Administration des bases de données

Rappels

Rappels

Modèle relationnel et SQL

Objet d'étude

- **Base de Données (BD)**
  - Ensemble structuré de données modélisant un univers réel pour enregistrer des faits, des opérations, …

- **Système de Gestion de Bases de Données (SGBD)**
  - **Data Base Management System** (DBMS)
  - Un système qui permet de gérer une BD partagée par plusieurs utilisateurs simultanément

Que doit permettre un SGBD ? (1/4)

- **Décrire des données**
  - **Indépendamment** de toute application particulière
  - Un SGBD doit intégrer un Langage de Définition des Données ou **Data Definition Language** (DDL)
- **Manipuler des données**
  - Afin d'interroger et mettre à jour les données
    - *Sans avoir à préciser le mode d'accès aux données*
  - Un SGBD doit intégrer un **langage de requêtes déclaratif** (pour dire ce que l'on cherche, sans préciser comment)
    - *Ex : quels sont les produits de prix inférieur à 100€ ?*
  - On parle également de langage de manipulation de données ou **Data Manipulation Language** (DML)

Que doit permettre un SGBD ? (2/4)

- **Contrôler les données**
  - En assurant la vérification de leur **intégrité** (qualité)
    - *En particulier, en permettant de spécifier des contraintes d'intégrité*
    - *Exemple : tout salaire d'un employé de l'université est compris entre 10 K€ et 80 K€*
  - En assurant la **confidentialité** des données stockées
    - *En particulier, grâce à un contrôle des droits d'accès*
  - Un SGBD doit intégrer un langage de contrôle des données ou **Data Control Language** (DCL)

Que doit permettre un SGBD ? (3/4)

- **Optimiser l'accès aux données**
  - Au travers de **techniques d'indexation** telles que le hachage, les arbres équilibrés (CM2)
- **Sécuriser les données**
  - En cas de pannes, en offrant des mécanismes **de reprise après panne** (CM3)
- **Partager des données**
  - Une BD est généralement partagée entre utilisateurs accédant **simultanément** aux données
  - Un SGBD doit ainsi contrôler les **accès concurrents** aux données (via la notion de transaction - CM4)

Que doit permettre un SGBD ? (4/4)

- **Assurer une indépendance physique**
  - Il doit être possible de **modifier les structures de stockage et d'indexation sans** que cela ait de **répercussions** au niveau des applications
    - *Pour un utilisateur, les méthodes de codage, de placement et d'accès aux données ne sont pas apparentes*
- **Assurer une indépendance logique**
  - Différentes applications doivent pouvoir avoir **des vues différentes des mêmes données**
    - *Pour un administrateur de BD, cela peut lui permettre de modifier l'organisation logique des données sans que cela ait de répercussions au niveau des applications (en utilisant des vues, à redéfinir par l'administrateur)*

Modèle relationnel

Modèle de données et
contraintes d'intégrité

Modèle de données

- Tout SGBD est conçu autour d'un **modèle de données** combinant 3 éléments
  - **Une structure de données** qui définit comment les données sont organisées (au niveau logique)
  - **Des langages d'interrogation** et **de manipulation** des données : pour chercher les donnée ou les mettre à jour
  - **Des contraintes d'intégrité** : permettent de maintenir l'intégrité / la cohérence / la qualité des données

Différents types de modèles

- **Modèle hiérarchique** (ex : XML)
  - **Structure** : une BD est une **forêt d'arbres**
  - **Langages** : XPATH, XQUERY
- **Modèle graphe** (ex : RDF)
  - **Structure :** une BD est **un graphe orienté et étiqueté** représenté par un ensemble de triplets (s,p,o)
  - **Langages** : SPARQL
- **Modèle relationnel**
  - **Structure** : une BD est un ensemble de relations, qui sont des ensembles de tuples
  - **Langages** : algèbre relationnelle, calcul relationnel, SQL

Modèle relationnel

- **Notions fondamentales**
  - Attribut et domaine
  - Relation et tuple
  - Schéma et instance
  - Contraintes d'intégrité
  - Dépendance fonctionnelle et clef primaire
  - Dépendance d'inclusion et clef étrangère

Attribut et domaine

- **Définitions**
  - Soit $\mathcal{U}$ un ensemble de noms d'attributs, appelé **univers**
  - Chaque attribut A ∈ $\mathcal{U}$ prend ses valeurs dans un **domaine**, noté dom(A)
- **Exemples**
  - dom(Note) = [0,20]
  - dom(Salle) = { 112, 202, 345, … }
  - dom(Cours) = { 'Graphe', 'Java', 'Réseau', … }

