

Localisation des données réparties

A ce niveau, on prend en compte que le fait que les données sont réparties et fragmentées. Des sélections sur les fragments qui ont des restrictions contraires aux restrictions qui ont permis de générer ces fragments, engendrent des relations vides.

En effet, dans le cas de jointures avec des fragments horizontaux on rencontre les deux suivants :

1er cas:

$A = A1 \cup A2 \Leftrightarrow A \bowtie B = (A1 \cup A2) \bowtie B = (A1 \bowtie B) \cup (A2 \bowtie B)$

Ceci est intéressant surtout dans le cas ou B est répliquée sur le site qui contient A1 et sur le site qui contient A2.

2éme cas:

A et B sont partitionnées tel que, $Ai \bowtie Bj = \emptyset$ Si $i \neq j$

Exemple : cas des relations *Employé* et *Département* partitionnées par ville département

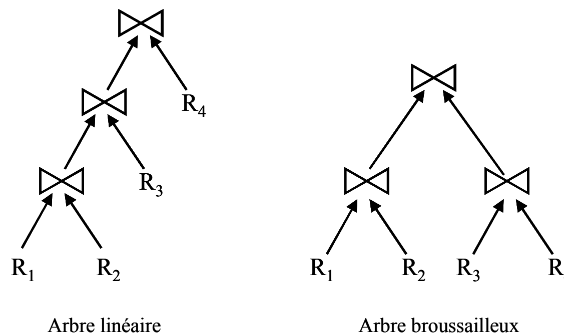
Parallélisme intra-requêtes

Utilisation de plusieurs threads, pour accèlérer l'exécution des requêtes. Par exemple, un site désire faire l'union de 3 tampons à recevoir de 3 sites différents, nécessite au moins un thread qui réceptionne les tampons et un thread qui traite les tampons reçus.

Arbre

Parmi les heuristiques prises en compte, est celle proposée par J. D. Ullman. Elle consiste en l'application des opérateurs unaires le plutôt possible afin de réduire la taille des relations intermédiaires.

Une autre heuristique concerne la forme de l'arbre de la requête. On distingue deux types d'arbres : les arbres linéaires et les arbres broussailleux (ang. *bushy trees*).



Dans le cas de BDs réparties, l'utilisation des opérateurs broussailleux permet d'augmenter le parallélisme et d'améliorer les temps de réponses des requêtes.

Statistiques

Pour une relation R définie par les n attributs $\{A_1, A_2 ..., A_n\}$ et fragmentée en $R_1, R_2 ..., R_r$ les données statistiques sont typiquement :

1. la longueur de chaque attribut A_i, ($long\ (A_i)$)
2. le nombre de valeurs distinctes pour chaque attribut A_i de chaque fragment R_j, ($card(\Pi_{Ai}(R_j))$)
3. le minimum et le maximum de valeurs pour chaque attribut ($min(A_i)$ et $max(A_i)$)
4. la cardinalité de chaque attribut, c'est à dire le nombre de valeurs uniques sur le domaine d'un attribut ($card\ (dom[A_i])$)
5. la cardinalité de chaque fragment ($card(R_i)$).

7. Gestion des Transactions Réparties

7.1. Définitions

- Une transaction est un ensemble d'opérations menées sur une BD,
- Ces opérations peuvent être en lecture et/ou écriture,
- Une opération est atomique, c'est donc une unité indivisible de traitement,
- Une transaction est soit validée par un *commit*, soit annulée par un *rollback*, soit interrompue par un *abort*,
- Une transaction a une marque de début (Begin Of Transaction BOT), et une marque de fin (End Of Transaction EOT).

La cohérence et la fiabilité d'une transaction sont garanties par 4 propriétés : l'**A**tomicité, la **C**ohérence, l'**I**solation, la **D**urabilité qui font l'***ACID***ité d'une transaction.

- *Atomicité* : cette propriété signifie qu'une transaction est traitée comme une seule opération. Toutes les actions sont toutes menées à bien ou aucune d'entre elles.
- *Cohérence* : une transaction est un programme qui amène la BD d'un état cohérent à un autre état cohérent, tel que toutes les contraintes d'intégrité restent vérifiées.
- *Isolation* : c'est la propriété qui impose à chaque transaction de voir la BD cohérente. Une transaction en exécution ne peut révéler ses résultats à d'autres transactions concurrentes avant d'effectuer le *commit*.
- *Durabilité* :c'est la propriété qui garantit lorsqu'une transaction a effectué son *commit*, le résultat sera permanent, et ne pourra être effacé de la BD quelques soient les pannes du système rencontrées.

7.2. Exemple de Transactions

Cas de transfert d'argent d'un compte épargne vers un compte courant.

```
Begin Transaction TransfertCE/CC
Begin
        Input(somme, numClt)
        EXEC SQL    UPDATE ComptesEpargne
                    SET solde = solde – somme
```

```
                    WHERE numClient = numClt ;
        EXEC SQL    UPDATE ComptesCourant
                    SET solde = solde + somme
                    WHERE numClient = numClt ;
        Output("Transfert compte épargne vers compte courant effectué ")
End
```

7.3. Interférences à éviter

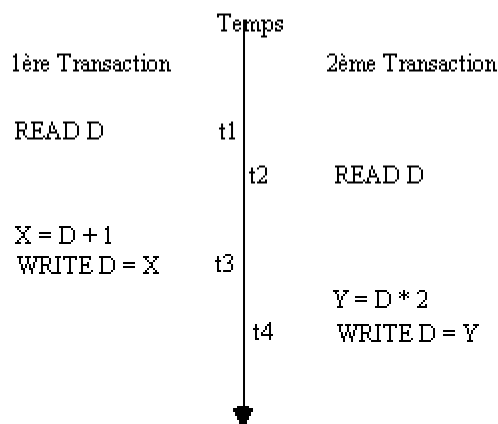
Nous distinguons deux cas d'interférences problématiques.

- Des interférences entre écrivains, càd transactions d'écriture.
- Des interférences entre lecteurs et écrivains, càd transactions de lecture et transactions d'écriture.

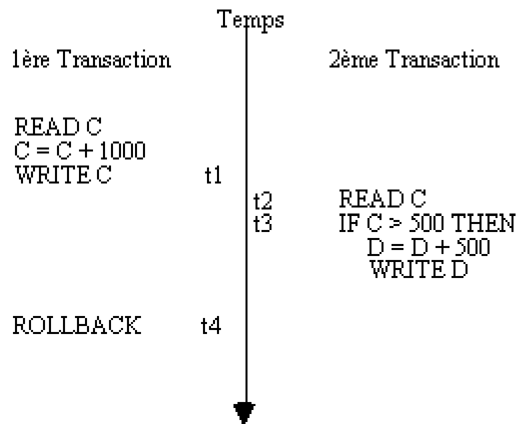
7.3.1. Entre écrivains

Perte d'opération (ang. *lost update*)

La mise à jour faite par la 1ère transaction est perdue.



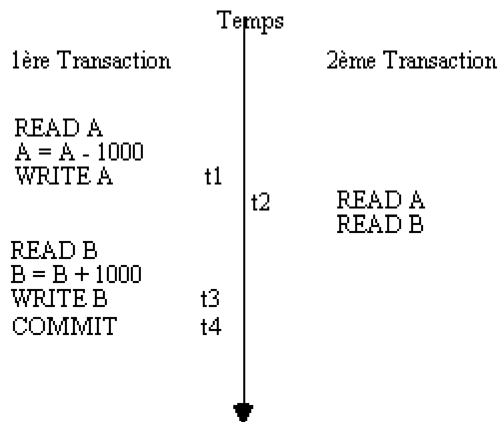
Ecriture inconsistante (ang. *dirty write*)



Initialement, C le crédit d'un compte est < 500. Une 1ère transaction modifie le crédit C d'un compte. Une 2ème lit C dans ce nouvel état, puis, constatant que la provision est suffisante, modifie le débit D du compte. Après annulation de la 1ére transaction, la contrainte d'intégrité D ≤ C n'est plus vérifiée.

