Université Pierre et Marie Curie - Master d'Informatique

Saison février - juin 2016

UE 4I803 : Bases de Données Réparties

TD - partie 2

Stéphane GANÇARSKI, Hubert NAACKE

URL : www-bd.lip6.fr/wiki/doku.php/site/enseignement/master/bdr/start

TABLE DES MATIERES

TD - Conception de BD réparties - Requêtes

Le but de ce TD est l'étude de la conception d'une base de données répartie. Dans la première partie, une base dont le schéma conceptuel a été défini est distribuée sur plusieurs sites. Le but principal de la distribution est de maximiser les accès locaux par rapport aux accès répartis. Dans la seconde partie, une base préexistante est intégrée au système. Dans ce cas, l'intégration doit être réalisée sans modification de la base locale qui possède ses propres applications locales déjà écrites, sous la forme d'une vue répartie. La troisième partie est une étude quantitative du coût de traitements des requêtes sur une BD répartie. Enfin, la dernière partie est une étude de cas.

1 - Conception de BD réparties

Schéma global de la base

La base de données hospitalière de la région Alsace a le schéma suivant :

Service (Snum, nom, hôpital, bât, directeur)

le directeur d'un service est un docteur désigné par son numéro

Salle (Snum, SAnum, surveillant, nbLits)

le numéro de salle est local à un service, i.e., il peut y avoir des salles avec le même numéro dans des services différents d'un même hôpital.

nbLits est le nombre total de lits d'une salle,

un surveillant de salle est un infirmier désigné par son numéro

Employé (Enum, nom, adr, tél)

Docteur (Dnum, spéc) -- *spéc est la spécialité du médecin*

Infirmier (Inum, Snum, rotation, salaire)

Un employé est soit infirmier soit docteur (Inum et Dnum font référence à Enum).

Patient (Pnum, Snum, SAnum, lit, nom, adr, tél, mutuelle, pc)

L'attribut pc est la prise en charge par la mutuelle

Acte (Dnum, Pnum, date, description, coef) -- *coef est le coefficient de l'acte médical*

Question 1

Exprimer en SQL la question suivante: "Donner le nom des cardiologues qui ont traité un ou plusieurs patients hospitalisés dans un service de gérontologie."

Répartition des données

La base est répartie sur trois sites informatiques, "Strasbourg", "Colmar" et "Régional", correspondant aux valeurs "*Ambroise Paré*", "*Colmar*" et "*autre"* de l'attribut *hôpital* de Service.

Question 2

Proposer (et justifier) une bonne décomposition de la base hospitalière sur ces trois sites. On pourra utiliser la fragmentation horizontale et/ou verticale ainsi que la réplication des données, en se basant sur les hypothèses suivantes (H1 à H5) :

• H1: Les sites Strasbourg et Colmar ne gèrent que les hôpitaux correspondants.

• H2 : Les infirmiers sont employés dans un service donné.

• H3 : Les docteurs travaillent le plus souvent sur plusieurs hôpitaux (ou cliniques).

• H4 : La gestion des lits d'hôpitaux est locale à chaque hôpital.

• H5 : On désire regrouper la gestion des frais d'hospitalisation au centre régional.

Pour chaque fragment, on donnera sa définition en algèbre relationnelle à partir du schéma global.

Question 3

Indiquer comment se calcule chaque relation de la base globale à partir de ses fragments.

Question 4

Proposer un plan d'exécution réparti pour la requête SQL vue en Question 1, sachant maintenant que les données sont réparties sur les trois sites selon la décomposition proposée à la Question 2.

2 - Conception de BD fédérées par intégration

On suppose que l'hôpital de Belfort est rattaché à la base de donnée hospitalière de la région Alsace après son implémentation répartie. L'hôpital de Belfort possède donc son propre site de traitement qui doit être connecté aux autres sites.

Le schéma de la base à Belfort avant l'intégration est le suivant:

B_Service (Snum, Nom, Bâtiment, Directeur)

B_Salle (Snum, SAnum, Surveillant, NbLits)

B_Docteur (Dnum, Nom, Adresse, Téléphone, spécialité)

B_Infirmier (Inum, Nom, Adresse, Téléphone, Snum, Salaire)

B_Patient (Pnum, Snum, SAnum, Lit, Nom, Adresse, Téléphone, Mutuelle, PriseEnCharge)

B_Acte (Dnum, Pnum, Date, Description, Code)

Question 5

Discuter les problèmes et proposer des solutions pour l'intégration de la base Belfort au système réparti déjà défini. L'intégration devra se faire sans transfert d'information et sans modification de des bases existantes, mais uniquement par définition de vues.

Question 6

Définir le nouveau schéma global intégrant la base Belfort. Chaque relation du schéma global (Service2, Salle2, …Acte2) est définie en fonction des fragments sur les 4 sites.

Question 7

L'hypothèse H5 est-t-elle toujours respectée après l'intégration de la base Belfort ? Si non, quelles sont les modifications de schéma nécessaires pour respecter H5 ?

Question 8

Proposer une décomposition et un plan d'exécution pour la question SQL précédente après l'intégration de la base "Belfort".

3 - Evaluation de requêtes réparties (juin 2002)

On considère la base de données répartie de schéma suivant **:**

Employés (#emp, #serv, salaire)

Service (#serv, #dir, budget)

L'attribut #dir représente l'identificateur de l'employé dirigeant le service.

La relation Employés est stockée à Naples, la relation Service est stockée à Berlin.

La taille des n-uplets de ces deux relations est de 20 octets. La taille de **#emp** et de **#dir** est de 10 octets. Les attributs **salaire** et **budget** contiennent des valeurs uniformément réparties dans l'intervalle [0, 1 000 000]. La relation **Employés** comprend 100 000 pages, la relation **Service** comprend 5000 pages. Chaque processeur a **400** pages de buffer. La taille d'une page (du buffer et du disque) est de 4000 octets. Le coût d'entrée/sortie d'une page est t_{IO}, le coût de transfert d'une page d'un site à un autre est t_T. L'unité de transfert est la page. On suppose qu'il n'y a pas d'index.

On considère la requête suivante :

```
SELECT *
FROM Employés E, Service S
WHERE E.#emp = S.#dir
```

La requête est envoyée de Londres, et on sait que 1% des employés sont des directeurs.

Question 1.

a) Calculer la taille d'un tuple du résultat.

b) Calculer la cardinalité du résultat.

c) Calculer la taille (en nombre de pages) du résultat de cette requête.

Question 2.

