

MU5IN852

Bases de Données Large Echelle

octobre 2020

hubert.naacke@lip6.fr

Plan des séances BDLE

- Données multidimensionnelles, OLAP
- Map reduce
 - Spark
 - Algo de ML en map reduce
- Cluster computing
 - exécution distribuée et optimisation de requêtes
- Big graphs
 - Bulk synchronous parallel processing
 - Modèle et langage de requêtes pour des grands graphes

Transversal

- savoir faire sur Spark et la gestion de big data

Mise à niveau

Rappels sur SQL

Motivations

SQL est un langage « durable »

- Fondation théorique solide
- Pérenne face à l'évolution des solutions big data
 - SQL pour l'accès aux SGBD relationnels « classiques »
 - SQL supporté dans les systèmes récents issus du NOSQL, exple Google Spanner
- Générique : cas d'usage très variés
 - ETL, préparation de données
 - BI / E-commerce
 - Stream
- Langage standardisé et largement adopté
 - Langage pivot entre un DSL applicatif et un système de gestion de données.
 - Exple : TAO de Facebook : accès réseau social vers SQL

Motivations :

SQL est un langage efficace

- Efficacité : traitement plus rapide
 - Une requête SQL peut être optimisée automatiquement
 - Accélère le traitement d'une requête sans la modifier
- SQL est un langage déclaratif. Avantages :
 - Indépendant de l'organisation réelle des données
 - Indépendant des algorithmes qui implantent les opérations de l'algèbre relationnelle
- Transformation SQL --> programme
 - Nombreuses opportunités pour optimiser une requête SQL
 - Reformulation logique en une requête équivalente
 - Invocation des primitives de calcul des processeurs modernes
 - Gestion dédiée des ressources cpu, mémoire

Schéma relationnel

- Description des données selon le modèle relationnel
 - Un schéma relationnel est un ensemble de relations
 - Une relation représente un ensemble de tuples
 - Un tuple a plusieurs attributs, chacun avec un nom et un type
- Clé d'une relation
 - Une clé sert à identifier chaque tuple d'une relation
 - Une clé est composée d'un ou plusieurs attributs
- Référence entre relations
 - Lien entre entités : une relation peut faire référence d'autres relations
 - Une clé étrangère fait référence à la clé primaire d'une relation
- Contraintes logiques
 - domaine d'un attribut, valeur non nulle, unicité, prédicat global

Requêtes SQL

- Mots-clés
 - `SELECT ... FROM ...` peut être complété de :
 - `WHERE ... GROUP BY ... ORDER BY ...`
- Principales opérations
 - Projection
 - Sélection
 - Agrégation
 - Regroupement
 - Jointure
 - Opérations sur des ensembles :
 - union, intersection, différence
 - Tri

Projection et valeurs distinctes

- Schéma: **Visite** (photoID, personID, ville, pays, note)
- Projection : liste des attributs à garder dans le résultat
`Select personID, ville
from Visite`
- Projection sans doubles
 - Select `distinct` personID, ville from Visite
 - Le résultat ne contient pas 2 nuplets avec la même personne et la même ville
- Renommage dans une projection
`Select ville as city
from Visite`
- Appliquer une fonction sur un attribut : valeur--> valeur
 - Permet de générer des nouveaux attributs
`Select upper(pays) as PAYS from Visite`

Sélection : WHERE

Schéma: **Visite** (photoID, personID, ville, pays, note)

- Sélection = filtre multi critères exprimé dans la clause **where**
Select *
from Visite
where *prédicat*
- Prédicat simple avec les opérateurs =, <, >, <>, like
Select * from Visite **where** note = 5
Select * from Visite **where** ville **like** 'New%'
- Prédicat composé avec des connecteurs logiques
 - and, or, not, in, between
Where (note **between** 2 and 4)
and (pays **in** ('France', 'Italie') **or** ville='Oslo')

Agrégation

Schéma: **Visite** (photoID, personID, ville, pays, note)

- Fonction d'agrégation :
 - f: ensemble de valeurs --> valeur

```
Select f(attribut)
From Visite
```
- Fonctions prédéfinies :
 - Ensemble de nombres --> Nombre
 - max(), min(), sum(), avg()
 - Ensemble de tuples --> Nombre
 - count(*)
- Fonctions ad-hoc définies par l'utilisateur
- Exprimer plusieurs agrégations

```
Select min(note), max(note), avg(note), count(distinct pays)
From Visite
```

Regroupement : GROUP BY

Schéma: **Visite** (photoID, personID, ville, pays, note)

- Regroupement toujours suivi d'agrégations
 - Découper une relation en plusieurs groupes disjoints
 - Chaque groupe produit **un et un seul** tuple du résultat
 - Les attributs définissant le regroupement peuvent être projetés dans le résultat
 - Les autres attributs doivent être agrégés
 - sauf si on sait qu'ils dépendent d'un attribut du regroupement

Select pays, count(*) as nbreVisite

From Visite

Group by pays

Regroupement : exemples

- Groupe composé de plusieurs attributs
Select personID, pays, count(*) as nbreVisite
From Visite
Group by personID, pays
- Un regroupement sans agrégation = une projection sans doubles
Select personID, pays
From Visite
Group by personID, pays
équivalent à
Select distinct personID, pays
From Visite
- Un regroupement avec attributs 'redondants'
VisiteDetail (photoID, personID, profession, ville, pays, note)
Select personID, profession, count(*)
From VisiteDetail
Group by personID, profession

