GESTION DE GRANDES MASSES DE DONNEES - CM1

QUEL OBJECTIF :

Offrir à l'utilisateur final un accès satisfaisant aux données selon le paradigme le plus adapté au besoin.

QU'EST-CE QU'UN ACCES SATISFAISANT :

Les requêtes/traitements aboutissent à un résultat avec une latence acceptable pour l'usage

QU'EST-CE QUE LE PARADIGME LE PLUS ADAPTE AU BESOIN :

Le paradigme qui offrira les meilleures performances, avec le niveau de fiabilité (des résultats ou du système) souhaité par l'utilisateur, pour effectuer les traitements

LES BASES DE DONNEES PARALLELES

Principe :
- Orchestration des calculs d'une requête à travers plusieurs CPUs et disques avec une gestion de la mémoire adaptée.

Verrous :
- Gestion de l'équilibrage de charge
- Complexité

Différents types d'architectures :
- Architecture à Mémoire Partagée
- Architecture à Disque Partagé
- Architecture à Mémoire Distribuée

ARCHITECTURE A MEMOIRE PARTAGEE (SHARED MEMORY)

Modèle de programmation :
- Mémoire partagée (multiprocessus, multithreading)

Avantages :
- Simplicité, équilibrage de la charge

Inconvénients :
- Coût du Réseau (bus parallèle ou multibus), faible extensivité

ARCHITECTURE A DISQUE PARTAGE (SHARED DISK)

Avantages :
- Coût du Réseau, extensibilité, migration depuis les monoprocesseurs, sûreté de fonctionnement

Inconvénients :
- Complexité

ARCHITECTURE A MEMOIRE DISTRIBUEE (SHARED NOTHING)

Avantages :
- Coût, extensibilité, disponibilité

Inconvénients :
- Complexité, équilibrage de charge

BASE DE DONNEES REPARTIE

Définition :

C'est une base de données logique dont les données sont stockées dans différents SGBD interconnectés

CARACTERISTIQUES

Une base de données répartie dispose d'un schéma global et d'un schéma d'allocation a distribution des données est transparente pour l'utilisateur

REMARQUE IMPORTANTE

Besoin de formaliser les éléments de la base de données répartie : ' PRINCIPE DE DECOMPOSITION D'UNE BASE DE DONNEES CENTRALISEE '

RAPPEL SUR LA CONCEPTION D'UN SGBD CENTRALISE

Analyse des besoins
-> Modèle conceptuel de données
-> Modèle relationnel (logique)
-> Implémentation (physique)

CONCEPTION D'UN SGBD DISTRIBUE

Analyse des besoins
-> Modèle conceptuel de données
-> Modèle relationnel (logique)
-> Implémentation (physique)
(Construction du schéma global)
-> Fragmentation/ Réplication / Migration (physique)
-> Allocation (Placement)
(Distribution des données)

LE SCHEMA GLOBAL EST CONSTITUE

Schéma conceptuel global :
- Contenant la description globale et unifiée de toutes les données de la BDR
- Indépendant de la répartition des données

Schéma de placement (d'allocation) :
- Contenant les règles de correspondance avec les données locales
- Indépendant à la fragmentation et à la réplication

DISTRIBUTION DES DONNEES

MIGRATION :

Transfère d'une relation complète sur un site distant

Intérêt :
- Rapprocher les données du besoin

SQL :
- Création d'un wrapper et d'une 'foreign table' pour pouvoir faire en local une copie de la BD source à la BD cible

REPLICATION/DUPLICATION :

Création d'une copie conforme d'une table (ou ensemble de tuples) sur un site distant. La copie doit rester cohérente avec les données sources

Intérêt :
- Rapprocher les données des besoins : avoir les mêmes données sur différents sites

SQL :
- Utilisation d'un processus de Publication / Suscription afin de synchroniser les tables

FRAGMENTATION :

Décomposition d'une relation en plusieurs fragments qui sont transférés sur différents sites distants

Intérêt :
- Découper logiquement les données

SQL :
- Création d'un wrapper et d'une 'foreign table' pour pouvoir faire en local une copie partielle de la BD source à la BD cible en intégrant les contraintes de sélection.