On suppose que toutes les jointures sont faites en utilisant l'algorithme de tri-fusion, dont le coût dépend du nombre de pages (noté *p()*) des relations participant à la jointure. On a :

$Coût(R \bowtie_a S) = Tri_a(R) + Tri_a(S) + [Lect(R) + Lect(S)]$

Tri(R) = Lect(R) + Ecr(R) si les données ne sont pas déjà triées sur les attributs de jointure
Lect(R) = p(R) si les données sont locales
= 0 si les données sont transmises directement depuis un site distant
Ecr(R) = p(R) // correspond au stockage temporaire des données

Par exemple, pour une jointure entre 2 relations locales non triées on a :

$Coût(R \bowtie_a S) = 3 * (p(R) + p(S)) * t_{IO}$

Pour une jointure entre 2 relations locales triées, on a :

$Coût(R \bowtie_a S) = (p(R) + p(S)) * t_{IO}$

Si R est déjà trié et est transmise depuis un site distant et si S est locale non triée, on a

$Coût(R \bowtie_a S) = (3*p(S)) * t_{IO}$

On demande de calculer le coût de l'exécution de cette requête pour les différents plans d'exécution suivants :

Plan P1: On calcule la requête à Naples en envoyant la relation Service à Naples ; le résultat est envoyé à Londres.

Plan P2: On calcule la requête à Berlin en envoyant la relation Employés à Berlin ; le résultat est envoyé à Londres

Plan P3: On calcule la requête à Londres, en envoyant les deux relations à Londres.

Plan P4: On calcule la requête à Berlin puis à Naples, en utilisant une semi-jointure à Berlin ; on envoie le résultat final à Londres. (attention, la semi-jointure peut amener à créer des relations temporaires qu'il faudra intégrer dans le coût)

Plan P5: On calcule la requête à Naples puis à Berlin en utilisant une semi-jointure à Naples ; on envoie le résultat à Londres.

Question 3.

D'après vos réponses, quel est le plan qui minimise les transferts ? Est-ce forcément le plan le plus intéressant ? Quel est le meilleur plan ?

4 - Fragmentation et requête répartie (bdweb janvier 2002)

La base de données d'un revendeur en ligne, PointCom, a le schéma global suivant :

PRODUITS (Num-Produit, Catégorie, Titre, Description, Prix, …)

INVENTAIRE (Num-Produit, Entrepôt, Nbr-Exemplaire, …)

COMMANDES (Num-Commande, Num-Produit, Adresse-Livraison, Quantité, …)

PointCom offre quatre catégories de produits : livres, musique, vidéos, et jeux, et possède trois entrepôts situés en Californie (CA), New York (NY) et Colorado (CO).

- L'entrepôt de la Californie gère des livres, de la musique, et des vidéos, et effectue des livraisons pour des commandes vers les régions de l'ouest et du centre des États-Unis.
- L'entrepôt de New York gère des livres, des jeux, et des vidéos, et effectue des livraisons pour des commandes vers les régions de l'est et du centre des États-Unis.
- L'entrepôt de Colorado gère des livres, des jeux, et de la musique, et effectue des livraisons pour des commandes vers toutes les régions des États-Unis.
- Num-Produit est la clé primaire de PRODUITS.
- Catégorie prend une des valeurs suivantes : "livre", "musique", "vidéo", et "jeux".
- Entrepôt prend une des valeurs suivantes : "CA", "NY", et "CO".
- Adresse-Livraison prend une des valeurs suivantes : "Est", "Ouest", et "Centre".

Question 1

Exprimer en SQL sur le schéma global la requête permettant de retrouver le nombre d'exemplaires disponibles du livre intitulé " X " pour une livraison vers l'ouest des États-Unis.

Question 2

Supposons maintenant que la base PointCom est répartie sur les trois sites informatiques de la Californie, de New York, et du Colorado. Proposer une bonne décomposition de la base sur ces trois sites. Donner, en algèbre relationnel, la définition des différents fragments.

Question 3

Proposer un plan d'exécution répartie pour la requête de la question 3 minimisant les transferts entre les sites (vous décidez du site qui pose initialement la requête).

TD : Conception des BD Réparties

Exercice 1 : Fragmentation (juin 2004) pts

Le schéma global d'une base de données cinématographique est le suivant :

Film (F, titre, année, durée, réalisateur).	un film est identifié par son numéro *F*
Artiste (A, nom, prénom, nationalité)	un artiste est identifié par son numéro *A*
Rôle (nom, F, A)	l'artiste numéro *A* joue le rôle *Nom* dans le film numéro *F*.
Cinéma (C, arrond, ville)	un cinéma est identifié par son nom *C*.
Salle (C, Sa, P, clim)	la salle numéro *Sa* du cinéma numéro *C* a une capacité de *P* places. L'attribut *clim*, valant *'oui'* ou *'non'*, indique si la salle est climatisée.
Séance (C, Sa, H, F)	le film numéro *F* est projeté dans la salle numéro *Sa* du cinéma *C* à partir de l'heure *H*.

Les attributs soulignés forment la clé d'une relation.

Question 1: Un spectateur Si possède une partie de la base sur son site personnel. Un spectateur *cinéphile* ne s'intéresse qu'aux films anciens réalisés jusqu'à 2000 inclus, un spectateur *tendance* ne s'intéresse qu'aux films récents réalisés en 2001 et après. Un spectateur ne s'intéresse qu'à des films projetés dans sa *ville* : Paris, Lyon ou Nice. Un spectateur s'intéresse à toutes les données liées aux films qui l'intéressent. Par contre, un spectateur ne s'intéresse jamais à la capacité d'une salle.

Remarque : On peut déduire de l'énoncé qu'un spectateur *cinéphile* ne s'intéresse pas aux films récents dans lesquels jouent des acteurs qui jouent aussi dans des films anciens.

Définir une fragmentation de la base cinématographique qui permette à un spectateur de construire sa base personnelle en répliquant des fragments sur son site, en respectant les conditions suivantes :

- un fragment est répliqué soit entièrement, soit pas du tout, sur le site d'un spectateur,

- la base d'un spectateur doit contenir toutes les données qui l'intéressent, et seulement celles-ci.

Pour **chaque relation** du schéma global :

a) Les n fragments d'une relation R sont notés R_i avec $i \in [1, n]$. Donner le nombre de fragments et écrire les expressions algébriques de définition des fragments.

b) La fragmentation proposée est-elle complète ? Est elle disjointe ? Si non expliquez pourquoi.

c) Définir le schéma de reconstruction de la relation à partir des ses fragments. Donner son expression algébrique

Exercice 2 : Optimisation de requêtes réparties (juin 2004) pts

Soit le schéma global

Artiste (A, nom, prénom, nation) un artiste est identifié par son numéro *A*

Rôle (nom_rôle, F, A) l'artiste numéro *A* joue le rôle *nom_rôle* dans le film numéro *F*.

Les données sont fragmentées de la manière suivante :

Artiste$_i$ = σ p_i (Artiste) où p_i est de la forme *nation = 'n$_i$'* avec $n_i \in$ {D, F, US}

Rôle$_i$ = (Rôle $\ltimes_A$ Artiste$_i$)

Les données sont réparties sur 4 sites dans 3 pays. Chaque pays contient les données en lien avec les artistes citoyens du pays :

Berlin (**B**) contient Artiste$_1$ et Rôle$_1$.

Paris (**P**) contient Artiste$_2$ et Rôle$_2$.

New York (**NY**) et Los Angeles (**LA**) contiennent les mêmes données : Artiste$_3$ et Rôle$_3$.