Relation et tuple (1)

- **Définitions**
  - Soit une relation (de nom) R
  - Le **schéma d'une relation** R, noté sch(R), est un sous-ensemble d'attributs (sch(R) $\subseteq \mathcal{U}$)
  - Un **tuple t sur R** est une fonction t: sch(R) → dom($\mathcal{U}$)
  - Une **instance de relation r** sur R est un ensemble de tuples sur R

Relation et tuple (2)

- **Exemples**
  - Soit une relation **Film** de schéma { Titre, Réalisateur }
  - I(Film) = { f1, f2 } est une instance de Film comportant deux tuples f1 et f2 définis par :
    - *f1(Titre) = 'The Cameraman' et f1(Réalisateur) = 'Buster Keaton'*
    - *f2(Titre) = 'To Be or Not to Be' et f2(Réalisateur) = 'Lubitsch'*
    - *On écrit parfois plus simplement f1 = ('The Cameraman', 'Buster Keaton') et f2 = ('To Be or Not to Be', 'Lubitsch')*
  - Une instance est souvent représentée par une table

Film	Titre	Réalisateur
	The Cameraman	Buster Keaton
	To Be or Not to Be	Lubitsch

Bases de données (1)

- **Définitions**
  - Un **schéma de base de données** relationnelle est défini par un ensemble de relations et de leurs schémas
  - Une **instance de base de données** est définie par un ensemble d'instances de relations
- **Exemples**
  - Soit la base de donnée **Cinéma** de schéma { **Film**(Titre, Réalisateur), **Acteur**(Nom,Titre) }
  - Une instance I de la base **Cinéma** est composée de deux instances **I(Film)** et **I(Acteur)** des relations Film et Acteur

Bases de données (2)

- **Suite exemple**
  - Une instance de Cinéma représentée par 2 tables

Film	Titre	Réalisateur
	The Cameraman	Buster Keaton
	To Be or Not to Be	Lubitsch

Acteur	Nom	Titre
	Buster Keaton	The Cameraman
	Carole Lombard	To Be or Not to Be
	Jack Benny	To Be or Not to Be

Bases de données (3)

- **Suite exemple :** en SQL et avec 3 tables
  - CREATE TABLE **Film** (
    IdFilm NUMBER(5),
    Titre VARCHAR(255),
    Realisateur VARCHAR (100));
  - CREATE TABLE **Acteur** (
    IdActeur NUMBER(5),
    Nom VARCHAR(255));
  - CREATE TABLE **Role** (
    IdActeur NUMBER(5),
    IdFilm NUMBER(5),
    Principal BOOLEAN);

Contraintes d'intégrités

- **Objectif**
  - Donner un cadre pour ajouter une sémantique au modèle relationnel
- **Définition**
  - **Une contrainte d'intégrité** est une propriété d'une base de données supposée satisfaite sur toutes ces instances
- **Exemples**
  - Tout film est associé à un seul directeur / réalisateur
  - Tout acteur ne peut être renseigné pour un film que si le réalisateur de ce même film est également renseigné

Dépendance Fonctionnelle (DF)

- **Définition**
  - Etant donnée une instance de relation r sur R, **une DF notée R : X → Y, est satisfaite sur l'instance r** ssi

$$(\forall t_1, t_2 \in r)(t_1[X] = t_2[X] \Longrightarrow t_1[Y] = t_2[Y])$$

  - On dit que X détermine (fonctionnellement) Y dans r
  - **Quel sens intuitif ?**
- **Exemple**
  - Soit la DF notée **Film : Titre → Réalisateur**
  - **Quelle sémantique ?**

Clef et clef primaire (1)

- **Définitions**
  - Une **clef d'une relation R** est un sous-ensembles d'attributs X de R déterminant tous les autres attributs de R (Y = sch(R) \ X) quel que soit son instance
  - Une **clef primaire d'une relation** est simplement une clef parmi toutes les clefs possibles, choisie par le concepteur pour sa simplicité
- **Notation**
  - La clef primaire d'une relation est souvent soulignée
  - Exemple : **Film**(Titre, Réalisateur)

Clef et clef primaire (2)

- **Suite exemple :** en SQL
  - CREATE TABLE **Film** (
    IdFilm NUMBER(5) **PRIMARY KEY**,
    Titre VARCHAR(255),
    Realisateur VARCHAR (100));
  - CREATE TABLE **Acteur** (
    IdActeur NUMBER(5) **PRIMARY KEY**,
    Nom VARCHAR(255));
  - CREATE TABLE **Role** (
    IdActeur NUMBER(5),
    IdFilm NUMBER(5),
    Principal BOOLEAN,
    PRIMARY KEY(IdActeur, IdFilm));