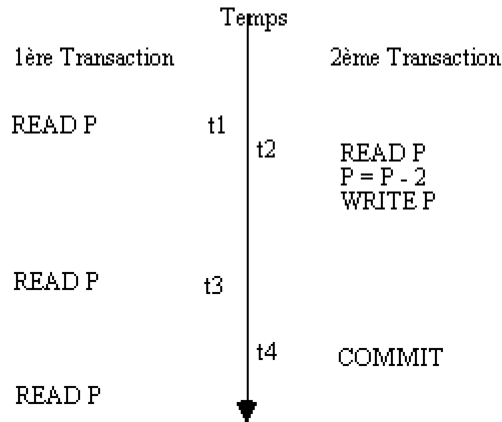
7.3.2. Entre lecteurs et écrivains

Lecture inconsistante (ang. *dirty read*)



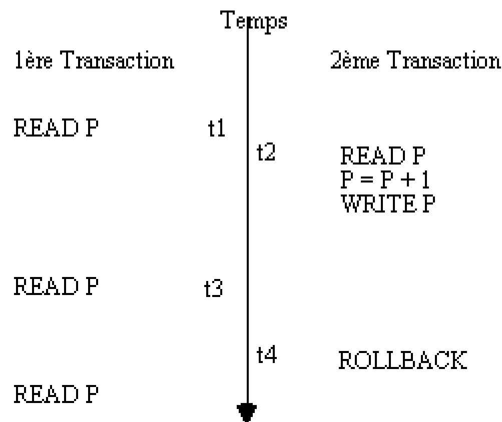
La 1ère transaction a pour but de faire un virement entre deux comptes A et B, qui satisfait la contrainte d'intégrité "somme A+B invariante". La deuxième transaction lit A et B juste après que la 1ére transaction ait déduit 1000 de A. Elle trouve A+B diminué.

Lecture non reproductible (ang. *non-repeatable read*)



Une 1ère transaction consulte les places libres dans un avion et laisse un temps de réflexion à l'utilisateur. Entre temps, une 2ème transaction, qui voit les mêmes places libres, valide deux réservations. La 1ère demande à nouveau les places libres: la liste est diminuée alors qu'elle n'a pas encore choisi ses réservations!

Lecture fantôme (ang. *phantom read*)



Cas identique au précédant mais avec libération au lieu de réservation de place par la 2ème transaction. Il n'y a pas alors de conflit à proprement parler mais seulement interrogation sur la validité de ces "apparitions fantômes".

7.4. Contrôle de concurrence

Plusieurs utilisateurs accèdent simultanément à la BD. L'accès concurrent permet de partager les ressources matérielles et d'améliorer les performances d'accès aux données.

Le contrôle de concurrence est un mécanisme du SGBD, qui contrôle l'exécution simultanée de transactions de manière à produire les mêmes résultats qu'une exécution séquentielle. Cette propriété est la **sérialisabilité**. Une exécution d'un ensemble de transactions est dite sérialisable si elle donne pour chaque transaction participante, le même résultat que l'exécution en séquentiel de ces mêmes transactions.

7.4.1. Les mécanismes utilisés

Les estampilles

Les estampilles permettent d'installer un ordre total entre les actions, et ainsi de les sérialiser. L'estampille est donc un identifiant unique des transactions qui permet en plus de les ordonner.

>> chaque transaction est repérée par un numéro d'ordre unique dans le système : estampille. L'estampille est le couple (valeur compteur : horloge locale, numéro du site).

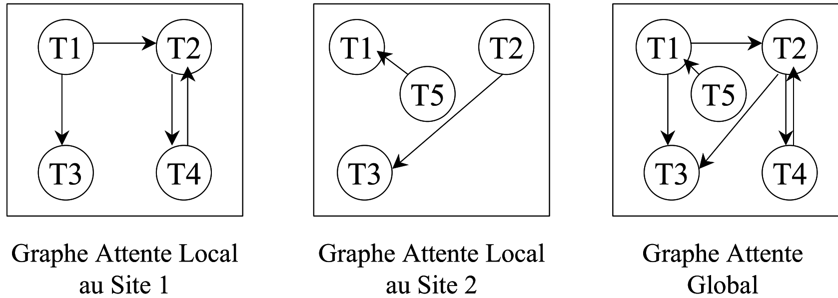
L'inconvénient majeur est la difficulté de synchroniser des sites de différentes horloges.

Verrouillage

La technique la plus répandue pour sérialiser les transactions est basée sur l'utilisation de verrous. On impose que l'accès aux données se fasse de manière mutuellement exclusive.

7.4.2. Interblocages

Toute méthode basée sur le verrouillage peut donner des interblocages lorsque deux transactions s'entre-attendent. Pour illustrer ce cas, on peut prendre un exemple. Une transaction T_i détient un verrou en lecture ou en écriture sur la donnée x. Une transaction T_j détient un verrou en lecture ou en écriture sur la donnée y. La transaction T_i attend un verrou en écriture sur la donnée y et la transaction T_j attend un verrou en écriture sur la donnée x. Il y a dans ce cas un interblocage.
Un outil de grande utilité dans l'analyse des interblocages est le graphe des attentes. Les noeuds du graphe des attentes sont des transactions simultanées et les arcs orientés qui les relient représentent l'attente d'une transaction pour une autre. Grâce à cette représentation, il est facile de caractériser les interblocages : ce sont les cycles sur le graphe. Dans un environnement réparti, les interblocages peuvent mettre en jeu différents sites. Le graphe des attentes local est donc insuffisant et il faut également en maintenir un au niveau global.



Graphe Attente Local au Site 1 | Graphe Attente Local au Site 2 | Graphe Attente Global

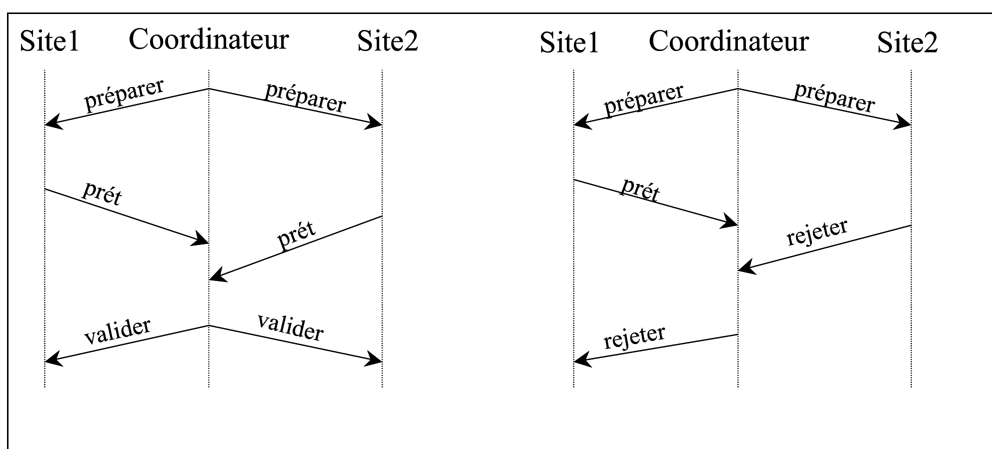
La solution est d'annuler des transactions concurrentes jusqu'à suppression de cycles, et de les reprendre plus tard.

7.4.3. Protocole de validation à deux phases

Une transaction globale est validée ssi toutes les transactions locales qui la composent valident. En effet, si au moins une transaction locale ne valide pas, toutes les transactions sont annulées.

Le protocole exige *un site coordinateur* et *des sites participants*. Il se compose de deux phases :

- *Phase de préparation* : le coordinateur demande à chaque participant de se préparer pour valider la transaction locale.
- *Phase de validation* : le coordinateur ordonne à tous les participants de valider ou d'annuler leur transaction. La décision est prise par le coordinateur tenant compte de la réponse de chaque participant.



Les transactions distribuées présentent deux avantages : (*i*) les données situées sur d'autres serveurs, peuvent être mises à jour, et les instructions UPDATE peuvent être

gérées comme étant une seule unité. (*ii*) utilisation du 2-PC transparente à l'utilisateur.

8. Les Architectures de Systèmes Parallèles

Dans ce qui suit, trois architectures parallèles, définies selon le critère de partage de ressources, et une architecture hybride sont présentées.

8.1. Architecture à mémoire partagée (ang. *Shared-Memory*)

Les disques et les mémoires centrales sont partagés par les processeurs du système.

(+) L'espace d'adressage global rend le système facile à implanter pour les vendeurs de SGBDs. La communication inter-processeurs est rapide, vu l'accès partagé aux mémoires centrales.

(+) Equilibre de charge entre les processeurs facile à réaliser.
L'échec d'un processeur n'entraîne pas la non-possibilité d'accès à données.

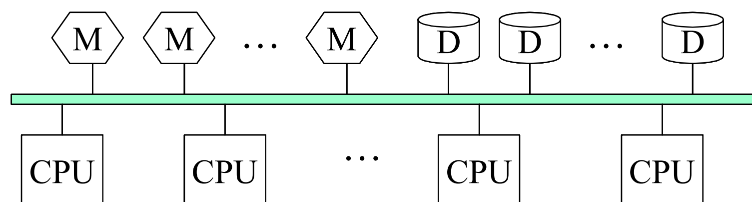
(-) Coût du système.