Les valeurs de tous les attributs, **sauf nationalité**, sont uniformément distribuées. Tous les attributs sont indépendants. Il y a 20 rôles par acteur, 10 rôles par film
Il y a 100 artistes allemands, 100 artistes français et 1000 artistes américains. On considère que tous les artistes sont acteurs. Un acteur n'a jamais 2 rôles dans le même film.

Il n'y a pas d'homonyme : card(π_{nom} Artiste) = card(Artiste)

Le coût d'un traitement local sur un site est négligeable par rapport au coût d'un transfert intersite.
Le coût de transfert dépend de la distance entre les sites. La fonction *tr(T, X, Y)* donne le coût pour transférer le résultat de l'expression algébrique *T* depuis le site *X* vers le site *Y*. On a:

tr(T, NY, P) = card(T)
tr(T, LA, NY) = 2* card(T)
tr(T, LA , P) = 3* card(T)
tr(T, X, Y) = tr(T, Y, X)

Le coût d'un plan d'exécution est la somme de tous les transferts nécessaires pour exécuter le plan.

Question 1.

a) Quelles sont les cardinalités de Artiste, Rôle, π_F(Rôle) $\sigma_{nation = 'US'}$ Artiste ?

b) Que vaut le facteur de sélectivité SF($\sigma_{nation = 'US'}$ Artiste) ?

Question 2.

On souhaite traiter les requêtes suivantes. Pour chaque requête, donner le plan d'exécution de **coût minimal**, en précisant l'expression algébrique de chaque sous requête T_i locale et les transferts. Utiliser la notation *tr(T_i, X, Y)* pour désigner les transferts. Le site sur lequel la requête est posée, reçoit le résultat.

a) Requête **R1** posée à Los Angeles (LA) : Donner le nom des acteurs français qui ont joué avec l'acteur américain nommé Pitt (*i.e.*, dans le même film que lui).

select a2.nom
from Artiste a1, Artiste a2, Role r1, Role r2
where a1.nom = 'Pitt' **and** a1.nationalité= 'US' **and** a2.nationalité = 'F'
and a1.a = r1.a **and** r1.f = r2.f **and** r2.a = a2.a

```
Etapes du plan de coût minimal :
1) sur le site ...
       traiter T1 = ...
       tr(T1, ... , ...) =

2) sur le site ...
       traiter T2 = ...
       tr(T2, ... , ...) =

etc...                  →                                    coût total = ..
```

b) La requête **R2** est posée à Paris (P) : Quels sont les films dont le casting est franco-américain ?

Plus précisément : Donner le numéro des films dans lesquels jouent au moins un acteur français et au moins un acteur américain. Quel est le plan optimal pour R2 ?

c) On suppose maintenant que les attributs *nation* et *F* ne sont plus indépendants. Pour cela on donne

card (π_F Rôle$_1$) = card (π_F Rôle$_2$) = 1000 et card (π_F Rôle$_3$) = 1200

et 5% des films où figurent au moins un acteur français a également un acteur américain

Quel est le plan optimal pour la requête R2 de la question précédente ?

Exercice 3 : Requêtes réparties (juin 2004) 3 pts

On considère la relation

Employé (E, nom, D, salaire)

Un n-uplet représente un employé identifié par son numéro *E*. Le numéro *D* fait référence au directeur de l'employé. Le directeur est aussi un employé.

Soit la requête R1 :

```
select a.nom
from Employé a, Employé b
where a.D = b.E
and a.salaire > b.salaire
```

Question 1. Traduire la requête R1 en une phrase, en français :

Question 2. Donner l'expression algébrique de la requête R1 en fonction de la relation globale *Employé*, et telle que les opérations les plus réductrices sont traitées le plus tôt possible.

Question 3. La relation *Employé* est fragmentée en 2 fragments *E1* et *E2* tels que:

E1 contient tous les employés dont le salaire est inférieur ou égal à 1000,

E2 contient tous les employés dont le salaire est supérieur à 1000.

Donner l'expression algébrique de la requête *R1*, en fonction des fragments *E1* et *E2*, et telle que l'expression soit de la forme :

$$R1 \Leftrightarrow \pi_{\text{nom}} (\text{T1} \cup \text{T2} \cup \ldots \cup \text{Tn})$$

où les sous expressions *Ti* ne contiennent pas d'union et peuvent ne pas être vides,

les opérations les plus réductrices sont traitées le plus tôt possible.

Préciser combien il y a de sous expressions Ti.

BD Réparties : optimisation de requêtes 6 pts

On considère le schéma relationnel suivant :

```
EMPLOYES (#emp, #service, salaire)
SERVICES (#service, #chef, budget)
```

Les valeurs de l'attribut `#chef` sont des valeurs de `#emp`. La relation `EMPLOYES` contient 100000 pages, la relation `SERVICES` contient 5000 pages.Les n-uplets des deux relations ont 20 octets. Les valeurs des attributs salaire et budget contiennent des valeurs uniformément réparties entre 0 et 1000000. La taille des pages est de 4000 octets.

Les relations sont réparties sur 10 sites, de la façon suivante : la relation SERVICES est partitionnée horizontalement sur les 10 sites par #service, avec le même nombre de n-uplets sur chaque site. La relation EMPLOYES est partitionnée horizontalement en fonction du salaire. Les n-uplets dont le salaire est <= 100000 sont stockés sur le site 1, ceux dont le salaire est compris entre 100000 et 200000 sont sur le site 2, etc. La partition des n-uplets dont le salaire est inférieur à 100000 est fréquemment lue, et très peu souvent mise à jour. Elle est donc répliquée sur tous les sites. Aucune autre partition de la relation EMPLOYE n'est dupliquée.

Décrivez le plan d'exécution des requêtes suivantes, et donnez leur coût, sachant que le coût de transfert d'une page est Ct, et le coût d'un accès disque est Cd.

Question 1. Calculer la jointure naturelle entre EMPLOYES et SERVICES, en utilisant la stratégie qui consiste à envoyer tous les fragments de la plus petite relation vers les sites contenant les n-uplets de la plus grande relation. L'algorithme de jointure utilisé est le trifusion, dont le coût est de 3(M+N)Cd, M et N étant les tailles des relations (N et M sont exprimées ici en nombre de **nuplets**) participant à la jointure.

Question 2. Quel est l'employé le mieux payé ? (en cas de dupliqués, la requête doit renvoyer tous les n-uplets répondant au critère).

Question 3.Quel est le chef le mieux payé ?

BDR - Examen juin 2006

Exercice 1 : Evaluation de requêtes (BD parallèles et BD réparties) 5 pts

On considère un SGBD parallèle dans lequel chaque relation est stockée par partitionnement horizontal des n-uplets sur tous les disques. On a la relation suivante :

Joueurs (*joueur-id* : integer**,** *eq-id* : integer**,** *salaire* : real**)**

On suppose que l'attribut salaire contient des valeurs dans l'intervalle [0, 1 500 000] uniformément distribuées. La relation a 150 000 pages. Une page a une taille de 4K octets, et comprend 200 n-uplets (un n-uplet a une taille de 20 octets). Le coût de lecture d'une page du disque (ou d'écriture sur le disque) est t_d, et le coût d'envoi d'une page d'un processeur vers un autre est t_s. On suppose que la relation a été partitionnée suivant l'algorithme round-robin, et qu'il y a 15 processeurs.