Dépendance d'Inclusion (DI)

- **Définition**
  - Etant donnée une base de données « d » comportant deux instances de relation r_1 et r_2 définies sur R_1 et R_2
  - **Une DI notée $R_1[X] \subseteq R_2[X]$, est satisfaite sur « d »** ssi

$$(\forall t_1 \in r_1)(\exists t_2 \in r_2)(t_1[X] = t_2[X])$$

  - **Quel sens intuitif ?**
- **Exemple**
  - Role[IdFilm] ⊆ Film[IdFilm]
  - Role[IdActeur] ⊆ Acteur[IdActeur]
  - **Quelle sémantique ?**

Clef étrangère (1)

- **Définition**
  - Une **contrainte d'intégrité référentielle** est **une DI** : $R_1[X] \subseteq R_2[X]$ **dont la partie droite est une clé**
    - *X est une clef de R_2*
    - *C'est un cas particulier de dépendance d'inclusion*
  - **Une clef étrangère** est la partie gauche d'une contrainte d'intégrité référentielle
    - *X est une clef étrangère de R_1*
- **Exemple**
  - Si **Role[IdFilm] ⊆ Film[IdFilm]** et **IdFilm est une clef de la relation Film**
  - Alors **IdFilm** est une **clef étrangère de Role**

Clef étrangère (2)

- **Suite exemple :** en SQL
  - CREATE TABLE **Film** (
    IdFilm NUMBER(5) **PRIMARY KEY**,
    Titre VARCHAR(255),
    Realisateur VARCHAR (100));

  - CREATE TABLE **Acteur** (
    IdActeur NUMBER(5) **PRIMARY KEY**,
    Nom VARCHAR(255));

  - CREATE TABLE **Role** (
    IdActeur NUMBER(5),
    IdFilm NUMBER(5),
    Principal BOOLEAN,
    PRIMARY KEY(IdActeur, IdFilm),
    FOREIGN KEY (IdActeur) **REFERENCES** Acteur(IdActeur),
    FOREIGN KEY (IdFilm) **REFERENCES** Film(IdFilm));

Langages du modèle relationnel

Algèbre relationnelle et SQL

Algèbre relationnelle (1)

- **Opérations ensemblistes** (binaires)
  - **Intersection** : $R_1 \cap R_2$
    - *Ensemble des tuples appartenant à la fois à R_1* ***et*** *R_2*
  - **Union** : $R_1 \cup R_2$
    - *Ensemble des tuples appartenant à R_1* ***ou*** *à R_2*
  - **Différence** : $R_1 \setminus R_2$
    - *Ensemble des tuples appartenant à R_1* ***mais pas à*** *R_2*

Algèbre relationnelle (2)

- **Autres opérations** (unaires)
  - **Sélection** : $\sigma_C(R)$
    - *Sélection des tuples appartenant de R satisfaisant la condition C*
      - Par exemple (A = a) où A ∈ sch(R) et a ∈ dom(A)
  - **Projection** : $\pi_X(R)$
    - *Supprime de R les colonnes (attributs) n'appartenant pas à X*
  - **Renommage :** $\rho_{[X/X']}(R)$
    - *L'attribut X de R est remplacé (syntaxiquement) par X'*

Algèbre relationnelle (3)

- **Autres opérations** (binaires)
  - **Produit cartésien** : $R_1 \times R_2$
    - *Regroupe tous les tuples $t_1 \circ t_2$ où t_1 est un tuple de R_1, t_2 un tuple de R_2 et $\circ$ une opération de concaténation*
  - **Jointure** : $R_1 \bowtie_C R_2 = \sigma_C(R_1 \times R_2)$
    - *Permet de composer deux relations comme le produit cartésien de deux relations, mais avec une condition*

Exemple de jointure

Achat

IdC	IdP	Date	Qte
1	1	1/1/2017	2
2	2	2/6/2017	4
3	1	1/5/2017	1

Produit

IdP	NomP	Prix
1	PC	800
2	MAC	1000

Achat $\bowtie_{IdP=IdP'}$ $\rho_{[IdP/IdP']}$**(Produit)**

IdC	IdP	Date	Qte	IdP'	NomP	Pric
1	1	1/1/2017	2	1	PC	800
2	2	2/6/2017	4	2	MAC	1000
3	1	1/5/2017	1	1	PC	800

Algèbre relationnelle (4)