(-) Accès conflictuels aux mémoires centrales peuvent dégrader les performances.

(-) Le nombre de processeurs est limité à 20-30, un nombre supérieur crée des goulots d'étranglements.

(-) Architecture non scalable.

Exemples de SGBD // : XPRS (U. de berkeley), DBS3 (Bull).



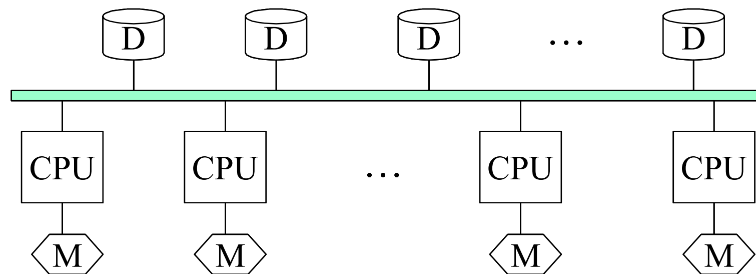
8.2. Architecture à disque partagé (ang. *Shared-Disk ou cluster*)

Chaque processeur a sa mémoire centrale privée, mais les disques sont partagés par tous les processeurs du système.

(+) Equilibre de charge facile à réaliser.
(+) L'échec d'un processeur n'entraîne pas la non-disponibilité des données.

(-) L'inconvénient majeur est lié à la complexité du maintien de la cohérence des caches des processeurs.

(-) L'accès aux disques peut créer un goulot d'étranglement, du à la limite de la capacité du bus.



Exemples : IMS/VS (IBM), VAX DBMS (DEC) …

8.3. Architecture à mémoire distribuée (ang. *Shared-Nothing*)

Chaque processeur a sa propre mémoire centrale et disque.

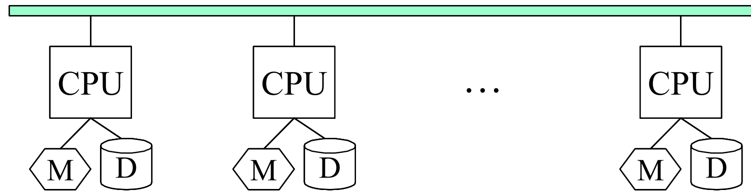
(+) Coût abordable, vu que le système est une collection de PCs.

(-) La haute disponibilité pose un problème. En effet, l'échec d'un processeur rend l'accès aux données impossible. D'où la nécessité de techniques de haute disponibilité (redondance ou duplication). Ceci fait émerger un autre problème de maintien de la consistance des miroirs ou des disques de redondance.

(-) Problème d'équilibre de charge dû au placement pré-déterminé des données.

(-) Coût de transfert sur le réseau de grand volume de données, étant le résultat d'une requête .

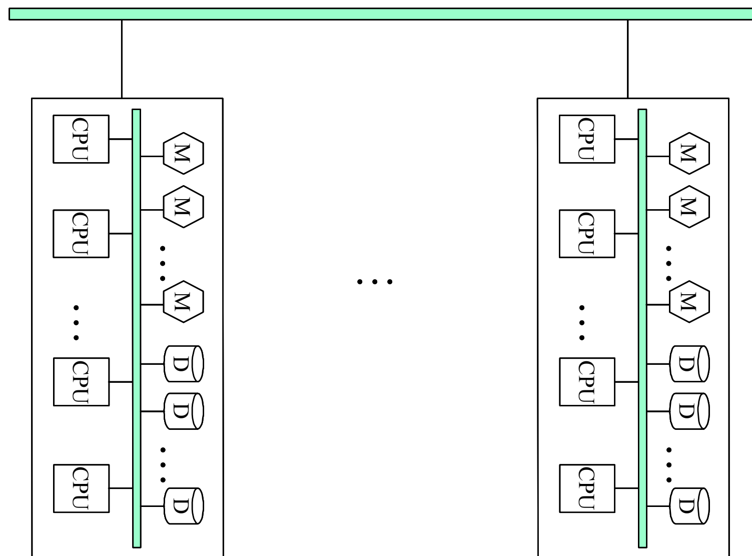
Exemples : GAMMA (U. de Wisconsic), BUBBA (MCC),



8.4. Architectures hybrides

Une architecture hybride à deux niveaux, peut être au niveau interne une architecture à mémoire partagée, et au niveau externe une architecture à mémoire distribuée. De telles architectures combinent les avantages de chaque architecture, et compensent les inconvénients respectifs des architectures.

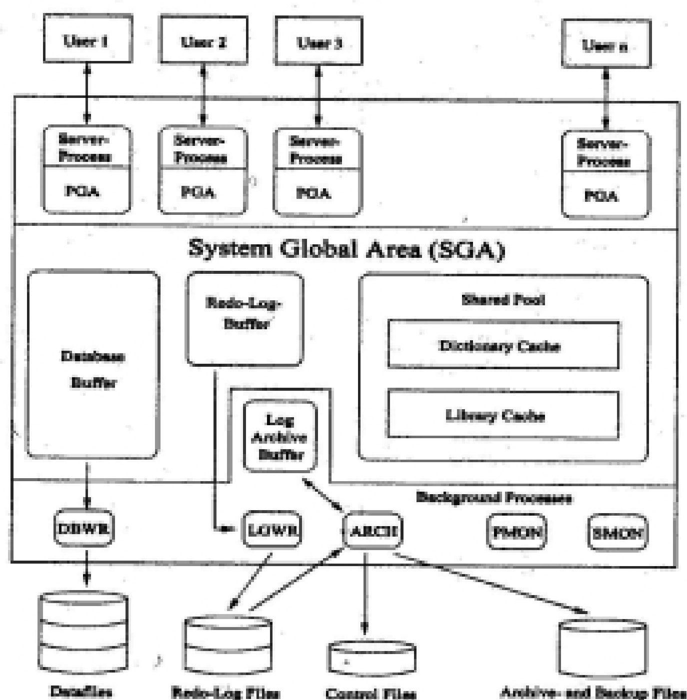
L'architecture hybride illustrée ci-dessous combine l'équilibrage de la charge des architectures à mémoire partagée et l'extensibilité des architectures à mémoire distribuée.



Les techniques de distribution des données sont au nombre de 3 : par intervalle, hachage, donneur de carte (ang. *round robin*).

Partie II : Mécanismes de Répartition dans Oracle8i

9. Introduction à Oracle: objets & architecture



Architecture d'Oracle

Chaque fois qu'une BD est lancée sur un serveur (ang. *instance start up*), une partie de la mémoire centrale dite SGA : *System Global Area*, est allouée ; et plusieurs processus d'arrière-plan sont lancés. Une instance de BD est un ensemble de structures de mémoire et de processus d'arrière-plan qui accèdent à un ensemble de fichiers de données. Dans *Parallel Server* plusieurs instances peuvent accéder à la même BD.

Les paramètres d'initialisation d'une instance sont dans *init.ora*. Le nom du fichier inclut en général le nom de l'instance, si celle-ci est X, le fichier aura pour nom initX.ora.

Gestion des données : table, index, view, materialized view, snapshot …

Stockage physique: cluster, tablespace, …

Stockage d'instructions : procedure, trigger, …

Gestion d'utilisateurs: profile, user, …

9.1. Structures logiques de la BD

- Tablespace : C'est une division logique d'une BD. Chaque BD possède au moins un tablespace appelé SYSTEM. La vue USER_TABLESPACES détaille les *tablespaces* existants.

- *Segments* : les segments sont des structures logiques de stockage des données physiques de la base, ils sont assignés à des *tablespace*. Il en existe quatre types : les segments de données, les segments d'index, les segments temporaires, et les segments de rollback. Les segments d'index, par exemple, stockent les données associées à des index. Afin de maintenir la cohérence en lecture entre plusieurs utilisateurs de BDs, et de pouvoir annuler des transactions, Oracle dispose d'un mécanisme appelé *segments de rollback*. Ces segments servent à reconstruire une image des données antérieure à leur modification dans le cas de transactions non validées par un *commit*. Un segment est constitué de structures appelées *extents*.

- *Extent* : un extent consiste d'une séquence de blocs de données contigus. Quand la taille d'un objet BD augmente, un extent de plus est alloué pour l'objet.

- *Bloc de données*: Un bloc détermine le niveau de granularité le plus fin où les données sont sauvées. Un bloc de données correspond à un nombre spécifique d'octets d'espace BD physique sur disque.