Question 1. Décrivez brièvement le plan d'évaluation de la requête R1, et calculez son coût en termes de temps d'accès (t_d) et temps d'envoi (t_s). On donnera le coût total de la requête, ainsi que le coût en temps écoulé (si plusieurs opérations sont effectuées en parallèle, le temps écoulé est le maximum du temps pris par chacun des processeurs pour faire son travail).

R1 . Quel est le joueur le mieux payé ?

Question 2. La relation Joueurs est maintenant stockée dans un SGBD réparti comprenant 15 sites. Elle est partitionnée horizontalement sur les 15 sites, selon les valeurs de l'attribut *salaire,* en stockant les n-uplets vérifiant la condition *salaire* <=100000 sur le premier site, les n-uplets vérifiant la condition 100000 <*salaire*<=200000 sur le second site, et ainsi de suite. La base est complétée par la relation **Equipes** suivante :

Equipes (*eq-id* : integer, *cap-id* : integer, *pays*: string)

Le champ *cap-id* de la relation Equipes est le *joueur-id* du capitaine de l'équipe. La relation Equipes comprend 4500 pages, et est répartie horizontalement selon l'attribut eq-id. On suppose que la répartition est uniforme (on a le même nombre de n-uplets sur chaque site), et aléatoire (il n'y a pas de critère pour répartir les n-uplets sur les sites). Les n-uplets de la relation Equipes ont une taille de 20 octets.

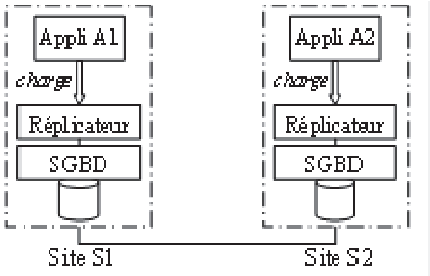
Décrivez brièvement les plans d'exécution des requêtes R1 et R2, et donnez leur coût en fonction de t_d et t_s.

R1. Quel est le joueur le mieux payé ?

R2. Faire la jointure naturelle des relations Joueurs et Equipes en utilisant la stratégie qui consiste à envoyer tous les fragments de la plus petite relation sur chaque site contenant les n-uplets de la plus grande relation.

Exercice 2 : Allocation de fragments 5 pts

On considère l'architecture répartie suivante :



Chaque site héberge un SGBD et une application. Une application produit une charge constituée de lectures et d'écritures. Pour maintenir les sites à jour, on ajoute un module de gestion des répliques (intergiciel appelé réplicateur). Si une donnée est répliquée sur plusieurs sites, une écriture doit être traitée sur **toutes** les répliques. Une lecture est traitée localement si possible.

Soit un fragment F et deux sites S1, S2. Le coût pour lire (resp. écrire) un nuplet vaut 1 (resp. 2). Le coût pour transférer un nuplet vaut 3. Ainsi, on a le modèle de coût suivant :

Lecture locale : LL = 1
Ecriture locale : EL = 2
Lecture distante, depuis Si, d'un nuplet de Sj (avec i≠j) : LD = 4
Ecriture distante, depuis Si, d'un nuplet de Sj (avec i≠j) : ED = 5

On souhaite allouer F sur un ou plusieurs sites de manière à minimiser le coût de traitement des applications.

Question 1
Une application A_i produit, sur le site S_i , une charge valant *6*n* lectures et *n* écritures de F (toutes les applications produisent la même charge). On veut calculer, en fonction de *n*, la quantité de lectures et d'écritures traitées par les SGBD des sites S1 et S2.
1.1) On suppose que F est seulement sur S1. Quel est le coût des traitements sur chaque site ?
1.2) On suppose maintenant que F est répliqué sur S1 et S2. Quel est le coût des traitements sur chaque site ?

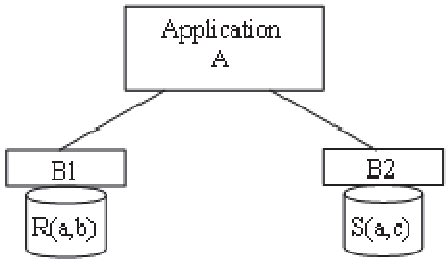
Question 2
On complète l'architecture initiale avec un troisième site S3 et son application A3. Les applications produisent la charge suivante : A2 produit 3 fois plus de traitements que A1, A3 produit 4 fois plus de traitements que A1. Chaque application produit 10 fois plus de lectures que d'écritures. A1 produit *n* écritures et *10*n* lectures.

On veut savoir s'il est intéressant d'allouer F sur S3 ; plus précisément, dans le cas où F ne serait pas alloué sur S3, quelle serait l'augmentation (ou la diminution) du coût que cela induirait pour les traitements de A3.

2.1.a) Quelle est, en fonction des 3 variables n, L (lecture) et E (écritures) la charge produite par chaque application.
2.1.b) Est-il nécessaire d'allouer F sur S3 pour minimiser globalement le coût des traitements ? Justifier brièvement.
2.1.c) Le fait d'allouer F sur S3 a-t-il un impact sur le coût de traitement des lectures de A3 ? Si oui, quelle est la différence de coût pour traiter les lectures de A3 en comparaison avec une configuration où F n'est pas sur S3 ? Donner une réponse en fonction de n.
2.1.d) Quelle est l'allocation pour laquelle F n'est pas sur S3 et les écritures sont minimisées ? Dans ce cas, quel est en fonction de n, le coût de traitement des écritures produites par A3 ?
2.1.e) Quelle est l'allocation pour laquelle F est sur S3 et les écritures sont maximisées ? Dans ce cas, quel est en fonction de n, le coût de traitement des écritures produites par A3 ?
2.1.f) Finalement, quelle est, en fonction de n, la plus petite augmentation (ou la plus forte diminution) de coût apportée par une configuration où F n'est pas sur S3.
2.2) Quel est le coût des traitements si F est seulement sur S3 ?
2.3) Quel est le coût des traitements si F est répliquée sur S2 et S3 ?
2.4) Quel est le coût des traitements si F est répliquée sur S1, S2 et S3 ?
2.5) Quelle allocation minimise globalement le coût des traitements ?

Exercice 3 : JDBC — 5 pts

Les relations **R**(<u>a, b</u>) et **S**(<u>a, c</u>) sont réparties selon l'architecture suivante :



L'application A, écrite en java, communique avec les SGBD B1 et B2 par JDBC. B1 et B2 ne peuvent pas communiquer directement entre eux. Soient les informations suivantes :

card(R) = 1000, card(S)=300, la taille des attributs en octets est a=4, b=10, c=10.
Les valeurs de a, b, c sont des entiers positifs. Il y a exactement 2 nuplets de R par valeur de *a* et 3 nuplets de S par valeur de *a*. R et S ont 4 valeurs distinctes de *a* en commun.

