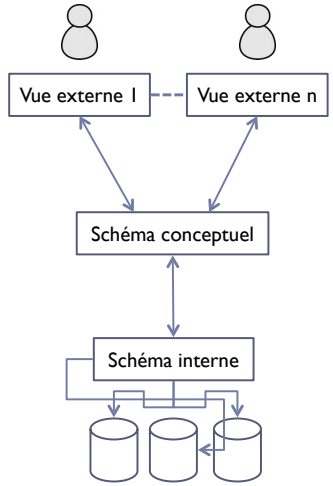
Cours Optimisation

SQL dans les langages de programmation

R3.07

Architecture des SGBD

- Architecture à trois niveaux
  - Niveau externe
    - Présentation de la BD aux utilisateurs
    - Deux utilisateurs différents signifie deux vues différentes des données
  - Niveau conceptuel
    - Définition de la BD (attributs, types, …)
    - Manipulation de la BD (requêtes)
  - Niveau interne
    - Stockage physique de la BD

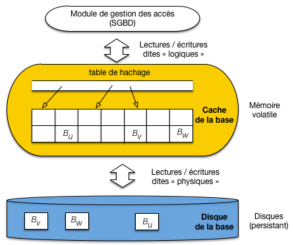


Niveaux de schémas

- Niveau externe
  - Vision spécifique du schéma conceptuel par un groupe d'utilisateurs
- Niveau conceptuel
  - Structure canonique des données qui existent dans une entreprise sans souci d'implantation en machine
  - Une vue intégrée pour tous les utilisateurs
    - Schéma conceptuel = schéma relationnel
- Niveau interne
  - Structure de stockage supportant les données
  - Fichiers (localisation), organisation du fichier. chemins d'accès (index), …

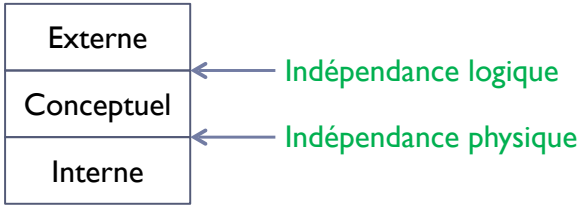
L'entreprise

SGBD



Indépendance des données

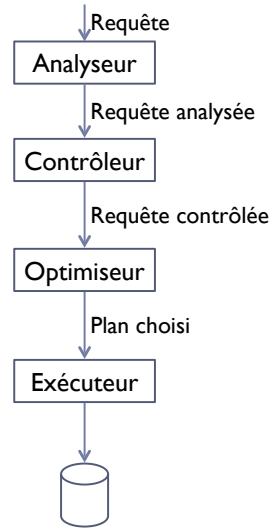
- Indépendance logique des programmes et des données
  - Pouvoir modifier les schémas externes sans changer les schémas conceptuels
  - Indépendance entre les différents utilisateurs
- Indépendance physique des programmes et des données
  - Pouvoir modifier les schémas internes sans changer les schémas conceptuels et externes



Traitement et optimisation des requêtes

Traitement d'une requête SQL

1) Analyse lexicale, syntaxique, et sémantique de la requête
2) Contrôle d'intégrité et contrôle d'autorisation
3) Ordonnancement des opérations élémentaires, optimisation, et élaboration d'un plan d'exécution
4) Exécution du plan, contrôle de concurrence, et atomicité des transactions



Optimiseur de requêtes

- Amélioration des temps d'exécution (différences considérables)
- Transformation des expressions : recherche d'une expression plus efficace
- Optimisation en fonction de la structure et du contenu de la base (statistiques internes)
- On ne cherche pas nécessairement la meilleure stratégie d'exécution
- Il faut faire attention au temps passé à optimiser

Etapes de l'optimisation

- Obtention d'une représentation canonique (représentation mathématique)
- Réécriture = transformation par
  - Simplification
    - Ex. `SELECT * FROM X WHERE 1=1;`
  - Ordonnancement des opérations élémentaires
    - Ex. `SELECT * FROM X,Y WHERE X.a=3;`
      - Produit cartésien → sélection ou
      - Sélection → produit cartésien

Indépendant des données

- Construction des plans d'exécution candidats
  - Choix des algorithmes pour chaque opérateur
  - Calcul du coût de chaque plan
  - Choix du meilleur plan