DIFFERENTES FRAGMENTATIONS

FRAGMENTATION HORIZONTALE - SELECTION (WHERE)

IdP	NOM	Prénom	Secteur
1	BON	Jean	N
2	HETTE	Rose	N
3	NAULT	Pia	S
4	THARE	Guy	N
5	HONETTE	Marie	S
6	GNOLLE	Guy	S

Fragmentation horizontale selon les secteurs

PERSONNE_N : SELECT * FROM PERSONNE **WHERE** SECTEUR = "N"

IdP	NOM	Prénom	Secteur
1	BON	Jean	N
2	HETTE	Rose	N
4	THARE	Guy	N

PERSONNE_S : SELECT * FROM PERSONNE **WHERE** SECTEUR = "S"

IdP	NOM	Prénom	Secteur
3	NAULT	Pia	S
5	HONETTE	Marie	S
6	GNOLLE	Guy	S

RECONSTRUCTION PAR UNION

Personne = personne_N UNION Personne_S

SQL :

```
SELECT * FROM personne_N
UNION
SELECT * FROM Personne_S
```

FRAGMENTATION VERTICALE - PROJECTION (SELECT)

⚠ **Contrainte : la clé primaire doit appartenir à chaque fragment** ⚠

Table :

IdP	NOM	Prénom	Secteur
1	BON	Jean	N
2	HETTE	Rose	N
3	NAULT	Pia	S
4	THARE	Guy	N
5	HONETTE	Marie	S
6	GNOLLE	Guy	S

PERSONNE_ID_NOM_PRENOM = SELECT ID, NOM, PRENOM FROM PERSONNE
PERSONNE_ID_SECTEUR = SELECT ID, SECTEUR FROM PERSONNE

Personne_ID_NOM_PRENOM

IdP	NOM	Prénom
1	BON	Jean
2	HETTE	Rose
3	NAULT	Pia
4	THARE	Guy
5	HONETTE	Marie
6	GNOLLE	Guy

Personne_ID_SECTEUR

IdP	Secteur
1	N
2	N
3	S
4	N
5	S
6	S

RECONSTRUCTION PAR JOINTURE

Personne = Personne_ID_NOM_PRENOM JOIN Personne_ID_SECTEUR
SQL :
SELECT * FROM Personne_ID_NOM_PRENOM PINP
JOIN Personne_ID_SECTEUR PIS ON PINP.ID = PIS.ID

ATENTION

Sans la clé primaire dans les fragment vertical, reconstruction impossible !

FRAGMENTATION MIXTE

L'association des deux !!
Exemple : Fragmentation horizontale suivie d'une fragmentation verticale de chaque fragment

LA FRAGMENTATION EST CORRECTE SI ET SEULEMENT SI ELLE EST :

- Complète (chaque élément doit se trouver dans un fragment)
- Reconstructible (on doit pouvoir recomposer la base à partir de ses fragments)
- Disjointe (chaque élément de la base (hormis les clés) ne doit pas être dupliqué)

GESTION DE GRANDES MASSES DE DONNEES - CM2

TRAITEMENT DE JOINTURES INTER-SITES

Jointures inter-sites :
C'est une jointure interrogeant des données distribuées sur différents sites, le coût de transfert dépendant du plan d'exécution !
Un plan d'exécution possible :

- Celui qui demande, récupère tout et c'est lui qui fait les calculs.
- Collaborative

COMMENT AMELIORER LES PERFORMANCES DE TRAITEMENT ?

Techniques d'exécution distribuée :
En tenant compte des propriétés du réseau :

- Row blocking : transfert de données par batch plutôt qu'individuellement
- Multicast : dans le cas de transfert des mêmes données sur plusieurs sites, valoriser les chemins les moins coûteux
- Multithreading : parallélisation de certains opérateurs
- Problème de communication synchronisée inter-thread

ORCHESTRATION D'UNE JOINTURE INTER-SITE :

Principe de base :
- Transférer la plus petite des deux relations (si on n'a pas le choix)

Solutions possibles

- Algorithme des semi-jointures
- Jointures parallèles
- Jointure parallèle avec partitionnement des données sur attribut de jointure
- Jointure parallèle sans partitionnement des données sur attribut de jointure

Algorithme des semi-jointures :

- Réduire la quantité d'information transférée pour exécuter une jointure quand peu de n-uplets participent à la jointure

Jointures parallèles avec partitionnement des données :

- Distribuer le traitement d'une jointure sur plusieurs machines, en tenant compte du partitionnement

Jointures parallèles sans partitionnement des données :

- Eviter de rassembler l'ensemble des fragments horizontaux sur un même nœud.