Soient CE le coût unitaire d'exécution d'une requête sur un site et CT le coût unitaire de transfert d'une donnée sur le réseau.

Le coût d'instruction JDBC est le suivant : (toutes les tailles sont en octets)

Interface	**Méthode**	**Coût (en octets)**
Statement	executeQuery(requête)	CE * taille du résultat de la requête
ResultSet	getString(...)	**CT** * taille de l'attribut lu

Toutes les autres instructions ont un coût nul.
Remarques : Le coût correspond au traitement d'une seule occurrence d'une instruction. Le coût total de plusieurs occurrences d'une même instruction, pendant tout le déroulement du programme, est déduit du tableau ci-dessus.

Question 1

Donner la valeur de card($R \bowtie_a S$)

Question 2

L'application exécute le programme P1 ci-dessous :

```
1       Connection c1 = DriverManager.getConnection("B1");
2       Connection c2 = DriverManager.getConnection("B2");
3       Statement s1 =  c1.createStatement();
4       Statement s2 =  c2.createStatement();
5       ResultSet r1 =  s1.executeQuery("select a,b from R");
6       while (r1.next()) {
7         String v = r1.getString(1);
8         ResultSet r2 = s2.executeQuery("select c from S where a = '" + v + "'");
9         boolean p = true;
10        String w = "";
11        while (r2.next()) {
12          if(p) {
13              w = r1.getString(2);
14              p = false;
15          }
16          out.println("[" v + ", " + w + ", " + r2.getString(1) + "]");
17        }
18        r2.close();
19      }
20      r1.close();
21      s1.close(); s2.close(); c1.close();  c2.close();
```

a) Pourquoi le programme contient-il l'instruction conditionnelle *if(p)* en ligne 12 ?
b) Combien de fois, pendant toute la durée d'exécution du programme, l'instruction de la ligne 13 est-elle traitée ?
c) Combien de fois, pendant toute la durée d'exécution du programme, l'instruction de la ligne 8 renvoie-t-elle un résultat vide (dont la taille est nulle) ?
d) Quel est le coût de P1? Pour chaque instruction dont le coût n'est pas nul : donner le n° de ligne, le nombre de fois que l'instruction est exécutée et son coût en fonction de CE et CT. Remarque : le coût d'une ligne correspond au coût **total** de traitement de cette ligne pendant toute l'exécution du programme.

Question 3

L'application exécute le programme P2 ci-dessous:

```
1     Connection c1 = DriverManager.getConnection("B1");
2     Connection c2 = DriverManager.getConnection("B2");
3     Connection c3 = DriverManager.getConnection("B1");
4     Statement s1 = c1.createStatement();
5     Statement s2 = c2.createStatement();
6     Statement s3 = c3.createStatement();
7     ResultSet r1 = s1.executeQuery("select distinct a from R");
8     while(r1.next()) {
9       String z = r1.getString(1);
10      ResultSet r2 = s2.executeQuery("select c from S where a = '" + z + "'");
11      while(r2.next()) {
12        String t = r2.getString(1);
```

```
13        ResultSet r3 = s3.executeQuery("select b from R where a = '" + z + "'");
14        while (r3.next() ) {
15          out.println( "[" + z + ", " + r3.getString(1) + ", " + t + "]");
16        }
17        r3.close();
18      }
19      r2.close();
20    }
21    r1.close();
22    s1.close(); s2.close(); s3.close(); c1.close(); c2.close(); c3.close();
```

P1 et P2 produisent-ils le même résultat? Quel est le coût de P2 ?

Question 4

On considère la requête R1 = R $\bowtie_a$ S. On veut calculer R1 en traitant la jointure par un algorithme de tri fusion. On donne une liste d'instructions parmi lesquelles certaines participent au calcul de R1. Pour chaque instruction utile, donner son numéro, le nombre de fois qu'elle est traitée et son coût. Répondre en triant les instructions dans l'ordre où elles sont traitées pour la première fois.

n°1 : r = executeQuery("select a, b from R order by a");
n°2 : r = executeQuery("select a, b from R order by b");
n°3 : r = executeQuery("select distinct a from R order by a");
n°4 : r.getString(1);
n°5 : r.getString(2);
n°6 : s = executeQuery("select a, c from S order by a");
n°7 : s = executeQuery("select a, c from S order by c");
n°8 : s = executeQuery("select distinct a from S order by a");
n°9 : s.getString(1);
n°10 : s.getString(2);

Exercice 4 : Requêtes réparties 5 pts.

Soit **R1**(a, b) sur le site S1, **R2**(a, c) sur S2. La requête R1 $\bowtie_a$ R2 est posée sur S3. Le schéma du résultat de la requête est (a, b, c). Les attributs sont des entiers positifs. La distribution des valeurs des attributs est uniforme. La taille, en octets, des attributs *a, b, c* est respectivement 1000, 1000 et 3000.
Le tableau suivant donne la taille des résultats intermédiaires en milliard (10^9) d'octets :

Expression	*taille*
R1	80
R2	400
π_a (R1)	10
π_a (R2)	20
R1 $\ltimes_a$ R2	40
R2 $\ltimes_a$ R1	100
R1 $\bowtie_a$ R2	T
π_a (R1 $\bowtie_a$ R2)	5

Remarque : on nomme T la taille du résultat de la requête

Question 1
a) Les affirmations suivantes sont-elles vraies ? Justifier brièvement.

a1) card(R1) = card(π_a (R1))
a2) T < 25 10^9 octets
a3) card(π_a (R1 $\bowtie_a$ R2)) = card(π_a (R1))
a4) card((π_a R1) $\cap$ (π_a R2)) = card(π_a (R2))

b) Donner la valeur numérique des expressions suivantes. Expliquer brièvement.

b1) card(R1)

b2) card(π_a (R1))

b3) le nombre moyen de nuplets de R1 pour une valeur de a donnée.

b4) card(π_a (R1 $\bowtie_a$ R2))

b5) card(R1 $\bowtie_a$ R2)

b6) T

Question 2 :

a) Donner la quantité (en milliard d'octets) de données à transférer pour traiter la jointure selon les plans suivants. Si nécessaire, donner une réponse en fonction de T.

P1) Transférer R1 et R2 vers S3 pour traiter la jointure sur S3

P2) Transférer R1 vers S2 pour traiter la jointure sur S2. Envoyer le résultat vers S3

P3) Transférer R2 vers S1 pour traiter la jointure sur S1. Envoyer le résultat vers S3

P4) Transférer les valeurs de R1.a vers S2, semi jointure sur S2, jointure sur S1, résultat envoyé vers S3.

P5) Transférer les valeurs de R2.a vers S1, semi jointure sur S1, jointure sur S2, résultat envoyé vers S3.

P6) Transférer les valeurs de R2.a vers S1, et celles de R1.a vers S2. Semi-jointures sur S1 et S2, puis jointure finale sur S3.