Les vues USER_SEGMENTS et USER_EXTENTS renseignent sur le nombre de blocs de données alloués par objet BD, et le nombre de libres.

Exemple:

```
SELECT  TABLESPACE_NAME, EXTENTS, BLOCKS
FROM USER_SEGMENTS
WHERE SEGMENT_NAME = 'CLIENT';
```

Pour les objets BD (table, index, cluster), qui requièrent leur propre espace de stockage, un segment dans le *tablespace* leur est alloué. ORACLE permet de configurer les paramètres de stockage en utilisant la clause STORAGE.

```
CREATE TABLE  Produit
(       NumProd NUMBER(2)
….)
STORAGE (       INITIAL         1M          NEXT                    400K
                MINEXTENTS      1           MAXEXTENTS   20      PCTINCREASE
                50) ;
```

INITIAL et NEXT spécifient la taille en octet resp. du 1er et du 2éme extent, par défaut ont la taille de cinq blocs. La taille d'un bloc *db_block_size* figure dans le fichier *init.ora*.

MINEXTENTS spécifie le nombre initial d'extensions allouées à la création du segment, la valeur par défaut est égale à 1.

MAXEXTENTS spécifie le nombre maximum d'extensions possibles pour le segment de données possible de la table. La clause UNLIMITED annule la valeur de MAXEXTENTS.

PCTINCREASE définit un pourcentage *p* dont la valeur par défaut est 50, qui signifie que chaque nouvelle extension est 1,5 fois plus grande que la précédente. La croissance est faite à partir du 2éme segment.

Pour la table *Produit*, la taille du 1er extent est 1M, le 2éme : 400K, le 3éme : 600K, le 4éme : 900K etc.

La clause STORAGE définit les paramètres de stockage niveau extent. D'autres paramètres : PCTFREE et PCTUSED, opèrent niveau bloc.

PCTFREE : est le pourcentage de l'espace libre réservé dans chaque bloc pour les modifications ultérieures. la valeur par défaut est égale à 10.

PCTUSED : est le pourcentage minimum de remplissage qui est maintenu dans chaque bloc. la valeur par défaut est égale à 40.

9.2. Structures BD physiques

Il existe 4 types de fichiers :

- *Fichiers de données* : (ang. *datafiles*) un *tablespace* est constitué d'un ou plusieurs fichiers de données stockés sur disque. Un fichier appartient à un seul *tablespace*, et une fois qu'un fichier a été ajouté à un *tablespace*, il ne peut plus en être retiré ni être associé à un autre *tablespace*.

- *Fichiers Redo-log* : Oracle consigne toutes les transactions de la base dans les fichiers du journal de reprise. Ces fichiers sont utilisés pour rétablir les transactions dans l'ordre approprié en cas de défaillance de la base.

- *Fichiers de contrôle* : l'architecture physique de la base est maintenue au moyen de ses fichiers de contrôle. Ceux-ci enregistrent des informations de contrôle relatives à tous les fichiers de la base. Ils garantissent la cohérence interne et guident les opérations de récupération.

- *Fichiers trace et journal d'alerte* : chaque processus d'arrière-plan qui s'exécute dans une instance possède un fichier trace associé, qui contient des informations sur les événements importants, et les erreurs internes qu'il rencontre. En plus des fichiers *trace*, Oracle maintient un *journal d'alertes*, qui consigne les messages d'erreurs, les exceptions et les opérations administratives.

9.3. Structures de mémoire

L'ensemble des structures de mémoire permet d'améliorer les performances de la BD en limitant le nombre d'opérations d'E/Ss exécutées sur les fichiers de données.
La zone SGA, regroupe un ensemble de structures de mémoire partagées qui contiennent les données et les informations de contrôle concernant une instance Oracle. Lorsque plusieurs utilisateurs sont connectés à la même instance, les données de cette zone sont partagées entre eux.

- *Le cache de tampons de blocs de données* : (ang. d*ata block buffer cache*), ce cache contient une copie des blocs de données lus à partir des segments de données de la base, tels que ceux de tables, d'index et de clusters. Le paramètre DB_BLOCK_BUFFERS, contenu dans le fichier *init.ora*, indique la taille du cache en nombre de blocs de données. La cache représente 1 à 2% de la taille de la base. Oracle gére l'espace disponible par l'algorithme LRU : *Least recently Used*.

- *Le tampon du journal de reprise* : (ang. *redo log buffer*), les fichiers de journal de reprise décrivent les modifications apportées à la BD. Avant d'être enregistrées dans les fichiers *redo log*, les transactions sont d'abord placées dans la zone SGA : *tampon redo log*. La base enregistre régulièrement par lots dans les fichiers *redo log*. La taille de ce tampon est définie au moyen du paramètre LOG_BUFFER du fichier *init.ora*.

- *Le cache du dictionnaire* : (ang. *dictionary cache*), les tables du dictionnaire contiennent les noms des fichiers de données, les noms de segments, les emplacements des extents, les descriptions de table et les privilèges. Lorsque la base a besoin d'un de ce type d'information, elle lit les tables du dictionnaire, et place les données extraites dans le cache du dictionnaire de la zone *SGA*. Ce cache est géré au moyen d'un algorithme LRU, il fait partie de la zone SQL partagée, dont la taille est définie à l'aide du paramètre SHARED_POOL_SIZE du fichier *init.ora*.

- *Le pool partagé* : (ang. *shared pool*), est la partie du SGA utilisée par tous les utilisateurs. Il contient le cache du dictionnaire et le cache de bibliothèques (ang. *library cache*). Ce dernier, maintient des informations sur les instructions exécutées dans la BD. Ce pool contient la représentation analysée (*parse tree*) et le plan d'exécution et des instructions SQL qui ont été exécutées. La deuxième fois qu'une instruction SQL identique est émise, les informations analysées du pool sont exploitées pour accélérer son exécution. Ce pool est géré au moyen d'un algorithme LRU. La taille du pool est définie à l'aide du paramètre SHARED_POOL_SIZE du fichier *init.ora*, et est exprimée en octets.

9.4. Processus d'arrière-plan

La relation entre les structures physiques et les structures de mémoire est maintenue, et mise en œuvre au moyen de processus d'arrière-plan (ang. *background processes*). Ces processus sont dédiés à une instance. Chaque processus d'arrière-plan crée un fichier trace, qui est maintenu pendant la durée d'activité de l'instance, dans lequel sont consignés les événements majeurs et les erreurs internes qu'il rencontre.

Dans ce qui suit, on décrit à titre non exhaustif les principaux processus:

- *SMON –System Monitor*: le processus entreprend une récupération après un crash. Il nettoie la BD des transactions avortées et les objets impliquées, et de la coalition des extents libres contigus afin d'avoir des extents larges.

- *PMON –Process Monitor*: le processus s'occupe de la récupération de processus utilisateurs défaillants. Il libère également le cache de blocs de données ainsi que les ressources qui étaient exploitées par l'utilisateur.

- DBWr –Database Writer: le processus d'écriture gère le contenu du cache de blocs de données et le cache du dictionnaire en réalisant des opérations d'écriture par lots des blocs de données modifiés de la zone SGA vers les fichiers de données. Alors qu'il n'existe qu'un seul processus SMON ou PMON par instance, plusieurs processus DBWr peuvent s'exécuter. Pour limiter les risques de contentions, le paramètre DB_WRITER_PROCESSES du fichier *init.ora* définit le nombre de processus.

- *LGWR –Log Writer*: le processus d'écriture dans le journal de reprise LGWR gère l'écriture par lots des entrées du tampon *redo log* dans les fichiers *redo log*. Ce tampon maintient l'état le plus à jour de la base.

- *CKPT –Checkpoint*: le processus de contrôle CKPT provoque l'exécution de DWRn. Ce dernier décrit tous les blocs de données qui ont été modifiés depuis le dernier point de contrôle dans les fichiers de données, puis met à jour les en-têtes de ces fichiers. Le paramètre LOG_CHECK_POINT_INTERVAL du fichier *init.ora* de l'instance définit la fréquence des points de contrôle.

- *ARCh –Archiver*: Le processus LGWR écrit dans les fichiers *redo log*. A chaque fois qu'un fichier devient plein, il écrit dans le suivant, et une fois le dernier fichier rempli, il écrase le contenu du premier. Il est possible de lancer une instance BD en mode archive-log. Dans ce cas, le processus ARCh copie les fichiers redo-log, avant que les entrées soient écrasées par le LGWR. Ainsi, il est possible de restaurer le contenu de la BD une fois que le mode archive lancé.