ALGORITHMES DE JOINTURES : RAPPELS

Approche par boucle (LOOP)
Pour chaque tuple de R, on parcourt S (avec ou sans index).
Approche par tri-fusion (SORT-MERGE)

- Algorithme du sort-merge join :
- On trie R et S sur les attributs de jointure, puis scan en parallèle pour permettre un seul parcours de chaque relation
- Partitionnement des données en amont du tri
- Facilite la parallélisation
- Généralement, nécessite la matérialisation les fragments ➔ latence supplémentaire
- Plusieurs niveaux de fusion en aval du tri

Approche par hachage (HASH)

- Algorithme du hash join
- Transfert de la plus petite des relations dans une structure de hachage avec les attributs de jointure comme clé, pour accéder efficacement aux valeurs
- Partitionnement des données en amont
- Facilite la parallélisation
- Création de plusieurs structures de hachage

REPLICATION LOGIQUE

Solution pour éviter des transferts est la réplication logique de certaines données

- Rapprocher les données des besoins

Important :

- Prendre en compte a fréquence de mise à jour des données répliquées
- Ce qui peut aussi impacter les performances

Au niveau configuration :

- Le nombre de connexions simultanées ➔ **max_connections**
- La taille des buffers de manipulation des données partagés ➔ **shared_buffers**
- La taille des buffers de maintien de l'intégrité des données ➔ **wal_buffers**
- La taille du cache ➔ **effective_cache_size**
- La mémoire utilisable pour le traitement d'opérateurs complexes ➔ **work_mem**
- La mémoire utilisable pour les processus de maintenance ➔ **maintenance_work_mem**
- Le coût des accès disques

Au niveau des données :

- L'exploitation des contraintes d'intégrités
- Le partitionnement des tables

Au niveau des requêtes

- L'utilisation d'index
- La parallélisation des calculs
- La matérialisation de résultats (intermédiaires)

PARTITIONNEMENT ET PARALLELISATION : LE 'SHARDING'

Principe :
- Fragmentation horizontale garantissant un partitionnement des données pour un traitement en parallèle des fragments

Choix de fragmentation :
- Basé sur des plages de valeur de la clé de 'sharding' (Range-based partionning)
- Basé sur une valeur de hachage de la clé de 'sharding' (Hash partionning)

Mise en place du 'sharding' en deux étapes :
- Allocation des Shards aux machines (Plusieurs Shards possibles par machine)
- Affectation des tuples à leur Shard

Range-based partionning :
EX : Shard 1 : Valeur < 15
Shard 2 : 15 < Valeur < 25
Shard 3 : 25 < Valeur

Hash partitionning :
EX : shard 1 : hash(id) % 3 -> 0
shard 2 : hash(id) % 3 -> 1
shard 3 : hash(id) % 3 -> 2

AVANTAGES DU SHARDING

Gain de temps de traitement :
- Traitement en parallèle (sur différentes machine CPU, RAM) des 'shard' ➔ Possibilités d'adapter la voilure de l'infrastructure (nombre de CPU et RAM) en fonction de la quantité de données à traiter

Evite la centralisation des données sur un seul et unique serveur (sécurité)

INCONVENIENTS DU SHARDING

Une mauvaise stratégie de 'sharding' (choix de la clé de sharding) peut rendre contreproductif le processus à cause de 'shard' de taille trop hétérogène
- Rééquilibrage complexe