P7) Sur S1 : transférer les valeurs de R1.a vers S2.
Sur S2 : semi jointure dont le résultat, appelé R3, est transféré vers S3. Transférer également les valeurs de R3.a vers S1.
Sur S1 : semi jointure dont le résultat est transféré vers S3
Sur S3 : jointure finale

P8) Sur S2 : transférer les valeurs de R2.a vers S1.
Sur S1 : semi-jointure dont le résultat, appelé R3, est transféré vers S3. Transférer également les valeurs de R3.a vers S2.
Sur S2 : semi jointure dont le résultat est transféré vers S3
Sur S3 : jointure finale

b) Quel est le plan optimal ?

TD : Transactions réparties

Exercice 1 : Verrouillage réparti de données répliquées

1. Quel est le problème posé lorsqu'on veut verrouiller une donnée répliquée ?
2. Qu'est ce qu'un verrou global , un verrou local ?

.

3. On suppose qu'un gestionnaire de verrou centralise les demandes de verrou et d'accès aux données. Expliquer son fonctionnement en cas de données répliquées. Quels sont les avantages et défauts d'une telle approche ?

4. On cherche maintenant à ne pas mettre en place de gestionnaire centralisé, mais de tirer parti des gestionnaire de verrous et d'accès locaux. Pour cela, les transactions doivent respecter le protocole suivant :

lorsqu'elles ont réussi à verrouiller un certain nombre de répliques d'une donnée, les transactions peuvent déduire qu'elle ont accès à la donnée. On dit qu'elle obtiennent un *verrou logique* sur la donnée grâce aux *verrous physiques* qu'elles ont obtenus sur les répliques. On suppose qu'une donnée *d* est répliquée sur *n* sites.

   a. On suppose que n=3. Montrer qu'en obtenant au moins 2 verrous physiques exclusifs, une transaction peut avoir un verrou logique exclusif sur *d* et donc accès en écriture à *d*, donc à toutes ses répliques.
   b. Soit k (resp. k') le nombre de verrous physiques exclusifs (resp. partagés) à obtenir pour en déduire un verrou logique exclusif (resp. partagé). Donner la valeur minimale de k (resp. k') en fonction de n que le protocole doit respecter pour que le verrouillage logique fonctionne.

Exercice 2 : sérialisation globale et locale

On suppose les objets x et y stockés sur le site S1, et les objets z et w sur le site S2. Déterminer pour chacune des exécutions suivantes de deux transactions Ti et Tj les graphes de précédence locaux et global et dire si elle est sérialisable ou non.

1) Exécution 1 :
   S1 : Li(x), Lj(x), Ej(x), Ei(x)
   S2 : Li(w), Lj(z), Ej(w), Ei(w)

2) Exécution 2 :
   S1 : Li(x), Ei(x), Lj(x), Ej(x), Li(y), Ej(y)
   S2 : Lj(z), Ej(z), Li(z), Ei(w)

3) Exécution 3 :
   S1 : Li(y), Lj(x), Ej(x)
   S2 : Ei(z), Li(w), Lj(w), Ej(w)

Exercice 3 : détection d'interblocage

A et B sont stockés sur S1, C et D sur S2. On utilise un verrouillage en deux phases. On exécute les transactions T1 à T9. Les demandes d'accès sur les différents granules sont les suivants :

S1 A : L7, E3, E2, L1
B: L3, L4, E6, L9

S2 C: L9, E8
D : E8, L7, L5

1) Quelles sont les transactions globales ?
2) Décrire l'évolution de la table des verrous dans le cas où une demande de verrou partagé sera toujours satisfaite si possible. Représenter la table avant la première libération de verrou possible. Rappeler quel est l'inconvénient d'une telle gestion.
3) Proposer une autre politique permettant de résoudre le problème précédent et montrer l'état des tables de verrous dans ce cas.
4) Construire les graphe d'attente locaux et le graphe d'attente global. Est-il nécessaire d'utiliser les premiers en entier pour construire le dernier ? Que concluez-vous du résultat obtenu ?
5) On centralise la détection d'interblocages sur S1. Comparer deux solutions : dans la première, chaque modification du graphe local d'attente de S2 est communiqué à S1. Dans la seconde, la copie complète du graphe local de S2 est communiqué périodiquement à S1.

TME JDBC

Introduction:

L'objectif de ce TME est savoir accéder à un ou plusieurs SGBD depuis une application java, en utilisant l'interface **JDBC.** Savoir parcourir le résultat d'une requête, définir une requête paramétrée puis l'invoquer, présenter le résultat d'une requête, interroger le dictionnaire (méta-données) du SGBD. Savoir mettre en œuvre l'évaluation de requêtes réparties en utilisant JDBC : combiner les résultats de plusieurs requêtes posées sur différents SGBD.

Le serveur de données (Oracle sur la machine db-oracle), est un SGDB relationnel supportant l'interface JDBC. Il contient la base de données tennis dont le schéma est :

Joueur2 (NuJoueur, Nom, Prenom, AnNaiss, Nationalite)

Gain2 (NuJoueur, LieuTournoi, Annee, Prime, Sponsor**)**

Rencontre2 (NuGagnant, NuPerdant, LieuTournoi, Annee, Score**)**

Les attributs NuGagnant, NuPerdant et NuJoueur sont définis sur le même domaine. Les clés des relations sont soulignées.

Accès à une base de données avec JDBC

JDBC permet d'accéder au SGBD depuis une application écrite en langage java. Afficher la documentation en ligne du TME (lien TME JDBC depuis la page d'accueil), puis afficher la documentation java (lien API).

Utilisation des bibliothèques Java

Les bibliothèques java sont regroupées en packages. Un package contient plusieurs interfaces et plusieurs classes. Une classe (et une interface) contient des méthodes et des variables. La documentation en ligne permet de naviguer dans les bibliothèques pour déterminer les classes, interfaces et méthodes à utiliser pour construire une application java. Les classes et interfaces JDBC du package ***java.sql*** permettent d'accéder à une base de données. Naviguer dans la documentation du package java.sql pour répondre aux questions suivantes :

a) Donner le nom et le type des paramètres des méthodes getConnection de la classe DriverManager.

b) A quoi sert la méthode *next* de l'interface ResultSet ?

c) A quoi sert la méthode *setMaxRows* de l'interface Statement ?

d) Quelles sont les sous-interfaces de l'interface Statement ?

1 Première connexion à une base de données

Installer l'environnement logiciel (la procédure d'installation est détaillée sur la fiche technique, ci-après) puis exécuter le programme Joueur.
1.1) Etudier le programme Joueur.java. (voir en annexe). Commenter brièvement les lignes importantes pour expliquer chaque étape du programme.
1.2) Sur une feuille (à faire chez soi), représenter sous la forme d'un graphe, les scénarios d'accès à une base de donnée en utilisant les interfaces, classes et les méthodes de JDBC. Le graphe est défini comme suit :

- un **nœud** (rectangle) représente une interface ou une classe.
- un **arc** orienté (flèche) représente une méthode. L'arc est tel que :
- son origine est l'interface dans laquelle la méthode est définie,
- sa destination est l'interface du résultat de la méthode.