- *RECO –Recoverer*: le processus de récupération RECO résout les défaillances dans les configurations de bases de données distribuées, et ce, en tentant d'accéder aux bases impliquées dans des transactions distribuées douteuses. Ce processus est défini dans la plate-forme qui supporte l'option *Distributed*, tel que le paramètre DISTRIBUTED_TRANSACTIONS du fichier *init.ora* est défini avec une valeur non nulle.

- *SNPn –Job Queue*: Les processus de file de tâches SNPn s'activent régulièrement. Ils actualisent les *snapshots*. Le paramètre JOB_QUEUE_PROCESSES du fichier *init.ora* définit le nombre de processus.

- *USER* : le processus communique avec d'autres processus lancés par les programmes application, tel que SQL*Plus. Le processus USER est alors responsable de l'envoi des requêtes au SGA, ça inclut la lecture des blocs des données.

9.5. Etapes de traitement d'un ordre SQL

Supposons qu'un utilisateur, travaillant sur SQL*Plus, émette une requête de mise à jour sur la table *T*, et que plusieurs tuples sont affectés par la requête. La requête est passée au serveur par le processus USER. Le serveur (exactement le *query processor*) vérifie si la requête existe déjà dans le *cache de bibliothèques*. Si non trouvée, elle est analysée, vérifier pr au *cache du dictionnaire* les privilèges et les attributs etc.., générer un plan d'exécution. La représentation analysée et le plan d'exécution sont alors sauvés dans le *cache de bibliothèques*.

Pour les objets affectés par la table *T*, on vérifie si les blocs de données sont dans le *cache de tampons de blocs de données* . Si non trouvés, le processus USER, les met dans le *cache de tampons de blocs de données*.
Avant que la mise à jour des tuples soit faite, 'l'image avant' des tuples est écrite sur les *segments de rollback* par le processus DBWr.
Pendant que le *tampon redo-log* est rempli durant la modification des blocs de mise à jour, le processus LGWR écrit le contenu du *tampon redo-l*og dans les fichiers redo-log.
Après la fin de mise à jour, l'utilisateur valide la mise à jour par un *commit*.
Tant qu'un *commit* n'a pas été exécuté, les modifications peuvent être annulées par un *rollback*. Dans ce cas, les blocs de données mis à jour dans le tampon BD sont écrasés par les blocs originaux sauvés dans les *segments de rollback*.
Si l'utilisateur exécute un commit, l'espace alloué pour les blocs dans les *segments de rollback* est désalloué, et peut être utilisé par d'autres transactions.

10. Oracle en réseau

* *Oracle Net services* fournit des solutions de *connectivité* dans des environnements distribués. Il est composé de:

1. Oracle Net
2. Modules d'écoute/listeners

   le fichier de configuration *LISTENER.ORA* contient :

   – son nom, par défaut LISTENER

   – son adresse (HOST et PORT) : (ADDRESS = (PROTOCOL = TCP) (HOST = localhost) (PORT = 1521)

   – lesSIDs (Service ID) des BD guettées
3. Oracle Connection Manager
4. Outils de configuration et de gestion : Oracle Net Configuration Assistant, Oracle Net Manager.

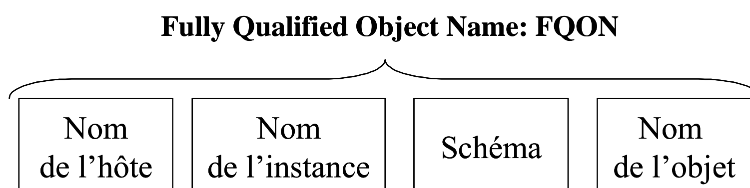
* *Transparent Network Substrate* (TNS) est une sous-couche d'Oracle Net qui reçoit les requêtes et émet des ouvertures ou fermetures de session, envoie les requêtes et reçoit des réponses.

* Lorsque les hôtes qui supportent les bases de données Oracle sont connectés via un réseau, les bases peuvent communiquer via *Net8 d'Oracle* (précédemment appelé *SQL*Net*). Les pilotes de Net8 s'appuient sur le protocole de communication local pour fournir la connectivité entre deux serveurs.

Afin que Net8 reçoive et traite les communications, l'hôte doit exécuter un processus appelé *listener* : module d'écoute sur un port de communication spécifique.

* Passerelle transparente (ang. *Oracle Transparent Gateway*) C'est la possibilité d'accéder à des objets non Oracle. La passerelle est exécutée sur l'hôte source qui contient la base à exploiter.

* Identification des Objets
Dans une BD centralisée, la combinaison du nom du propriétaire d'un objet et du nom de l'objet permet de l'identifier de façon unique. Dans les BDR deux couches d'identification sont ajoutées. Le quadruplet [hôte, instance, schéma, objet] forme un nom d'objet complet ou FQON.



Pour accéder à une table distante son identifiant FQON doit être connu. La transparence d'emplacement cache à l'utilisateur le triplet [hôte, instance, schéma].

11. Les liens de base de données

Pour interroger une BD distante, il faut créer un lien de base de données.

Un lien de base de données est un chemin unidirectionnel d'un serveur à un autre. En effet, un client connecté à une BD *A*, 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. La session demeure ouverte au cas où elle serait de nouveau nécessaire.

En créant un lien de BD, on doit indiquer le nom du compte auquel on se connecte, le mot de passe de ce compte, et le nom de service associé à la base distante. En l'absence d'un nom de compte, Oracle utilise le nom et le mot de passe du compte local pour la connexion à la base distante.

CREATE [SHARED|PUBLIC|PRIVATE] DATABASE LINK *NomLien*

CONNECT TO …..
CURRENT_USER
User IDENTIFIED BY *password*
USING connect_string

Un lien est soit privé ou public. Seul l'utilisateur qui a crée un lien privé peut l'utiliser, alors qu'un lien public est utilisé par tous les utilisateurs de la base de données. Le lien partagé est propre à la configuration de serveur multithreaded.

La clause CONNECT TO active une session vers la base distante.

La clause CURRENT_USER crée un lien BD pour l'utilisateur courant. L'utilisateur doit disposer d'un compte valide dans la base distante.

La clause USING *connect_string* spécifie le nom de service d'une base distante. Les noms de service d'instances sont stockés dans le fichier de configuration utilisé par Net8 intitulé *tnsnames.ora*. Ce fichier spécifie l'hôte, le port, et l'instance associés à chaque nom de service.

Des informations sur les liens de BD publics et privés, figurent respectivement dans les vues du dictionnaire de données : DBA_DB_LINKS et USER_DB_LINKS.

L'exemple suivant crée un lien public de bases de données, appelé RH_Lien :

```
CREATE PUBLIC DATABASE LINK RH_Lien
CONNECT TO RH IDENTIFIED BY PUFFINSTUFF
USING 'hq';
```

'hq' désigne le nom de service de la BD.

Utilisation du lien :

```
SELECT * FROM Employee@RH_Lien
WHERE office = 'ANNAPOLIS';
```

12. Transparence d'emplacement

Après avoir crée les liens de bases de données, plusieurs objets : les vues, les synonymes et les procédures stockées, peuvent servir à cacher la distribution des données aux utilisateurs :

12.1. Vues

Les vues peuvent fournir une transparence par rapport aux tables locales et distantes. Par exemple, supposons que la table *Employé* est sur une BD locale et la table *Département* est sur une BD distante. Pour rendre ces tables transparentes aux utilisateurs. Nous pouvons créer une vue dans la BD locale qui fait la jointure des données locales et distantes, comme ci-dessous : Les utilisateurs accédant à cette vue n'ont pas besoin de savoir où les données sont stockées.

Exercice 1:
Ecrire en SQL la commande de création de la vue *Entreprise*.

12.2. Synonymes

Les synonymes sont des noms simples qui permettent d'identifier de façon unique dans un système distribué les objets qu'ils nomment. Les synonymes peuvent être crées pour différents objets : Tables, Types, Views, Snapshots, Procedures, Functions, Packages. Ils figurent dans le dictionnaire de données.

```
CREATE [PUBLIC] nom-synonyme
FOR [schéma.]nom-objet[@nom-lien-BD] ;
```

Exercice 2:
Ecrire en SQL la requête suivante « On recherche les employés affectés au département génie logiciel » dans les deux cas suivants :

- La requête est émise du site A où se trouve la table *Employé*.
- La requête est émise du site B où se trouve la table *Département*.

12.3. Procédures

Les unités de programmes PL/SQL, peuvent servir à (*a*) référer à des données distantes, (*b*) appeler des procédures distantes, (*c*) utiliser des synonymes pour référer à des procédures distantes.