- **Autres opérations** (binaires)
  - **Division** : $R_1 \div R_2$
    - *Suppose que $sch(R_2) \subseteq sch(R_1)$*
    - *Retourne une relation de schéma $sch(R_1) \setminus sch(R_2)$*
    - *Un tuple $(a_1, \ldots, a_n)$ appartient à $R_1 \div R_2$ si :*
      - $(a_1,\ldots a_n,b_1,\ldots,b_p)$ appartient à R_1
      - Pour **tous** les tuples $(b_1,\ldots,b_p)$ de R_2

Exemple de division

Inscription

Etudiant	NomUE
Ivan	Foot
Eva	Natation
Ivan	Judo
Eva	Judo
Ivan	BD

Sport

NomUE
Foot
Judo

Inscription ÷ Sport

(Les étudiants inscrits à toutes les UE de sport)

Etudiant
Ivan

Langage SQL

- **Structured Query Langage**
  - Une implémentation de l'algèbre relationnelle **mais avec quelques différences**
    - *Pas de distinction entre schéma et instance de relation, les deux étant désigné par la* ***notion de table***
    - ***Une table SQL*** *n'est pas un ensemble de tuples, mais un* ***multi-ensembles de tuples***
      - Principalement pour des raisons d'efficacité, la détection de doublons pouvant nécessiter des tris couteux
      - **{ 1, 2, 3, 2 } est un exemple de multi-ensemble** (ou bag en anglais)

Syntaxe d'une requête SQL simple

- **A la base, une requête SQL comporte trois clauses**

```
SELECT    < liste d'attributs >
FROM      < liste de relations >
[WHERE    < conditions sur les lignes >]
```

  - La clause **SELECT** permet de coder **la projection**, le résultat de la requête étant une table de schéma défini par la clause SELECT
  - La clause **WHERE** permet de coder **la sélection** [en étant optionnelle]

Ensemble versus muli-ensembles

- **Des passerelles existent**
  - Grâce aux mots clefs DISTINCT et ALL
    - ***DISTINCT*** *(après un SELECT) : permet de passer à un ensemble en supprimant les doublons*
    - ***ALL*** *(après un UNION) : peut forcer la sémantique multi-ensembles, les opérateurs ensemblistes (UNION, INTERSECT et MINUS)*
  - **Il faut donc être prudent** et se méfier des coûts engendrés par un DISTINCT (nécessitant une opération cachée de tri)

```
SELECT   DISTINCT nom, prenom
FROM     Personne
WHERE    age = 30;
```

Renommage en SQL

- **A la fois possible**
  - Sur les attributs de la clause SELECT
  - Sur les relations de la clause FROM

```
SELECT   DISTINCT nom [AS] identifiant
FROM     Personne [AS] P
WHERE    P.age = 30;
```

  - **AS** est optionnel !
  - En cas d'ambiguïté, on peut préfixer un nom d'attribut par le nom de sa relation (ex : Personne.age)

Expressions et chaînes de caractères

- **Expressions arithmétiques**
  - Peuvent être utilisées sur les valeurs retournées (dans la clause SELECT)
    - *Le caractère '=' pouvant alors être utilisé pour le renommage*
- **String Matching**
  - Peut être effectué avec le mot clef **LIKE** et
  - **Les jokers '%'** (autant de caractères que l'on veut) ou **'_'** (pour un seul caractère quelconque)

```
SELECT   nom, age2 = age*2
FROM     Personne
WHERE    nom LIKE 'A_%a';
```

Quel sens ?

Opérateur ensembliste

- **SQL permet d'exprimer les opérations ensemblistes**
  - En connectant des SELECT par les mots clefs **UNION**, **INTERSECT** ou **MINUS**

```
SELECT   P.IdP
FROM     Produit P
WHERE    P.Prix > 1000
UNION
SELECT   V.IdP
FROM     Vente V
WHERE    V.Qte > 100
```

Quel sens ?

Produit cartésien et jointure

- **Opérateur JOIN**
  - **Produit cartésien** (très dangereux car coûteux)

```
SELECT   *
FROM     Produit P, Vente V;
```

  - **Jointure**

```
SELECT   *
FROM     Produit P, Vente V
WHERE    P.IdP = V.IdP;
```

***Note** : dans la clause SELECT, on peut lister tous les attributs avec le caractère '*'*

Requêtes imbriquées

- **SQL permet l'imbrication de sous-requêtes**
  - Au niveau de la clause WHERE, elles peuvent être utilisées
    - *Dans les prédicats de comparaison (=, >, <, …)*
      - La sous-requête doit alors se réduire à **une seule valeur**
    - *Dans des prédicats **IN, ALL** ou **ANY***
      - La sous-requête doit alors retourner une table **d'une seule colonne**
    - *Dans des prédicats **EXISTS***
      - Sans contrainte sur la sous-requête (dont on teste uniquement si elle est vide ou non)
    - *Sachant que les prédicats **IN** et **EXISTS** peuvent être précédés de **NOT***