Le graphe doit contenir les classes, interfaces et méthodes du package java.sql qui sont utilisées dans *Joueur.java*, ainsi que les éléments suivants : PreparedStatement, ResultSetMedatata, DatabaseMetadata, executeUpdate, int, String et getMetaData.

2 Accès paramétré à la base de données

2.1) Exécuter le programme MaxPrime (saisir une année, par ex. 1992). Editer le fichier MaxPrime.java. Compléter l'en-tête avec vos nom et prénom. Compléter tous les commentaires pour expliquer ce que fait le programme.
2.2) Copier et modifier le programme précédent pour obtenir le programme *MaxPrime2.java* qui exécute la même requête **en boucle,** en demandant à chaque itération une nouvelle valeur à l'utilisateur. Le programme MaxPrime2 se termine lorsque la saisie de l'utilisateur est vide. Pour améliorer les performances du programme, deux conditions sont requises :

- la connexion vers le SGBD est ouverte une seule fois pendant toute la durée du programme (*i.e.,* réutiliser le même objet de type Connection à chaque itération).

- le SGBD analyse la requête une seule fois pendant toute la durée du programme (*i.e.,* utiliser l'interface PreparedStatement pour définir une requête paramétrée ; utiliser la méthode setInt pour affecter une valeur à un paramètre de la requête).

Remarque : ne pas oublier d'appeler le constructeur de MaxPrime2. (*ie*., remplacer new MaxPrime() par new MaxPrime2().

3 Requête générique

3.1) Quel est le rôle de l'interface ResultSetMetaData et de sa méthode getColumnCount ?

3.2) Compléter le programme *Generique.java* pour traiter une requête SQL quelconque passée en paramètre, et afficher les valeurs des tuples du résultat sous la forme « val_1 val_2 … val_n » (un tuple par ligne).

Tester l'exécution avec la requête donnant tous les Joueurs nés en 1972 :

```
java Generique ''select * from Joueur2 where annaiss = 1972''
```

3.3) Compléter le programme pour afficher en en-tête le **nom des attributs** du résultat.

Exemple d'exécution (ne pas chercher à tabuler le résultat)

```
NUJOUEUR    NOM         PRENOM      ANNAISS     NATIONALITE
-------------------------------------------------------------
1           MARTINEZ    Conchita    1972        Espagne
14          SAMPRAS     Pete        1972        Etats-Unis
```

3.4) Ecrire le programme *GeneriqueXHTML.java* qui affiche le résultat d'une requête quelconque dans un tableau formaté en XHTML. Tester l'exécution en stockant le résultat dans un fichier :

```
java GeneriqueXHTML ''select * from Joueur2'' > resultat.html
```

Exemple de résultat :

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">
<html>
  <head ><title>Résultat</title></head>
<body>
  <h3>La requete est : </h3> select * from Joueur2
  <h3>le resultat est : </h3>
  <table border="2">
   <tr><th>NUJOUEUR</th><th>NOM</th><th>PRENOM</th><th>ANNAISS</th><th>NATIONALITE</th></tr>
   <tr><td>1</td><td>MARTINEZ</td><td>Conchita</td><td>1972</td><td>Espagne</td></tr>
   <tr><td>2</td><td>NAVRATILOVA</td><td>Martina</td><td>1957</td><td>Etats-Unis</td></tr>
   …
   <tr><td>14</td><td>SAMPRAS</td><td>Pete</td><td>1972</td><td>Etats-Unis</td></tr>
  </table>
</body>
</html>
```

Visualiser graphiquement le résultat dans un navigateur avec la commande :

```
netscape resultat.html (ou mozilla resultat.html)
```

4 Schéma d'une relation

4.1) Quel est le rôle de l'interface DatabaseMetaData et comment obtenir une instance de ce type à partir d'une connexion ? Expliquer la méthode getColumns de l'interface DatabaseMetaData. Dans un SGBD les relations sont généralement organisées en hiérarchie à 2 niveaux (catalogue et schéma). Ainsi, une relation appartient à un schéma lui-même appartenant à un catalogue. Dans Oracle, l'organisation n'a qu'un seul niveau : une relation appartient à un schéma égal au nom de celui qui a créé la relation, il n'y a pas de niveau catalogue.

4.2) Créer le programme Schema.java qui affiche le nom et le type des attributs d'une relation passée en paramètre.

Exemple d'exécution

```
java Schema JOUEUR2                    // écrire le nom de la relation en MAJUSCULE
Le schéma de JOUEUR2 est :             // résultat affiché
NOM          TYPE
------------------------------
NUJOUEUR           NUMBER
NOM                VARCHAR2
PRENOM             VARCHAR2
ANNAISS            NUMBER
NATIONALITE        VARCHAR2
```

5 Jointure inter-bases

Soit une deuxième base de données (située dans un autre SGBD différent du premier), contenant la relation

SPONSOR (NOM, NATIONALITE) // nationalité des sponsors

La deuxième base de données est accessible seulement en lecture, par une connexion à la base oracle en tant qu'utilisateur «anonyme» avec le mot de passe «anonyme».La relation Sponsor contient **100 000** tuples. Soit la requête R1 :«Donner le nom et la nationalité des joueurs avec le nom et la nationalité de leurs sponsors, dans l'ordre des noms de joueur».

5.1) Quelle est la durée approximative d'une lecture séquentielle de la relation Sponsor ? Répondre en écrivant le programme *Generique2.java* (obtenu en modifiant *Generique.java*), puis en utilisant la commande *time* :

```
time java Generique2 "requete"…        // mesure le temps de réponse de la requête
```

Veiller à omettre l'affichage du résultat dans le terminal pour éviter de mesurer le temps d'affichage au lieu du temps de lecture des données.

5.2) Ecrire R1 en SQL.

5.3) Pourquoi ne peut-on pas exécuter cette requête R1 avec une seule connexion au SGBD ?

5.4) On veut obtenir la cardinalité du résultat de R1 ? Donner en SQL une requête R2 telle :

R2 est une sous-expression de R1, et R2 peut être exécutée entièrement sur le premier SGBD

la cardinalité de R2 est égale à celle de R1, *i.e.,* card(R2) = card(R1)

Exécuter R2 et donner la valeur de card(R1).

5.5) Compléter le programme Sponsor.java pour traiter R1. Pendant tout le traitement de la requête R1, combien d'instances de type Connection et Statement sont créées ? Combien de sous-requêtes sont exécutées ? Combien de fois la relation Sponsor est-elle lue ? Quels sont les nuplets transférés depuis le SGBD vers l'application java ?

Exemple d'exécution :

```
java Sponsor                           //commande
CONNORS, Etats-Unis, Dunlop, Ecosse
CONNORS, Etats-Unis, Lacoste, France
EDBERG, Suede, Dunlop, Ecosse
…
SAMPRAS, Etats-Unis, Reebok, Angleterre
WILANDER, Suede, Dunlop, Ecosse
WILANDER, Suede, Kennex, USA
```

5.6) Mesurer le temps moyen d'exécution du programme avec la commande time :