(*a*) Référer à des données distantes
Considérons la procédure *LicencierEmployé* :

```
CREATE PROCEDURE LicencierEmployé (enum NUMBER) AS
BEGIN
DELETE FROM Employé@hq.acme.com
WHERE NumEmp = enum
END;
```

Quand l'utilisateur ou l'application appelle la procédure *LicencierEmployé*, il/elle ne voit pas que la table distante est en cours de mise à jour.

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

```
CREATE SYNONYM Emp FOR Employé@hq.acme.com;
```

La procédure *LicencierEmployé* devient :

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

(*b*) Appeler des procédures distantes (RPC : ang. *Remote Procedure Call*)
On peut utiliser une procédure locale pour appeler une procédure distante. La procédure distante pourra exécuter les instructions LMD requises.

Exercice 3:
Ecrire le scénario suivant :
Un utilisateur *scott* identifié par *tiger* se connecte à la BD distante *hq.acme.com*. Il crée la procédure *FinEmployé*, ayant pour paramètre le numéro de l'employé, qui supprime l'employé de la table *Employé*.
L'utilisateur *scott* identifié par *tiger* se connecte à la BD locale *fin.acme.com*. Il crée la procédure *LicencierEmployé*, ayant pour paramètre le numéro de l'employé, qui appelle la procédure *FinEmployé*.

L'emplacement approprié pour une procédure dépend de la distribution et de l'utilisation des données. L'objectif étant de limiter le trafic sur le réseau pour résoudre les requêtes.

13. Mise au point des requêtes distribuées

Un serveur BD Oracle génère à partir d'une requête distribuée, des requêtes distantes, qu'il envoie aux nœuds distants pour exécution. Les nœuds exécutent alors les requêtes et retournent les résultats au serveur local. Un post-traitement est exécuté pour enfin retourner le résultat à l'utilisateur ou à l'application.

Les stratégies décrites dans ce qui suit, sont utilisées pour optimiser les requêtes.

13.1. Collocated Inline Views

Le moyen le plus efficace d'optimiser une requête consiste à réduire au maximum l'accès aux BDs distantes et de ne rapatrier que les données requises. Pour ce, Oracle utilise des «*Collocated Inline Views*», c'est à dire des vues de plusieurs tables distantes et en ligne, afin de forcer les restrictions sur les sites distants.

13.2. Optimisation basée sur le calcul des coûts

Oracle utilise une méthode basée sur le calcul des coûts (ang. *Cost based SQL optimizer : CBO*) pour trouver et générer la requête SQL qui extrait uniquement les données nécessaires des tables distantes.
Les données subissent un premier traitement sur le site distant, puis le site distant envoie le résultat au site local, d'où la requête a été émise.

Ré-écrire la requête suivante :
CREATE TABLE AS (

```
        SELECT  L.a, L.b, R1.c, R1.d, R1.e, R2.b, R2.c
FROM local L, Remote R1, Remote R2
WHERE (L.c = R1.c) AND (R1.c = R2.c) AND (R1.e > 300)
                );
```

La principale tâche exécutée par l'optimiseur consiste à ré-écrire une requête distribuée pour utiliser les *collocated inline views*. Par contre, si la requête distribuée contient des fonctions agrégats, des sous-requêtes, du complex-SQL, l'optimisation basée sur le coût ne peut être utilisée.

13.3. Statistiques

Oracle permet également de collecter des statistiques sur les différentes tables du système.

(*a*) Activation:
Il faut exécuter l'une des instructions suivantes:

```
ALTER SESSION OPTIMIZER_MODE = CHOOSE ;
ALTER SESSION OPTIMIZER_MODE = COST ;
```

(*b*) Analyse des tables :

```
ANALYZE TABLE Employé COMPUTE STATISTICS ;
ANALYZE TABLE Département COMPUTE STATISTICS ;
```

13.4. Hints

Les hints conviennent aux requêtes distribuées contenant des fonctions agrégats, des sous-requêtes, ou du complex-SQL.
Voir http://www.oradev.com/hints.jsp pour une liste de hints

→ DRIVING_SITE Hint

L'instruction DRIVING_SITE permet de préciser manuellement le site d'exécution de la requête.

Exemple :

```
SELECT /*+DRIVING_SITE(Département )*/  *
FROM Employé, Département@remote.com
WHERE Employé.NumDept = Département.NumDept ;
```

La jointure se fera sur le site où se trouve la table Département.

13.5. Analyse du plan d'exécution

Un des aspects les plus importants pour la mise au point de requêtes distribuées est l'analyse du plan d'exécution d'une requête distribuée.

Avant de pouvoir générer et d'afficher le plan d'exécution, il faut préparer une BD servant à sauver le plan d'exécution. Ceci est fait en exécutant le script suivant :
SQL>@utlxplan.sql
Ainsi, une table PLAN_TABLE est créée dans le schéma courant pour sauver temporairement le plan d'exécution.
La clause EXPLAIN PLAN FOR génère le plan d'exécution.

Exemple : Requête « *Noms des départements auxquels sont attachés plus que 3 employés* »

```
EXPLAIN PLAN FOR
        SELECT    D.NomDept
        FROM   Département D
        WHERE   D.NumDept   IN (SELECT  NumDept
                                        FROM  emp@orc2.world
                                        GROUP  BY  NumDept
                                        HAVING  COUNT(NumDept)  > 3);
```

Enfin, pour afficher le plan d'exécution sauvé dans PLAN_TABLE, il suffit d'exécuter le script suivant:
SQL>@utlxpls.sql
Pour voir l'expression SQL exécutée sur le site distant, on exécute :

```
SELECT other
FROM plan_table
WHERE operation = 'REMOTE';
```

On obtient par exemple:

```
SELECT DISTINCT ″A1″.″NumDept″
FROM  ″Employé″  ″A1″
GROUP BY ″A1″.″NumDept″
HAVING  COUNT(″A1″.″NumDept″)  >  3
```

14. Réplication des données

Afin de réduire la quantité de données transmises sur le réseau, et améliorer par conséquent les performances des requêtes, plusieurs options de réplication peuvent être envisagées.

14.1. 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 RH/PUFFINSTUFF@loc -
CREATE Employes -
USING –
SELECT * FROM Employes –
```

```
WHERE Etat = 'NM' ;
COMMIT ;
```

La base est identifiée par le service nommé LOC. durant la connexion, une session devrait être initiée par le compte RH avec le mot de passe PUFFINSTUFF.
L'inconvénient est que les données ne peuvent pas être mises à jour. La commande REPLACE est utilisée pour remplacer le contenu des tables.

14.2. Snapshots

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.

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

```
CREATE SNAPSHOT LocalEmp
TABLESPACE data2
STORAGE (INITIAL 100K next 100K PCTINCREASE 0)
REFRESH FAST
    START with SysDate
    NEXT SysDate+7
AS SELECT * FROM Employes@RH_Lien;
```

Un REFRESH FAST utilise un *snapshot log*, pour actualiser le *snapshot*. Ce fichier se trouve sur le même site que la table maître. Dans le *snapshot log*, sont stockées les modifications intervenues sur la table maître. Ainsi, pour chaque mise à jour, seules les modifications qui sont envoyées, et non l'ensemble des données. Par contre, un REFRESH COMPLETE est obligatoire pour les *snapshots* complexes.

Le *snapshot log* est à créer avant le *snapshot*:

```
CREATE SNAPSHOT LOG ON Employes
TABLESPACE  DATA
STORAGE (INITIAL 10K NEXT 10K PCTINCREASE 0)
PCTFREE 5 PCTUSED 90;
```

Une utilisation classique des *snapshots* en mise à jour est le cas du contrôle technique automobile. Tous les centres de contrôle stockent des données concernant les véhicules qu'ils ont contrôlés durant la journée. Chaque nuit, les données sont déversées dans la base nationale qui centralise les données de l'ensemble du parc automobile du pays. Notons que, les *snapshots* en mise à jour peuvent engendrer des conflits. Un déclencheur (ang. *trigger*) sauve les mises à jour opérées sur le *snapshot* et les transmet au site maître au moment du rafraîchissement du *snapshot*.

Les *snapshots* utilisent le package DBMS_JOB pour organiser les rafraîchissements, et nécessitent que le paramètre du fichier *init.ora* JOB_QUEUE_PROCESSES soit supérieur à 0.
Les vues ALL_SNAPSHOTS, ALL_SNAPSHOT_REFRESH_TIMES, ALL_SNAPSHOT_LOG détaillent les caractéristiques des snapshots crées.

14.3. Vues matérialisées

Une vue matérialisée peut apporter plusieurs avantages au niveau performances. Selon la complexité de la requête, on peut la remplir avec des changements incrémentiels, à l'aide du journal de vues matérialisées (MATERIALIZED VIEW LOG), au lieu de la recréer.
A l'inverse des *snapshots*, les vues matérialisées peuvent être utilisées directement par l'optimiseur, afin de modifier les chemins d'exécution des requêtes. Pour ce, il faut disposer du privilège QUERY REWRITE pour pouvoir réécrire la requête, et que QUERY_REWRITE_ENABLED soit TRUE (ALTER SESSION SET QUERY_REWRITE_ENABLED = TRUE).
Une vue matérialisée crée une table locale pour stocker les données, et une vue qui y accède.

Exemple:

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

La clause ENABLE QUERY REWRITE permet à l'optimiseur de rediriger les requêtes émises sur la table vers la vue matérialisée s'il le juge approprié.
La clause NEVER REFRESH empêche tout type d'actualisation de la vue matérialisée.

Une vue matérialisée ne peut pas contenir les op. UNION, MINUS, INTERSECT.
Les vues DBMS_MVIEW, ALL_MVIEW_ANALYSIS détaillent les caractéristiques des vues matérialisées créées.

15. Administration de grandes bases de données

La définition d'une grande BD change sans cesse. En 1995, c'est une BD de taille supérieure à 100Go, de nos jours elle fait plusieurs téraoctets.

15.1. Partitions

Au fur et à mesure que les tables augmentent en taille, leur maintenance devient complexe. Une table volumineuse est divisée en partitions selon les plages de valeurs de la colonne de partitionnement. Les index peuvent être partitionnés. Cette répartition a pour but d'améliorer les performances des opérations de maintenance, de sauvegarde et de récupération, ainsi que celles des transactions et des requêtes.

Oracle8*i* offre trois types de partitionnements: par plages, par hachage et composé.

15.1.1. Partitionnement par plages (ang. *Range partitioning*)

```
CREATE TABLE Employes (
    NumEmp     NUMBER(10) PRIMARY KEY,
    Nom        VARCHAR2(40),
    NumDept    NUMBER(2),
    Salaire          NUMBER(7,2),
    Date_naiss DATE,
    NumSecSoc VARCHAR2(9),
CONSTRAINT   FK_NumDept FOREIGN KEY (NumDept) REFRENCES
Département(NumDept) )
PARTITION BY RANGE (NumDept)
(PARTITION E1 VALUES LESS THAN (11)  TABLESPACE PART1_TS,
PARTITION E2 VALUES LESS THAN (21)  TABLESPACE PART2_TS,
PARTITION E3 VALUES LESS THAN (31)  TABLESPACE PART3_TS,
PARTITION E4 VALUES LESS THAN (MAXVALUE) TABLESPACE
PART4_TS ) ;
```

L'attribut de partitionnement est une un attribut NOT NULL.
Le partitionnement par plages peut se faire sur plusieurs colonnes de partitionnement.

Si on connaît le nom de la partition, il est possible d'invoquer le nom de la partition dans la clause FROM, comme suit,

```
SELECT *
FROM Employes PARTITION (E2)
WHERE NumDept BETWEEN 11 AND 20;
```

Par contre, si la plage des valeurs d'une partition change, le résultat ne répondra plus à la requête. Sachant qu'Oracle place des contraintes de contrôle CHECK sur chacune des partitions, il n'est pas recommandé d'utiliser cette syntaxe.

Oracle utilise les contraintes de contrôle des partitions, afin de déterminer la partition impliquée.

L'ajout d'une partition se fait comme suit,

```
ALTER TABLE  Employes
ADD PARTITION E5 VALUES LESS THAN (41) TABLESPACE PART5_TS;
```

La fusion de partition se fait comme suit,

```
ALTER TABLE  Employes
```

MERGE PARTITIONS E1, E2 INTO PARTITION E2 ;

15.1.2. Partitionnement par hachage (ang. *Hash partitioning*)

Une partition par hachage détermine l'emplacement physique des données en exécutant une fonction de hachage sur les valeurs de l'attribut de partitionnement. Dans l'exemple suivant, le nombre 10 désigne le nombre de partitions.

```
CREATE TABLE Employes (
    NumEmp     NUMBER(10) PRIMARY KEY,
    Nom        VARCHAR2(40),
    NumDept    NUMBER(2),
    Salaire          NUMBER(7,2),
    Date_naiss DATE,
    NumSecSoc VARCHAR2(9),
CONSTRAINT FK_NumDept FOREIGN KEY (NumDept) REFRENCES
Département(NumDept)        )
PARTITION BY HASH (NumDept)
PARTITIONS  10;
```

L'ajout d'une partition se fait comme suit,
ALTER TABLE Employes ADD PARTITION ;

L'opération inverse, supprime une partition
ALTER TABLE Employes COALESCE PARTITION ;

15.1.3. Partitonnement composé (ang. *Composite partitioning*)

Cette technique permet de combiner deux types de partitions. L'exemple suivant partitionne par plages la table *Employés* sur la colonne *NumDept*, et partitionne par hachage les partitions de cette colonne sur les valeurs de la colonne Nom.

```
CREATE TABLE Employés (
    NumEmp     NUMBER(10) PRIMARY KEY,
    Nom        VARCHAR2(40),
    NumDept    NUMBER(2),
    Salaire          NUMBER(7,2),
    Date_naiss DATE,
    NumSecSoc VARCHAR2(9),
CONSTRAINT   FK_NumDept FOREIGN KEY (NumDept) REFRENCES
Département(NumDept))
PARTITION BY RANGE (NumDept)
SUBPARTITION BY HASH (Nom)
SUBPARTITIONS 10
(PARTITION E1 VALUES LESS THAN (11)  TABLESPACE  E1_TS,
PARTITION E2 VALUES LESS THAN (21)   TABLESPACE  E2_TS,
PARTITION E3 VALUES LESS THAN (31)   TABLESPACE  E3_TS,
PARTITION E4 VALUES LESS THAN (MAXVALUE) TABLESPACE  E4_TS) ;
```

15.1.4. Partitionnement par liste de valeurs (ang. *List partitioning*) –Oracle9*i*

```
CREATE TABLE Departement (
	NumDept	NUMBER(2) PRIMARY KEY,
	NomDept	VARCHAR2(14),
	Ville		VARCHAR2(13))

PARTITION BY LIST (Ville)
	(PARTITION  dEst VALUES ('NEW YORK'),
	PARTITION  dOuest VALUES ('SAN FRANCISCO', 'LOS ANGELES'),
	PARTITION  dSud VALUES ('ATLANTA', 'DALLAS', 'HOUSTON'),
	PARTITION  dNord VALUES ('CHICAGO', 'DETROIT'));
```

Un tuple qui comporte une valeur autre que les spécifiées est rejeté.

15.1.5. Partitions d'index

L'index d'une table partitionnée peut être partitionné avec les mêmes plages de valeurs que celles de la table.

```
CREATE INDEX Employés_NumDept
ON Employes(NumDept)
LOCAL
(PARTITION IE1   TABLESPACE  IE1_NDX_TS,
PARTITION IE2   TABLESPACE  IE2_ NDX_TS,
PARTITION IE3   TABLESPACE  IE3_ NDX_TS,
PARTITION IE4  TABLESPACE P IE4_ NDX_TS) ;
```

Le mot clé LOCAL, crée un index séparé pour chaque partition de la table. Un index global contient des valeurs provenant de plusieurs partitions.

```
CREATE INDEX Employés_NumDept
ON Employès(NumDept)
GLOBAL
(PARTITION IE1 VALUES LESS THAN (11)  TABLESPACE IE1_NDX_TS,
PARTITION IE2 VALUES LESS THAN (21)  TABLESPACE IE2_ NDX_TS,
PARTITION IE3 VALUES LESS THAN (31) TABLESPACE IE3_ NDX_TS,
PARTITION IE4 VALUES LESS THAN (MAXVALUE) TABLESPACE IE4_ NDX_TS) ;
```

15.2. Gestion de Clusters

Les clusters fournissent une méthode de stockage des tables de données. Un cluster est formé de groupes de tables qui partagent les mêmes blocs de données. Les tables sont regroupées car elles ont des colonnes en commun, et sont très souvent interrogées ensemble. C'est le cas par exemple des tables *Employé* et *Département* qui ont en commun l'attribut *NumDépt*. L'attribut *NumDépt* est dit *clé du cluster*. Si on met en

cluster les deux tables, Oracle stocke –physiquement, pour chaque département, le département et ses employés dans les mêmes blocs de données.

Les clusters sont à éviter si les tables sont fréquemment accédées individuellement. Par contre, sont conseillés dans le cas où l'opération de jointure entre les deux tables, est fréquente.

