BD-BioInfo

Bases de données

Cristina Sirangelo
IRIF, Université Paris Diderot
cristina@irif.fr

Infos pratiques

- Cours : Cristina Sirangelo < cristina@irif.fr>
  - Jeudi 14h -16h, 418C HaF, 5 séances (cf. emploi du temps)

- TD/TP : Sylvain Perifel <sylvain.perifel@irif.fr>
  - En général : un TD ou TP par semaine 12 seances - cf. emploi du temps

    L3 : http://biteach.sdv.univ-paris-diderot.fr/lbi/

    (ATTENTION : celui du M1 :

    http://biteach.sdv.univ-paris-diderot.fr/m1bi/etudiant.html#edt

    n'est pas à jour !)

Infos pratiques - suite

- Contrôle des connaissances :
  - un projet en binôme
  - examen final
- Mode de calcul de la note finale:
  - session 1 : 35% projet + 65% examen
  - session 2 : 100% examen
- Page du cours sur Moodle : **s'inscrire!**
  - https://moodlesupd.script.univ-paris-diderot.fr/course/view.php?id=10811
    - ou chercher :"Bases de données - L3 et M1 BioInfo"
  - annonces, slides du cours, sujets des TD/TP, projet, soumission des projets
- Plus de détails en cours de route…

Plan du cours

- Introduction
- Modèle relationnel des données, contraintes d'intégrité, SQL DDL
- Modélisation conceptuelle des données
  - modèle E/R et passage au modele relationnel
- Manipulation des données en SQL
- Interrogation des données en SQL
  - requêtes simples
  - jointure
  - regroupement et agrégation
  - requêtes imbriquées
  - opérateurs ensemblistes
  - compléments : types de jointure, information incomplète, vues

Textes conseillés pour ce cours et bibliographie

- *Database Systems Concepts*
  par Silberschatz, Korth and Sudarshan, 6eme edition, McGraw-Hill.
- *Database Systems: the Complete Book*
  par H. Garcia-Molina, J.Ullman and J. Widom, Prentice Hall.
- *Database Management Systems*
  par Raghu Ramakrishnan and Johannes Gehrke. McGraw-Hill.

- Un texte en français
  Bases de données et modèles de calcul
  par Jean-Luc Hainaut

- Pour aller plus loin en théorie des bases de données :
  Foundations of Databases
  par S.Abiteboul, R.Hull and V.Vianu, Addison-Wesley, 1995.

Introduction

Bases de données

Toutes les applications qui offrent un service manipulent des données persistantes

aujourd'hui surtout sur le Web :

- Site de vente en ligne : descriptions et prix des articles vendus, le transactions de vente, …
- Site d'une compagnie aérienne : quels vols à quels horaires, ..
- Espace utilisateur de n'importe quel site : informations d'enregistrement des utilisateurs, …
- Réseaux sociaux : les profils des membres, les posts, qui suit qui, qui aime quoi…

mais également dans le contexte scientifique : physique, biologie, astronomie etc.

Toutes ces données doivent être stockées de façon durable chez le serveur,

- pas uniquement pendant une session utilisateur

- pas uniquement pendant que le serveur est actif - elles doivent continuer à exister même si le serveur est arrêté

Ces données doivent donc exister ailleurs que dans la mémoire centrale du serveur (et elle ne serait pas suffisante de toute façon!)

La gestion des données persistantes

Un problème vieux comme l'informatique

Pour gérer une liste (par exemple de films),

pourrait-on utiliser un fichier texte?

```
> cat films.txt
Alien 1979 Scott
Vertigo 1958 Hitchcock
Psychose 1960 Hitchcock
Kagemusha 1980 Kurosawa
Volte-face 1997 Woo
```

La gestion des données persistantes

Ou une structure de données dans un langage de programmation sérialisée sur disque?