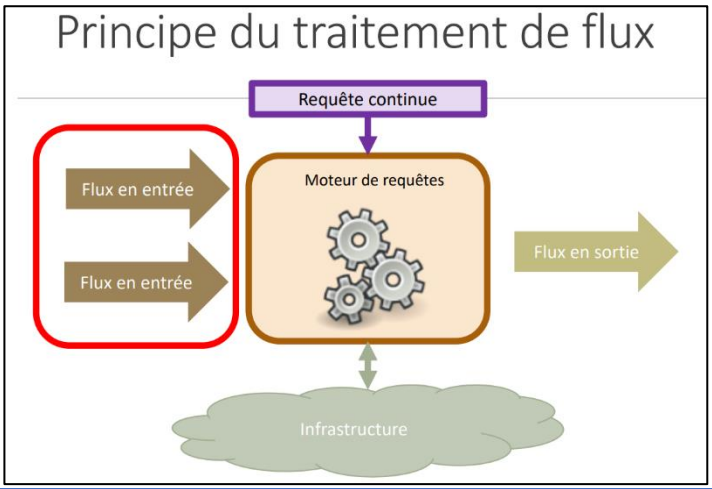
Sensible à la performance des réseaux
Sensible à la disponibilité et l'évolution du cluster de machine

GESTION DE GRANDES MASSES DE DONNEES - REVISION CM3

SYSTEME DE GESTION DES FLUX DE DONNEES (DATA STREAM MANAGEMENT SYSTEM DSMS)

Traitement de flux de données à débit variant

Contraintes :
- Maîtriser la qualité des résultats retournés
- Maîtriser les ressources (CPU, RAM bande passante) nécessaires au traitement



FLUX EN ENTREE

DEFINITION

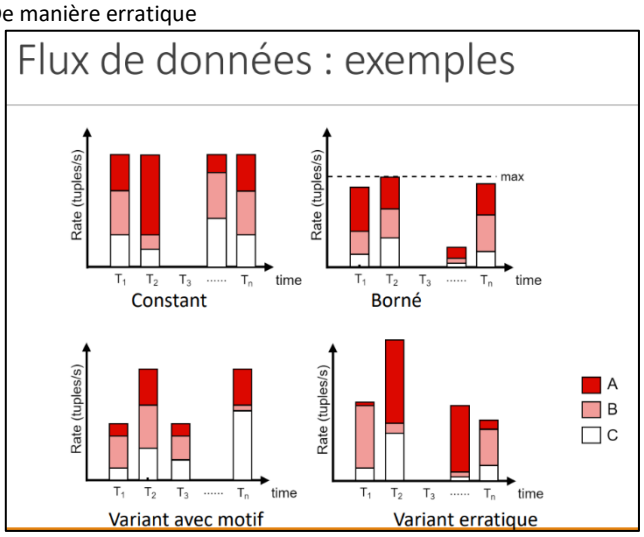
Un flux S est un ensemble potentiellement infini de paires < s, τ >, où :
- s est un tuple respectant le schéma de S
- τ est le timestamp de cet élément.

Remarque :
- Un timestamp τ n'appartient pas au schéma de S et il peut y avoir zéro, un ou plusieurs éléments de S partageant le même τ.

CARACTERISTIQUES

Un flux S est caractérisé par :
- La valeur de son débit exprimé en nombre de tuples par unité de temps

L'évolution de son débit dans le temps :
- Débit constant
- Débit borné
- Débit variant
- Avec motifs régulier
- De manière erratique



REQUETES CONTINUES

DEFINITION

Une requête continue est une requête qui est émise une fois et qui s'exécute de façon logique sur les données jusqu'à ce qu'elle soit terminée.
Le traitement d'une requête continue nécessite l'exécution d'opérateurs (avec ou sans état) avec une fréquence et une portée paramétrée.

OPERATEURS

Le traitement d'une requête continue peut nécessiter des opérateurs de différents types

Opérateur sans état ('Stateless') :
- Calcul indépendant de l'état/résultat calculé précédemment
- Exemple : requête appliquant un filtre sur les données

Opérateur avec état ('Stateful') :
- Calcul dépendant de l'état/résultat calculé précédemment
- Exemple : requête calculant l'évolution du nombre moyen de valeurs apparaissant toutes les x secondes

REQUETES CONTINUES : PARAMETRES

Exemple de requete : Donner la moyenne sur les 15 dernières minutes du niveau de polluants HAP observées à partir des 10 000 capteurs répartis dans la région Auvergne-RhôneAlpes et ce toutes les 5 minutes.

Une requête continue dispose :
- D'une fréquence de résultat attendu : "toutes les 5 minutes."
- D'une portée sur laquelle la requête est calculée : "sur les 15 dernières minutes."