Jointure sur clé

- Schéma

Visite (photoID, **personID**, ville, pays, note)

Personne (**id**, profession, age)

- Jointure sur clé = **équi-jointure**

Select * from Visite v, Personne p

Where v.personID = p.id

- La clé peut être composée de plusieurs attributs

InfoVille (nomVille, pays, population)

select v.photoID, i.population

from Visite v, InfoVille i

where v.pays = i.pays

and v.ville = i.nomVille

Jointure générale

Généralisation du prédicat de comparaison entre 2 relations

Schema: Personne (id, profession, age, profil)

- Thêta-jointure

```
Select p1.profession as j1, p2.profession as j2  
From Personne p1, Personne p2  
Where p1.age < p2.age
```

- Jointure par similarité

- Fonction de similarité entre les 2 tuples $\text{sim}: t \times t \rightarrow \text{nombre}$
- Souvent utilisé dans le cas d'une auto-jointure

```
Select p1.id, p2.id  
From Personne p1, Personne p2  
Where sim(p1.profil, p2.profil) > 0.9
```

- Produit cartésien

- Génère toutes les paires de tuples

Jointure externe

- Situation
 - Compléter un tuple avec des informations détaillées qui n'existent que pour une partie des tuples
 - Cas d'une association optionnelle entre 2 entités
- Exemple
 - Personne(nom, age) Vélo(nomP, marque)
 - Obtenir PersonneInfo(nom, age, marque) pour toutes les personnes y compris celles qui n'ont **pas** de vélo
- Syntaxe
 - Select p.nom, p.age, v.marque
 - From Personne p **left outer join** Vélo v **on** p.nom = v.nomP

Requêtes imbriquées et négation

- Schéma
Visite (photoID, personID, ville, pays, note)
- Imbrication dans le where : sous-requête = valeur
Select photoID
From Visite
Where note = (select max(note)
from Visite where personID='bob'))
- Imbrication dans le where : sous requête = relation
Select *
From Visite
Where pays in (select pays from Visite where note =1)
- Négation
Select *
From Personne p
Where p.id not in (select personID from Visite where note < 4)

Opérations sur des ensembles

- Intersection

(select ...) **intersect** (select ...)

Exple: les personnes ayant visité Paris et Oslo

- **Union**

- Exple : les personnes ayant visité Paris *ou* Oslo

- Union de tuples issus de plusieurs relations

- Exple: Les personnes ayant visité Paris *et* celles ayant 25 ans

- Union conservant les doubles : union **all**

- Différence

(select ...) **minus** (select ...)

Exple: les personnes qui ont visité Paris mais jamais Oslo

- Division

- S'exprime avec deux négations imbriquées

Division: exemple

- « Les personnes ayant visité tous les pays » s'écrit:
 - Les personnes A pour lesquelles
 - Il n'existe pas de pays P tel que
 - Il n'existe pas de Visite V pour la personne A et le pays P.

Select A.id

From Personne A

Where not exists (

 Select *

 From Pays P

 Where not exists (

 Select *

 From Visite v

 Where v.personID = A.id and v.pays = p.pays)

Tri : ORDER BY

- La clé de tri peut être composée
Select v.photoID, v.personID, v.pays, v.note
From Visite v
Order by personID, pays
- Sens du Tri ascendant **asc** , descendant **desc**
 - Order by pays **asc**, note **desc**
- Le tri peut être évalué «en dernier» après le select
Select v.personID as numPersonne
From Visite v
Order by numPersonne
- Top k sur des données triées : LIMIT
Order by pays
Limit k

Complément de syntaxe SQL

- Alias de relation
 - From Visite **v1**, Visite **v2**
- Prédicats sur les valeurs manquantes
 - Tester une valeur nulle = non renseignée
 - Where pays **is not null**
 - Where note is **null** (différent de: where note = 0)
- Imbrication dans la clause from
 - Select **t**.note
 - From Visite v, (**select** note from ... where ...) **t**
- Alternative
 - Select **if**(note>3, 'haute', 'basse') as niveau from Visite

SQL : exercices

- Objectif : penser SQL
 - Reformuler une requête en langage naturel en :
 - Une expression logique : il existe, il n'existe pas ...
 - Une expression algébrique : enchainement d'opérations
- Nombreuses ressources en ligne
 - UE : L3 BD (3IN009), M1 MLBDA (4IN801)
 - Exercices SQL sur Leetcode
- Requêtes sur la base [Mondial](#)
 - Voir le [TP de M1](#) avec le SGBD H2 et l'interface SQLWorkbench

SQL: Conclusion

- SQL : Abstraction générale pour
 - Définir des données (semi) structurées
 - Exprimer des traitements déclaratifs
- Une requête SQL est un « objet d'étude »
 - Reformulé en une requête équivalente
 - Traduit dans un autre langage
 - Optimisé
 - Pré calculé
 - Adapté à un environnement d'exécution
 - etc ...

Travaux pratiques

- Evaluer des requêtes SQL avec un outil scalable
- Plateforme Spark
 - Moteur SQL
 - Databricks community : community.cloud.databricks.com
 - Interface : Notebook mixant SQL et python
 - Supporte la syntaxe SQL : tag %sql
 - Le résultat d'une requête SQL est manipulable en python