- Définir une classe d'objets qu'on va mettre dans un tableau.

```
class Film {
   public $titre,
          $année,
          $réalisateur,
}
```

- Créer un tableau d'objets films, le sérialiser dans un fichier sur le disque

Les problèmes des systèmes à base de fichiers

Accès difficile aux données

Ouvrir le fichier

Parcourir les “lignes”

Effectuer les comparaisons

Format complexe (tenir compte des espaces…)

Format de fichiers non standards ⇒ partage des données difficile

Ecriture d’un programme spécifique pour chaque interrogation/modification des données : coûts de développement élevés, coûts de maintenance élevés

Intégrité logique des données :

Comment garantir que les données restent cohérentes
(par exemple éviter :
`Vertigo 1958 Hitchcock`
`Vertigo 1962 Hitchcock`)

Les problèmes des systèmes à base de fichiers

Concurrence

Que se passe-t-il si plusieurs utilisateurs ajoutent, modifient ou suppriment des lignes simultanément ?

Gestion des pannes

Comment faire en sorte que les données ne soient pas laissées dans un état incohérent si le serveur va en panne pendant qu'il les manipule?
À programmer soi même (l'OS ne suffit pas)

Sécurité

Comment empêcher un programme erroné ou malveillant d'écrire des données autres (e.g. écraser le contenu du fichier) ?

etc.

SGBD

Les Systèmes de Gestion de Bases de Données ont été conçus pour apporter des (bonnes) réponses à ces problèmes

- Manipuler informations complexes de façon abstraite
  - sans se préoccuper de comment elle sont physiquement organisées sur disque (indépendance physique)
- Optimiser et rendre efficace l'accès aux données
- Garantir l'intégrité des données et la tolérance aux pannes
- Assurer un résultat cohérent quand plusieurs utilisateurs accèdent simultanément aux données

SGBD relationnels

Représentation de l'information sous forme de tables

La manière dont l'information est réellement stockée sur disque est inconnue de l'application

L'application ne voit que les "tables" présentées par le SGBD

Langage "standard" pour l'interrogation/manipulation de ces tables :
SQL (norme ANSI de 1992) - *Structured Query Language*

Il existe de très nombreux systèmes de gestion de bases de données relationnelles

Oracle, PostgreSQL, SQL Server, DB2, MySQL…

Un aperçu du modèle relationnel des données

Modèle relationnel des données

- Une base de données consiste en plusieurs tables (relations)
- Chaque table a un nom

- Dans une table chaque colonne a un nom
- Les noms des colonnes d'une table sont appelés attributs (ou champs)
- Chaque attribut a un domaine associé (i.e. l'ensemble des valeurs possibles pour cet attribut)

- Les données dans chaque table sont un ensemble de lignes (tuples)
  - une ligne fournit à chaque attribut une valeur de son domaine

Une table

Nom de la table

Attributs

Lignes

STUDENT	Name	SSN	HomePhone	Address	OfficePhone	Age	GPA
	Benjamin Bayer	305-61-2435	373-1616	2918 Bluebonnet Lane	null	19	3.21
	Katherine Ashly	381-62-1245	375-4409	125 Kirby Road	null	18	2.89
	Dick Davidson	422-11-2320	null	3452 Elgin Road	749-1253	25	3.53
	Charles Cooper	489-22-1100	376-9821	265 Lark Lane	749-6492	28	3.93
	Barbara Benson	533-69-1238	839-8461	7384 Fontana Lane	null	19	3.25

Exemple : les films sous forme de table

Films

titre	*annee*	*realisateur*
Alien	1979	Scott
Vertigo	1958	Hitchcock
Psychose	1960	Hitchcock
Kagemusha	1980	Kurosawa
Volte-face	1997	Woo
Pulp Fiction	1995	Tarantino
Titanic	1997	Cameron
Sacrifice	1986	Tarkovski

Schéma et instance d'une table

- La structure des données est rigide (toutes les lignes ont les mêmes attributs)
  - Cette structure est appelé schéma de la table :