FENETRAGE

Rappel :
- Un flux est infini

Conséquence :
- Il n'est pas possible d'interroger le flux dans sa globalité

Besoin :
- Système de fenêtrage

Paramètres : taille, pas

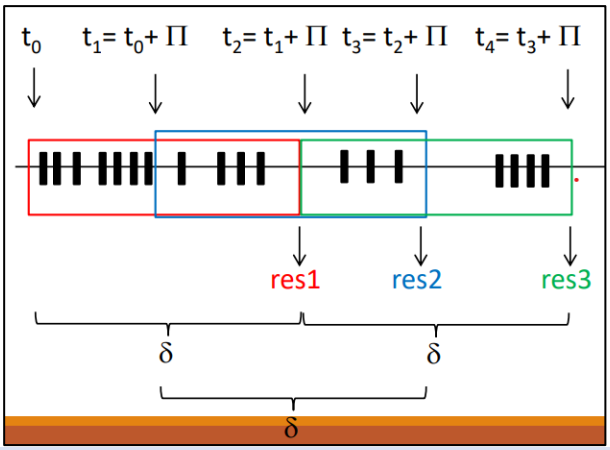
Différents types :
- Fenêtres basées sur le temps
- Fenêtres basées sur le nombre de tuples reçus
- Fenêtres basées sur le nombre de valeurs de tuples reçus

FENETRES SAUTANTES

- Fenêtre glissante où pas = taille

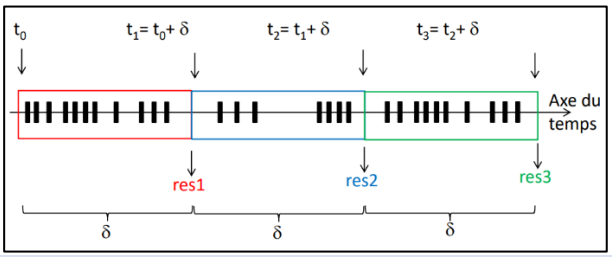
FENETRE BASEE SUR LE TEMPS (TIME-BASED)

- La fenêtre est définie à partir d'une durée
- Le calcul de la requête se fait sur les tuples reçus depuis δ secondes et ce toutes les π secondes



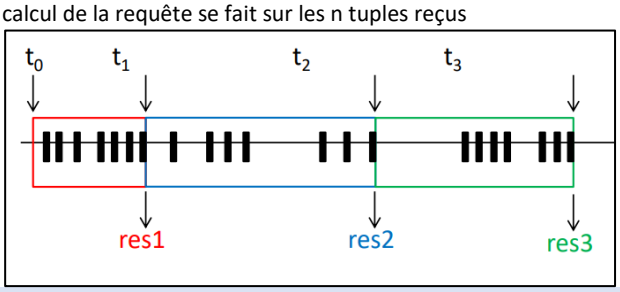
FENETRE SAUTANTE

- La fenêtre est définie à partir d'une durée
- Le calcul de la requête se fait sur les tuples reçus depuis δ secondes et ce, toutes les π secondes



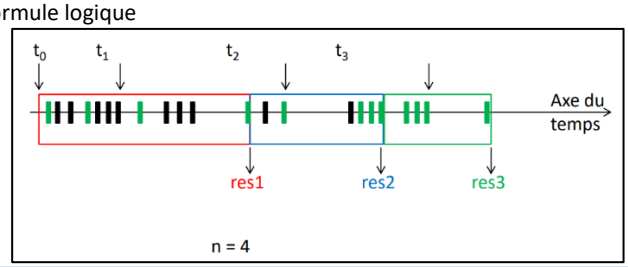
FENETRES SUR LE NB DE TUPLES REÇUS (TUPLE-BASED

- La fenêtre est définie à partir d'un nombre de tuples reçus
- Le calcul de la requête se fait sur les n tuples reçus



FENETRES SUR LE NB DE TUPLES REÇUS (PARTITION-BASED)

- La fenêtre est définie à partir d'un nombre de valeurs de tuples reçus
- Le calcul de la requête se fait sur les n tuples reçus et vérifiant une formule logique



GESTION DU TEMPS

Problème de synchronisation présent en cas de plusieurs sources (ordre non garanti, latence réseau…).