Dépendant des données
- Il y a des index
- Volume de données
- …

Deux grandes classes de techniques pour optimiser

- Optimisation algébrique
  - Algèbre relationnelle
- Sélection optimale des chemins d'accès
  - Utilisation des index

Rappel: les opérateurs algébriques principaux

- **Sélection R1 / C** (notée σ)
  - Un sous-ensemble de lignes d'une relation R1 vérifiant la condition C
- **Projection R1[VA, ...]** (notée $\prod$)
  - Un sous-ensemble de colonnes (VA, ...) d'une relation R1
- **Produit cartésien R1 X R2** (noté $\times$)
  - Combinaison de deux relations
- **Jointure R1 * (VA = VB) R2** (notée $\bowtie$)
  - Combinaison de deux relations R1 et R2 dont la valeur d'un attribut est la même

Optimisation algébrique

- Remarque
  - Suivant l'ordre des opérateurs algébriques, le coût d'exécution est différent
    - Ex. sélection puis jointure
- Pourquoi
  - Le coût des opérateurs varie en fonction du volume des données traitées : plus le nb de n-uplets des relations traitées est petit, plus les coûts CPU et d'E/S sont minimisés
  - Certains opérateurs diminuent le volume des données : sélection et projection
    - **Avancer ces opérateurs**

Exemple

```
EMPLOYE (NSS, Nom, Prenom, DDN, Adresse, NumDep)
DEPARTEMENT (Dnuméro, Dnom, #NSSChef)
PROJET (Pnum, Pnom, Pville, #Dnum)
TRAVAILLE-SUR (#ENSS, #PNO, Heures)
```

```
SELECT      Nom
FROM        EMPLOYE, TRAVAILLE-SUR, PROJET
WHERE       Pnom = "Esprit" AND Pnum = PNO
            AND ENSS = NSS AND DDN > "30/01/70"
```

Optimisation algébrique

- Une requête SQL est une combinaison des **opérateurs algébriques**

```
EMPLOYE (NSS, Nom, Prenom, DDN, Adresse, NumDep)
PROJET (Pnum, Pnom, PVille, #Dnum)
TRAVAILLE-SUR (#ENSS, #PNO, Heures)
```

Projection
Filtrer les colonnes

Produit cartésien
Entre deux relations

Sélection
Filtrer les n-uplets

```
SELECT      Nom
FROM        EMPLOYE, TRAVAILLE-SUR, PROJET
WHERE       ENSS = NSS AND Pnum = PNO
            Pnom = 'Esprit' AND DDN > '30/01/70';
```


Optimisation algébrique

- Une requête SQL est une collection de **blocs**

```
SELECT nom, prenom
FROM Employe
WHERE numdep IN
    (SELECT Dnumero
    FROM  Departement
    WHERE NSSChef = '1234567890') ;
```

Bloc externe

Bloc imbriqué

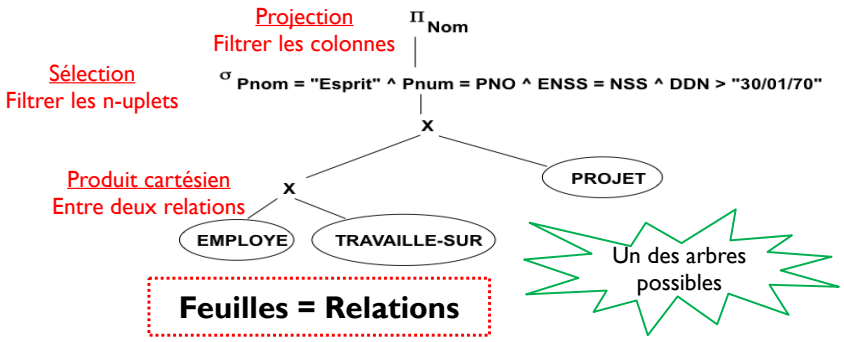
Optimisation algébrique

- Pour chaque bloc, l'optimiseur génère un ensemble de plans
- Plan = Programme combinant des méthodes d'évaluation des opérateurs algébriques
- Représentation d'un plan = Arbre