Prédicat EXISTS

- **Exemple de requête**
  - Noms de produits qui n'on pas été vendus

```
SELECT   P. nom
FROM     Produit P
WHERE    NOT EXISTS
         ( SELECT  *
           FROM    Vente V
           WHERE   P.IdP = V.IdP);
```

Quel sens si on supprime le NOT ?

Produit(IdP, NomP, Prix)
Vente(IdP, IdC, Date, Qte)
Client(IdC, PrenomC, NomC)

Prédicat IN

- **Exemple de requête**
  - Nom et prénom des clients ayant acheté un MAC

```
SELECT   C.NomC, C.PrenomC
FROM     Client C
WHERE    C.IdC IN
         ( SELECT V2.IdC
           FROM   Produit P, Vente V2
           WHERE  P.IdP = V2.IdP AND
                  P.NomP = 'MAC');
```

Quel sens si on ajoute un NOT devant le IN ?

Produit(IdP, NomP, Prix)
Vente(IdP, IdC, Date, Qte)
Client(IdC, PrenomC, NomC)

Prédicats ALL et ANY

- **Exemple de requête**
  - Identifiants des clients ayant acheté **un** produit en quantité supérieure **à chacune des** (ou **à au moins l'une des**) quantités de produits achetés par le client 1

```
SELECT   V. IdC
FROM     Vente V
WHERE    V.Qte > ALL ( ou ANY)
         (  SELECT  W.Qte
            FROM    Vente W
            WHERE   W.IdC = 1);
```

Produit(IdP, NomP, Prix)
Vente(IdP, IdC, Date, Qte)
Client(IdC, PrenomC, NomC)

Fonctions d'agrégats

- **Une extension de l'algèbre relationnelle**
  - Elles opèrent sur un ensemble de valeur et les agrègent
  - **SUM**, **AVG**, **STDDEV** : pour calculer les sommes, moyennes et écart types d'un ensemble de valeurs
  - **MIN**, **MAX** : pour calculer les valeurs minimales ou maximales d'un ensemble de valeurs
  - **COUNT** : pour compter un nombre de valeurs
    - ***Attention au rôle possible du DISTINCT***

```
SELECT COUNT(*)
FROM   Vente V;
```

≠

```
SELECT COUNT(DISTINCT V.PiD)
FROM   Vente V;
```

Nombre total de ventes effectuées

Nombre total de produits ayant été vendus

Opération de groupements

- **La clause GROUP BY**
  - Permet de partitionner une table en plusieurs groupes
  - Toutes les tuples d'un même groupe ont la même valeur pour tout les attributs spécifiés après le GROUP BY

```
SELECT      < attributs cibles >
FROM        < liste de relations >
[WHERE      < conditions sur les lignes >]
GROUP BY    < attributs de groupement >
HAVING      < conditions sur les groupes >
```

  - **Important** : les attributs cibles sont
    - *Soit des attributs présents dans les attributs de groupement*
    - *Soit des fonctions d'agrégations*

Clause GROUP BY

- **Sans restriction sur les groupes**
  - Quantité de produits vendus par produit de plus de 1000€

```
SELECT    V.Pid, SUM(V.Qte)
FROM      Vente V, Produit P
WHERE     V.Pid = P.Pid AND P.Prix > 1000
GROUP BY  V.Pid;
```

- **Avec restriction sur les groupes**
  - Quantité de produits vendus par produit pour les produits toujours vendus par paquet de 10 ou plus

```
SELECT    V.Pid, AVG(V.Qte)
FROM      Vente V, Produit P
WHERE     V.Pid = P.Pid
GROUP BY  V.pid
HAVING    MIN(V.Qte) > 10;
```

Produit(IdP, NomP, Prix)
Vente(IdP, IdC, Date, Qte)
Client(IdC, PrenomC, NomC)

Opération de tri

- **Clause ORDER BY**
  - Liste des produits par ordre alphabétique de plus de 1000€

```
SELECT    *
FROM      Produit P
WHERE     P.Prix > 1000
ORDER BY  V.NomP [DESC];
```

  - En ordre croissant [ASC – par défaut] ou décroissant [DESC]

Fin du premier cours

Prochain cours : stockage sur disque et indexation