Système en 'battement de cœur' possible de deux manières :

- Associer un timestamp à chaque tuple entrant.
- Chaque source envoie une ponctuation lorsqu'elle a fini d'envoyer des tuples associées à un timestamp et la synchronisation s'effectue grâce aux sources

INFRASTRUCTURE

CENTRALISEE

Multicœurs :

- Possibilité de parallélisation des threads

DISTRIBUEE

Grille, Cloud :

- Possibilité de parallélisation des threads
- Mise à disposition de nouvelles ressources
- Introduction de latence réseau

SYSTEMES DE GESTION DE FLUX (DSMS)

PERFORMANCE DE TRAITEMENT

Pourvoir obtenir des résultats en quasi-temps réel
Optimiser le traitement de la requête

QUALITE DES RESULTATS

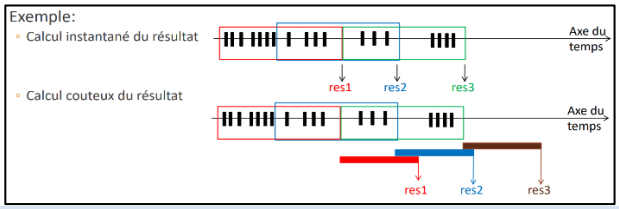
Obtenir des résultats en ayant limité les pertes de données
Eviter la congestion des opérateurs

ADAPTABILITE DES RESSOURCES

Avoir les ressources nécessaires pour traiter le flux
Ne pas bloquer des ressources inutilement

REMARQUE

La fin d'une fenêtre n'implique pas forcément la disponibilité d'un résultat.



CONGESTION

Le système est considéré comme congestionné, si :

- Des données en entrée du système sont perdues (file d'attente saturée)
- Des données avec TTL dépassé

ELASTICITE DES DSMS

DEFINITION

Adapter les ressources nécessaires au traitement des requêtes

ENJEUX (INTERSECTION BIG DATA / GREEN IT)

Performance du traitement de la requête
Qualité des résultats retournés

Consommation énergétique :

- Pouvoir adapter la quantité de ressources nécessaires en cas de variations du débit des flux à traiter

Automaticité :

- Pouvoir adapter automatiquement les ressources

GESTION DE L'ELASTICITE

Deux niveaux de traitements de l'élasticité :

- Ajout/suppression de ressources supplémentaires pour l'exécution des threads
- Concentration/Etalement des threads

Deux contextes :

- Ressources disponibles et suffisantes
- Ressources limitées et insuffisantes

APPROCHES WORKFLOW

DEFINITION

Les requêtes sont considérées comme des graphes d'opérateurs

TRAITEMENT DE REQUETES

Choix de la topologie

- Définition d'une topologie réduisant le nombre de tuples manipulés

Choix du degré de parallélisme des opérateurs

- Définition du nombre de threads permettant de paralléliser l'exécution d'un opérateur

Choix de l'allocation d'une tache sur une unité de traitement

- Définition d'une stratégie d'allocation des threads sur les unités de traitement

Choix de l'implémentation des opérateurs

- Définition d'une stratégie de sélection d'implémentation des opérateurs

STRATEGIE DE REPLICATION

Réplication incrémentale :
- Ajout / suppression d'une réplique à la fois par reconfiguration
- Peut générer de nombreuses reconfigurations

Réplication multiple :
- Ajout / suppression de plusieurs répliques à la fois par reconfiguration
- Peut générer de fortes variations de ressources

STRATEGIES D'ALLOCATION - DIFFERENTES STRATEGIES :

Orienté répartition :
- Round robin des affectations des threads sur les unités de traitements
- Permet d'équilibrer la charge (si les temps de traitement des tuples considérés comme identiques)
- Pas de considération du réseau

Orienté échange réseau :
- Tient compte de la position des opérateurs dans la topologie pour choisir leur affectation
- Permet de réduit le trafic réseau
- Mais concentre la charge sur certaines unités de traitement

Orienté ressources :
- Tient compte des ressources (CPU, RAM) disponibles sur une unité de traitement
- Permet une occupation maximale d'un sous-ensemble minimal de ressources (algo sac à dos)
- Mais reste faible robustesse en cas de variation du flux