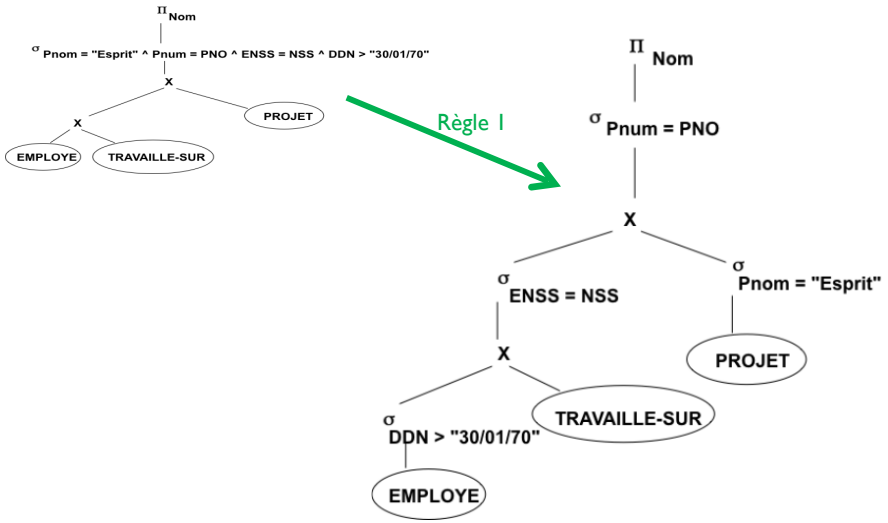
Plan ⇨ algorithmes + ordre d'exécution

Exemple d'arbre relationnel

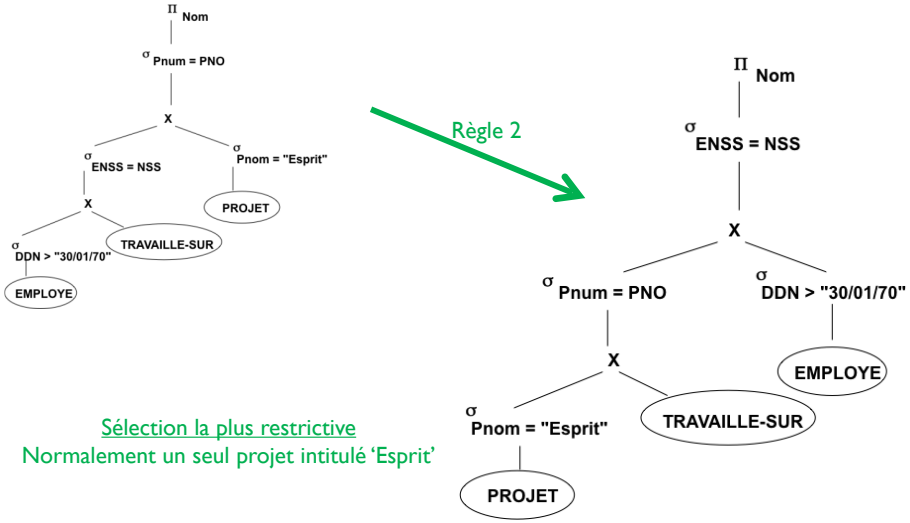
```
SELECT      Nom
FROM        EMPLOYE, TRAVAILLE-SUR, PROJET
WHERE       ENSS = NSS AND Pnum = PNO
            Pnom = 'Esprit' AND DDN > '30/01/70';
```