```
CREATE CLUSTER EmpDept (NumDept NUMBER(3))
PCTUSED 80
PCTFREE 5
SIZE 600
TABLESPACE  users
STORAGE (INITIAL          200K    NEXT          300K
         MINEXTENTS   2         MAXEXTENTS  20
         PCTINCREASE  33 );

CREATE TABLE Departement (
    NumDept   NUMBER(3)   PRIMARY KEY, ….)
CLUSTER EmpDept(NumDept);

CREATE TABLE Employe (
    NumEmp   NUMBER(5)   PRIMARY KEY, ….,
       NumDept   NUMBER(3) REFRENCES Departement)
CLUSTER EmpDept(NumDept);
```

L'instruction suivante supprime un cluster:

```
DROP CLUSTER EmpDept ; //si le cluster ne contient aucune table
DROP CLUSTER EmpDept INCLUDING TABLES ; // si le cluster contient des tables
```

Oracle offre une deuxième alternative consistant en le stockage de tables dans des *hash cluster*.

```
CREATE CLUSTER EmpDept (NumDept NUMBER(3))
PCTUSED 80
PCTFREE 5
TABLESPACE  users
STORAGE (INITIAL      250K    NEXT          50K
           MINEXTENTS    1        MAXEXTENTS  3
       PCTINCREASE  0 )
HASH IS NumDept HASHKEYS 150;
```

Le nombre 150 indique le nombre de valeurs générées par la fonction de hachage.

16. Oracle Parallel Query

Cette option permet d'exécuter certains ordres SQL en parallèle. Le parallélisme des traitements est assuré par un processus coordinateur et plusieurs processus parallèles.

Les ordres DDL suivants peuvent être parallélisées, si l'objet source ou cible est fragmenté:

Create table . . . as select . . .
Create index

Rebuild d’un index ou d’une de ses partitions
Déplacement d’une partition
Eclatement d’une partition

Les instructions DML suivantes peuvent être parallélisées :

Delete from . . . appliqué sur des tables fragmentées.
Update . . . set . . . appliqué sur des tables fragmentées.
Insert . . . select . . . appliqué sur des tables fragmentées.

SQL*LOADER peut agir en parallèle sur des tables fragmentées.
En ce qui concerne les tables partitionnées, un degré supplémentaire de parallélisme peut être recherché dans le cas de l’utilisation des sous partitions.
Le traitement parallèle est activé par l’utilisation du hint PARALLEL dans l’ordre SQL:
SELECT **/*+parallel(matable,3) */** * FROM matable WHERE . . .

Le dictionnaire de données et PDML

Dans le dictionnaire de données, les vues V$PX_SESSION, V$PX_SESSTAT, V$PX_PROCESS, V$PX_PROCESS_SYSSTAT fournissent des statistiques concernant l’exécution des requêtes en parallèle:

Activation de PDML

ALTER SESSION [enable|disable|force] PARALLEL [DML/DDL] [parallel n] ;

Le degré de parallélisme des ordres DML est dicté d’une manière prioritaire par les hints. En ce qui concerne les ordres DDL, ils seront exécutés avec le degré de parallélisme par défaut. L’option force concerne les objets qui n’ont aucune clause de parallélisme dans leur clause de stockage ou ceux qui sont attaqués par des requêtes sans HINTS. Dans tous les cas, le degré de parallélisme doit se trouver dans l’intervalle [PARALLEL_MIN_SERVERS, PARALLEL_MAX_SERVERS].
Même avec PDML actif dans la session, aucune garantie de la réalisation de l’ordre en parallèle ne peut être assurée.

De nouveaux hints ont été crées pour être utilisés avec les INSERT : APPEND et NOAPPEND.

Exemple de requêtes parallélisées

```
SQL> commit;
Commit complete.
SQL> alter session enable PARALLEL dml;
Session altered.
SQL> alter session enable PARALLEL ddl;
Session altered.
SQL> select username, pdml_status, pddl_status, pq_status, pdml_enabled
  2  from v$session where username is not null;
USERNAME                       PDML_STA PDDL_STA PQ_STATU PDM
------------------------------ -------- -------- -------- ---
SYS                            ENABLED  ENABLED  ENABLED  YES
SQL> select * from v$pq_sesstat where statistic like '%Parallel%';
```

```
STATISTIC                     LAST_QUERY SESSION_TOTAL
------------------------------ ---------- -------------
Queries Parallelized                 0          9
DML Parallelized                     0          0
SQL> create table anp.opqb as
  2    select * from anp.opq;
Table created.
SQL> select * from v$pq_sesstat where statistic like '%Parallel%';
STATISTIC                     LAST_QUERY SESSION_TOTAL
------------------------------ ---------- -------------
Queries Parallelized                 1         10
DML Parallelized                     0          0
SQL> insert /*+parallel (anp.opqb,5) */ into anp.opqb
  2    select /*+parallel (anp.opq, 4) */ * from anp.opq;
2560 rows created.
SQL> select * from v$pq_sesstat where statistic like '%Parallel%';
STATISTIC                     LAST_QUERY SESSION_TOTAL
------------------------------ ---------- -------------
Queries Parallelized                 1         11
DML Parallelized                     0          0
SQL> commit;
Commit complete.
SQL> UPDATE /*+PARALLEL (anp.opqb,2) */
  2    anp.opqb SET c1 = c1*10;
5120 rows updated.
SQL> select * from v$pq_sesstat where statistic like '%Parallel%';
STATISTIC                     LAST_QUERY SESSION_TOTAL
------------------------------ ---------- -------------
Queries Parallelized                 0         11
DML Parallelized                     0          0
SQL> UPDATE /*+PARALLEL (anp.opqb,2) */
  2    anp.opqb SET c1 = c1*10;
5120 rows updated.
SQL> DELETE /*+PARALLEL (anp.opqb,4) */
  2  FROM anp.opqb
  3  WHERE to_number(substr(c2,1,1)) < 5;
4096 rows deleted.
SQL> select * from v$pq_sesstat where statistic like '%Parallel%';
STATISTIC                     LAST_QUERY SESSION_TOTAL
------------------------------ ---------- -------------
Queries Parallelized                 0         11
DML Parallelized                     0          0
SQL> spool off;
```

L'exemple précédent montre l'utilisation des processus en parallèle pour toutes les opérations de création et d'interrogation de tables. La parallélisation n'a pas été effectuée pour les *delete* ou *update*, car ce n'est possible que pour les tables partitionnées.

Bibliographie

[1] S. Spaccapietra, C. Vingenot, *Bases de données réparties*, http://lbdwww.epfl.ch/f/teaching/courses/slidesBDA/BDR/BDR_se.pdf

[2] M. T. Özsu & P. Valduriez, *Principles of Distributed Database Systems*, Prentice Hall 1999.

[3] D. Kossmann, *The State of the Art in Distributed Query Processing*, http://www.db.fmi.uni-passau.de/~kossmann

[4] D. Donsez, *Les SGBDs Parallèles*, http://www.adele.imag.fr/~donsez/cours/

[5] D. Donsez, *Répartition, Réplication, Nomadisme, Hétérogénéité dans les SGBDs*, http://www.adele.imag.fr/~donsez/cours/

[6] *Chapitre 8: La gestion des transactions*, http://medias.obs-mip.fr/cours/concept/chap8.htm

[7] J. Durbin, L. Ashdown, *Oracle8i : Distributed Database Systems*, Oracle Press, 1999.

http://sunsite.eunnet.net/documentation/oracle.8.0.4/server.804/a58247.pdf

[8] K. Loney, M. Theriault, *Oracle8i DBA Handbook*, Oracle Press 2000.

[9] Oracle8 Product Documentation Library, http://www-rohan.sdsu.edu/doc/oracle/

[10] I. Ben-Gan, T. Moreau, *Advanced Transact-SQL for SQL Server 2000*, Chapitre 13 p.459: *Horizontally Partitionned Views*, *a*!Press.

[11] B. Ducourthial, Les bases de données réparties,

http://wwwhds.utc.fr/~ducourth/TX/BDD/

[12] M. Gertz, ORACLE/SQL Tutorial, http://www.db.cs.ucdavis.edu.

[13] Oracle Parallel Query, http://www.tafora.fr/wp/pqo.doc.html.

Note : les exemples et exercices du cours sont extraits principalement des références suivantes [1][2][8][9][12][13]