```
time java Sponsor                              //commande
…
real 0m3.957s          user 0m1.530s           // affiche le temps de réponse
```

5.7) Ecrire le programme SponsorTF.java pour traiter la requête R2 : «Donner le nom et la nationalité des joueurs avec le nom et la nationalité de leurs sponsors, dans l'ordre des **noms de sponsor**» en utilisant l'algorithme de jointure par tri fusion. Combien d'instances de type Connection et Statement sont créées pendant le traitement de la requête ?

5.8) Comparer les temps de réponse de R1 et R2. Mesurer (en pourcentage) l'écart entre le temps de réponse de Sponsor et SponsorTF. Interpréter le résultat. Proposer une solution Sponsor2.java pour améliorer le temps de réponse de la requête R2.

5.9) Détailler une solution pour traiter R1 en transférant les clés (voir cours : semi-jointure)

5.10) (facultatif) Proposer une implémentation pour d'autres algorithmes de jointure (grace join, hybrid hash join).

Fiche Technique pour les TME

Installer l'environnement logiciel (Java, JDBC)

```
cd                                  // aller dans le répertoire $HOME
tar zxvf $BD_TOOL/jdbc-etu.tgz      // installer l'archive
cd jdbc-etu                         // aller dans le répertoire de travail
```

Dans la suite du TME, sauvegarder tous vos programmes dans le répertoire de travail jdbc-etu

1.3) Editeur de texte:

Editer les fichiers avec emacs. Activer les options suivantes :

Programme en couleur : Menu Option > Syntax Highlighting

Repérage de la structure du programme : Menu Option > Paren Match Highlighting

1.4) Environnement java

Pour tester la compilation du code source en bytecode, compiler le programme de test Bonjour.java (cela crée le fichier Bonjour.class)

```
javac Bonjour.java
```

Pour tester la machine virtuelle lancer l'exécution du fichier Bonjour.class :

```
java Bonjour
```

Fichier Joueur.java

```
1  import java.sql.*;
2  import java.io.*;
3
4    public class Joueur {
5      String server = "frelon";
6      String port = "1521";
7      String database = "oracle";
8      String user = "p6lip000";
9      String password = "p6lip000";
10     String requete = "select nom, prenom from Joueur";
11     Connection connexion = null;
12     …
13     /** La méthode traiteRequete */
14     public void traiteRequete() {
15         try {
16            String url = "jdbc:oracle:thin:@" +server + ":" +port + ":"
17 +database;
18            connexion = DriverManager.getConnection(url, user, password);
19            Statement lecture =  connexion.createStatement();
20            ResultSet resultat = lecture.executeQuery(requete);
21            while (resultat.next()) {
22              String tuple = resultat.getString(1) + " " +
23 resultat.getString(2);
24              out.println(tuple);
25            }
26            resultat.close();
27            lecture.close();
28            connexion.close();
29         }
           catch(Exception e){ … }
       } }
```

Tme2PC

Transactions réparties : validation en 2 étapes (2 Phases Commit)

Préparation

- Voir le cours: protocole 2PC
- Installer les fichiers du tme:
  - tar zxvf $BD_TOOL/tme-2PC.tgz
  - cd tme-2PC
  - source config-simjava
  - javac -encoding ISO-8859-15 -source 1.4 Exemple1.java
  - appletviewer exemple1.html
  - si appletviewer ne fonctionne pas, il faut ouvrir un navigateur, double cliquer sur exemple1.html, régler la vitesse sur 100 environ, puis "run". Fermer le navigateur après chaque simulation. Une autre possibilité est de lancer firefox avec le nom du fichier html en paramètre.
  - pour les utilisateurs d'Eclipse, vous pouvez aussi lancer l'applet directement dans eclipse (click droit sur le fichier java, puis "run as java applet"

Dans les exercices suivants recopier les fichiers java et html :

- cp Exemple1.java Exemple*NN*.java
- cp exemple1.html exemple*NN*.html

puis éditer les nouveaux fichiers pour **remplacer** le nom de la classe Exemple1 par Exemple*NN*

Veiller à **recompiler** votre exemple avant chaque exécution car les noms de classes *Participant* et *Coordinateur* existent dans plusieurs exercices.

Documentation

- la doc java et simjava (voir les classes Sim_entity et Sim_event du package eduni.simjava)

Exercice 1 : Prise en main du simulateur

- Dérouler la simulation à faible vitesse (*speed entre 100 et 150*)
- Expliquer brièvement le fonctionnement du protocole implémenté dans le fichier Exemple1.java
- Comprendre les méthodes utilitaires de la classe Entité : envoyer_message, attendre_message_timeout, panne, date, ...

Exercice 2 : Tolérance aux pannes

Panne d'un participant

- Dans un fichier Exemple**2a**.java, compléter le protocole pour gérer la panne d'un participant avant de s'être déclaré prêt. Suivre le diagramme de la diapo 13.
  - Le coordinateur décide d'abandonner s'il n'a pas reçu tous les votes dans un délai de 300 unités de temps.
  - Est-ce que le coordinateur attend que les participants en panne aient repris, avant de répondre à l'application ?
  - Que se passe-t-il si un participant qui n'est pas en panne mais trop lent, reçoit la décision d'abandonner alors qu'il n'a pas encore envoyé son vote au coordinateur ?
- Dans un fichier Exemple**2b**.java, compléter le protocole pour gérer la panne d'un participant **après** s'être déclaré prêt (diapo 14).
  - Le coordinateur répète sa décision à un participant qui la lui demande.

Panne du coordinateur (diapo 15)

- Dans un fichier Exemple**2c**.java, compléter le protocole pour gérer la panne du coordinateur **après** l'étape 1.

Exercice 3 : Gestion des délais

Durée maximum de la validation

- Dans un fichier Exemple**3a**.java, modifier le protocole pour répondre à l'application dans un délai borné . Quelle sont les réponses possibles du coordinateur à l'application, après un temps D, lorsque la validation n'est pas terminée ?

Réponse anticipée

- Dans un fichier Exemple**3b**.java, modifier le protocole pour signaler, à l'application, l'abandon de la transaction dès qu'un participant a voté pour un abandon. Que fait le coordinateur lorsqu'il reçoit le prochain commit en provenance de l'application alors que les participants n'ont pas encore fini la validation précédente.

Exercice 4 : Coordonner plusieurs utilisateurs

- Dans un fichier Exemple**4**.java, modifier le simulateur pour gérer plusieurs transactions provenant simultanément de 2 applications.

Exercice 5: Décision majoritaire

- Dans un fichier Exemple**5**.java, modifier le protocole pour décider d'une validation dès que la majorité des participants a voté pour une validation. Dans ce cas, que répond le coordinateur aux participants minoritaires qui ont voté pour l'abandon ?

Documentation diverse

- tutorial simjava

LesTravauxDirigés, **LesCours**, **Accueil**

Propriétaire : **bdr** Dernière modification le avril 8, 2013 5:50 par **bdr**