Règle 1 : les sélections d'abord

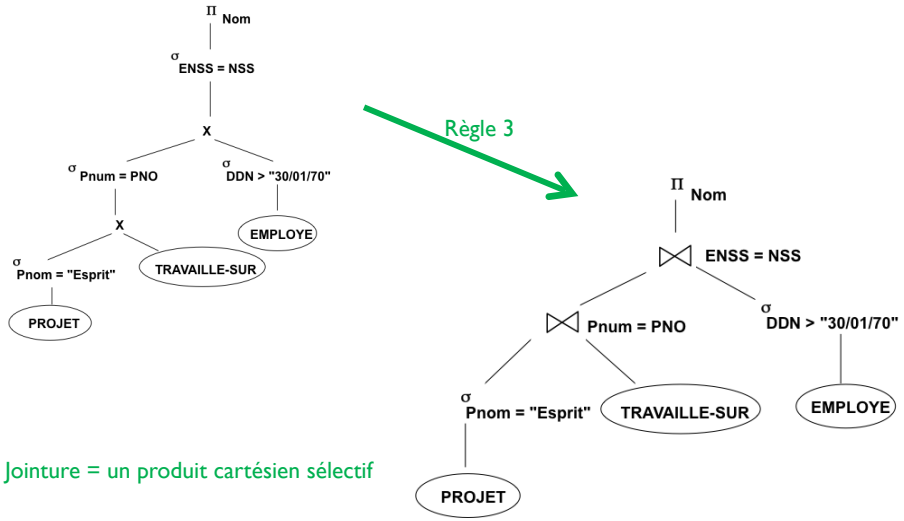


Règle 2 : Commencer par les sélections les plus restrictives



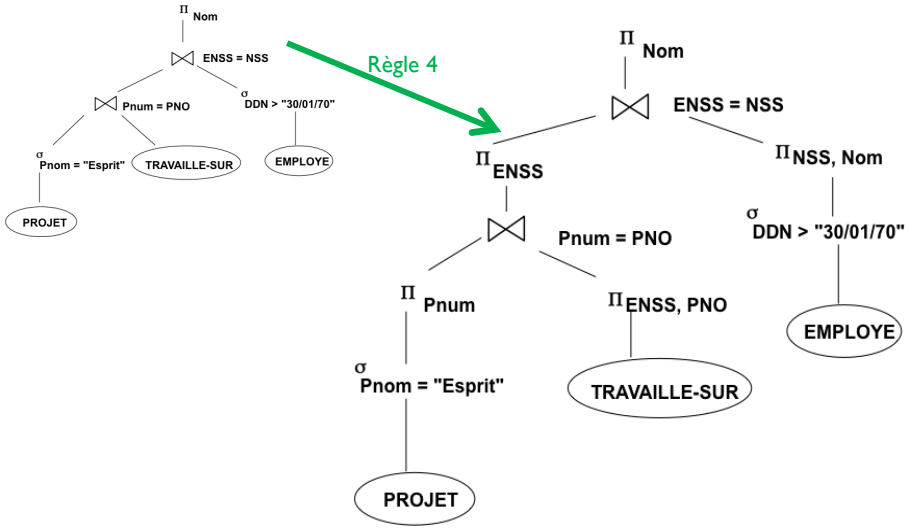
Sélection la plus restrictive
Normalement un seul projet intitulé 'Esprit'

Règle 3 : Préférer les jointures (et non les produits cartésiens)



Jointure = un produit cartésien sélectif

Règle 4 : Diminuer le nombre des constituants



TD

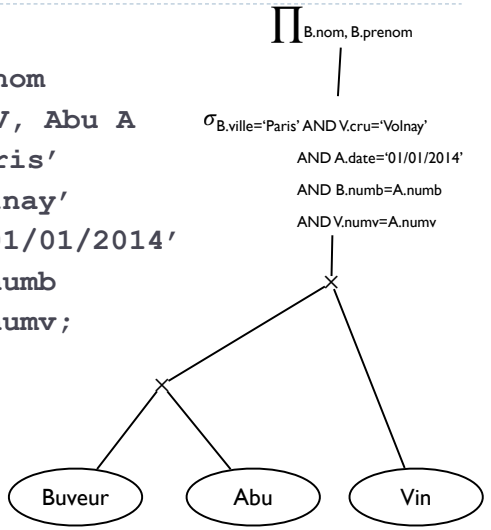
- Soit la base de données des buveurs de vins
  - BUVEUR(numb, nom, prenom, ville)
  - VIN(numv, cru, millesime, degré)
  - ABU(#numb, #numv, date)

- Donner l'arbre relationnel optimisé pour la requête suivante
  - **Noms et prénoms des buveurs parisiens qui ont bu un vin de cru 'Volnay' depuis le premier janvier 2014.**