FOCUS SUR LE DEGRE DE PARALLELISME

Changer le degré de parallélisme d'un opérateur. Quand ? :
- La détection de la congestion de l'opérateur (approche curative)
- La détection d'un risque de congestion de l'opérateur (approche préventive)

Comment ?
- En répliquant un opérateur (scale out) / supprimant une réplique (scale in)

TP1 - DEPLOIEMENT D'UNE BASE DE DONNEES REPARTIE

DEFINITION DES CONNEXIONS INTER-BASES

Définir le 'wrapper' qui permettra de vous connecter à la base :

```
CREATE EXTENSION IF NOT EXISTS postgres_fdw;
CREATE SERVER small FOREIGN DATA WRAPPER
postgres_fdw OPTIONS (host '192.168.246.201',
port '5432', dbname 'insee');
```

Définir l'association utilisateur local/utilisateur distant qui vous donnera le droit d'accès à la base insee :

```
CREATE USER MAPPING FOR "postgres" SERVER small
OPTIONS( user 'etum2' ,password 'etum2');
```

Définir les 'foreign tables' dbase insees depuis une VM :

```
CREATE FOREIGN TABLE remote_personnes(idp
serial4, nom varchar(80), prenoms varchar(80))
SERVER small OPTIONS(schema_name 'public',
table_name 'personnes');
```

Définir une 'table' à partir d'une 'foreign tables' :

```
CREATE TABLE personnes AS SELECT * (ou les
colonnes qui nous intéressent si fragmentation
vertical) FROM remote_personnes WHERE (si ya
besoin partition horizontale);
```

Créer une 'publication' :

```
CREATE PUBLICATION pub_update_mairie FOR TABLE
mairie;
```

Créer une 'suscriptions' :

```
CREATE SUBSCRIPTION sub_update_mairie CONNECTION
'host=192.168.246.205 port=5432 password=etum2
user=etum2 dbname=insee' PUBLICATION
pub_update_mairie;
```

Créer un 'Trigger' sur les updates :

```
CREATE TRIGGGER TT AFTER UPDATE ON commune c FOR
EACH ROW BEGIN UPDATE personne set lieunaiss =
new.c.com where lieunaiss = old.c.com; END
```

TP 2 - GGMD

DEFINITION DES CONNEXIONS INTER-BASES

Création d'un index sur personne :

```
CREATE INDEX Q2 ON personne (nomprenom,
datenaiss, lieunaiss);
```

TP 3 - SPARK

LECTURE DE DONNEES

- `sc.textFile(path)` : l'appel à cette méthode crée un RDD à partir des lignes du fichier. A noter que le `path` du fichier peut être un chemin réseau, ce qui nous sera utile par la suite.

TRANSFORMATIONS

- `map(func)` : applique une fonction à chacune des données.
- `filter(func)` : permet d'éliminer certaines données.
- `flatMap(func)` : tout comme map, mais chacune des données d'entrée peut être transformée en plusieurs données de sortie.
- `sample(withReplacement, fraction, seed)` : récolte un échantillon aléatoire des données. Cette méthode est utile pour tester un algorithme sur un petit pourcentage de données. L'argument `seed` permet de réaliser des expériences reproductibles. Exemple : `.sample(False, 0.01, 42)`.
- `distinct()` : supprime les doublons.
- `reduceByKey(func)` : applique une fonction de réduction aux valeurs de chaque clé. Par exemple, la fonction `func` peut renvoyer le maximum entre deux valeurs.
- `sortByKey(ascending)` : utilisée pour trier le résultat par clé.
- `join(rdd)` : permet de réaliser une jointure, ce qui a le même sens que dans les bases de données relationnelles.

ACTIONS

- `reduce(func)` : applique une réduction à l'ensemble des données.
- `collect()` : retourne toutes les données contenues dans le RDD sous la forme de liste.
- `count()` : retourne le nombre de données contenues dans RDD.

EXEMPLE

```
cleanedTortoises = tortoises.filter(lambda t:
len(t) > 1 and int(c[t]) == 21)

jsonTortoises = cleanedTortoises.map(lambda t: {
"id": int(t[1]), "top": int(t[2]), "position":
int(t[3]), "nbAvant": int(t[4]), "nbTour":
int(t[5]), "distanceTotal": 254 * int(t[5]) +
int(t[3]) })

min = values.reduce( (a, b) => a min b)
```