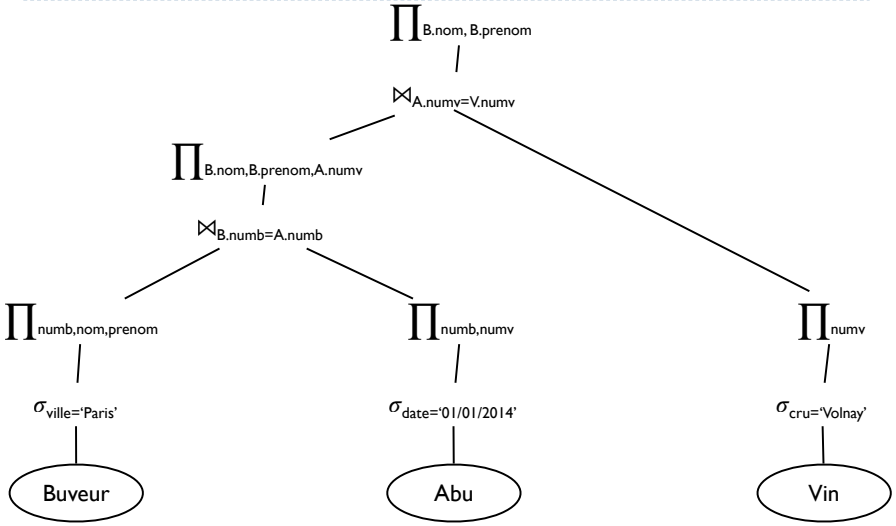
TD – correction 1

- La requête

```
SELECT B.nom, B.prenom
FROM Buveur B, Vin V, Abu A
WHERE B.ville = 'Paris'
    AND V.cru = 'Volnay'
    AND A.date >= '01/01/2014'
    AND B.numb = A.numb
    AND V.numv = A.numv;
```



TD – correction 2



Exemple de BD : les villes, départements et régions de France

- **TOWNS (id, code, article, name, department)**
- **REGIONS (id, code, capital, name)**
- **DEPARTMENT (id, code, capital, region, name)**

- Volume de données
  - 36684 TOWNS
  - 26 REGIONS
  - 100 DEPARTMENTS

Plan d'exécution en Postgres

- Pour voir le plan d'exécution choisi par Postgres
  - lors de l'exécution d'une requête : **EXPLAIN**

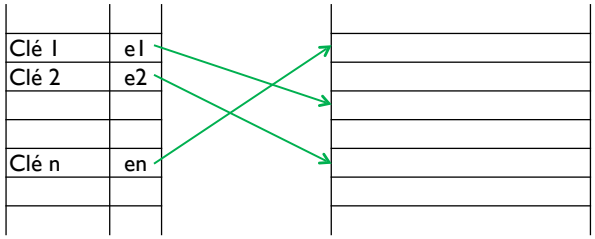
```
bases3a00=> select * from towns where name = 'Grenoble';
  id   | code | article |   name   | department
-------+------+---------+----------+------------
 14859 | 185  |         | Grenoble | 38
(1 row)

bases3a00=> explain select * from towns where name = 'Grenoble';
                        QUERY PLAN
-----------------------------------------------------------
 Seq Scan on towns  (cost=0.00..698.55 rows=1 width=54)
   Filter: (name = 'Grenoble'::text)
(2 rows)
```

  - Parcours séquentiel
  - Cost = (coût pour obtenir le 1er n-uplet, coût pour obtenir le dernier n-uplet)
  - Rows = nb de n-uplets prévus dans le résultat
  - Width = largeur du résultat (en bytes)

Sélection optimale des chemins d'accès

- Chemin d'accès sur une relation
  - Balayage sur toute la relation (séquentiel)
  - Index sur un ou plusieurs attributs
- Index = Fichier de clés pour faciliter l'accès à un autre fichier



Index

- Pour améliorer les performances lors des recherches
- Création d'index
  - `CREATE [UNIQUE] INDEX nom_index ON nom_relation (liste_attributs);`
- Suppression d'index
  - `DROP INDEX nom_index;`
- Par défaut, Postgres crée un index sur la clé primaire
- Attention à bien choisir les (autres) index
  - Surcharge l'insertion et la suppression des données

Plan d'exécution avec index

- Sans index

```
bases3a00=> select * from towns where name = 'Grenoble';
  id   | code | article |   name   | department
-------+------+---------+----------+------------
 14859 | 185  |         | Grenoble | 38
(1 row)

bases3a00=> explain select * from towns where name = 'Grenoble';
                        QUERY PLAN
-----------------------------------------------------------
 Seq Scan on towns  (cost=0.00..698.55 rows=1 width=54)
   Filter: (name = 'Grenoble'::text)
(2 rows)
```

- Recherche à partir de l'index
- Plus rapide !

- Avec index

```
bases3a00=> create index i_name on towns(name);
CREATE INDEX
bases3a00=> explain select * from towns where name = 'Grenoble';
                        QUERY PLAN
-----------------------------------------------------------
 Index Scan using i_name on towns  (cost=0.00..8.27 rows=1 width=54)
   Index Cond: (name = 'Grenoble'::text)
(2 rows)

bases3a00=>
```