STUDENT

Name	SSN	Phone	Adress	OfficePhone	Age	GPA

  - Aussi dénoté Student (Name, SSN, Phone, Adress, OfficePhone, Age, GPA)
  - L'ensemble des lignes est appelé instance de la table

- Le schéma est défini une fois pour toute
- L'instance varie dans le temps selon les données qu'on insère et on supprime

- Des contraintes d'intégrité sont en général associées à chaque table, qui empêchent les instances non valides (valeurs requis, uniques, etc…)

Schéma et instance d'une base de données

- Une bases de données est formée par plusieurs tables

STUDENT

Name	SSN	Phone	Adress	OfficePhone	Age	GPA
Bayer	347294	333	Albyn Pl.	367	18	3.24
Ashly	5784673	466	Queen St.	390	20	3.53
Davidson	4357387	589	Princes St.	678	25	3.25

EXAM

COURSE

Course-Id	Title
12	CS
34	DB

Student-SSN	Course-Id
347294	12
5784673	12
4357387	34
347294	34

- Schéma de la base de donnée : l'ensembles de schémas de toutes ses tables
- Instance de la base de données : l'ensemble des instances de toutes ses tables (i.e les données)

SQL

Langage déclaratif

Permet de manipuler à la fois le schéma et les instances d'une bases de données

Composante DDL (Data Definition Language)

- définir le schéma de la bases de données

Composante DML (Data Manipulation Language)

- manipuler les données (insérer/ supprimer/ mettre à jour)
- interroger (extraire de l'information de) des données de manière déclarative :

```
SELECT titre            -- l'attribut recherché
FROM Films               -- la table interrogée
WHERE annee > 1980;  -- le critère de sélection
```

Quelques éléments de SQL DDL pour la création de schémas de BD

Création d'une base de données

```
CREATE DATABASE ma_base; -- commande SQL
```

Au sein d'un serveur on peut créer plusieurs bases de données différentes

Chaque base à ses propres tables, mais une table appartient à une seule base

On peut associer des droits d'accès aux utilisateurs des bases, et des tables

Création d'une table (schéma)

Au sein de la base de données courante :

```
CREATE TABLE Films
(titre      VARCHAR(30),
 annee      INTEGER,
 realisateur  VARCHAR(30)
);
```

Les types des attributs varient d'un SGBD à l'autre

Cette déclaration contient en générale aussi la définition de plusieurs contraintes d'intégrité (donc on discutera plus loin)

À sa création, une table est "vide" (instance vide, on ne crée que le schéma)

- Elle ne contient aucune donnée, i.e. aucune ligne
- D'autre commandes SQL permettent d'insérer et/ou supprimer des lignes (voir plus loin)

Détruire une table

Détruire la table (schéma et données):

```
DROP TABLE Films;
```

Supprimer toutes les données (mais garder le schéma):

```
TRUNCATE TABLE Films;
```

Contraintes d'intégrité

Contraintes d'intégrité

- Contrainte d'intégrité : propriétés des données que l'on demande au système de garantir
- Exemples :
  - Un attribut doit toujours avoir une valeur
  - Un (ensemble d') attribut(s) identifie les lignes d'une table
  - Un attribut d'une table fait référence à l'identifiant d'une autre table
  - Un attribut ne peut prendre qu'une des valeurs prédéfinies d'un ensemble
  - etc.

Contrainte NOT NULL

Si la valeur d'un attribut n'est pas spécifiée pendant l'insertion, la valeur "vide" NULL lui sera affectée

Films

titre	*annee*	*realisateur*
Alien	1979	Scott
Vertigo	1958	Hitchcock
NULL	1960	Hitchcock
Kagemusha	NULL	Kurosawa
Volte-face	1997	Woo
Pulp Fiction	1995	Tarantino
Titanic	NULL	Cameron
Sacrifice	1986	Tarkovski

La contrainte de NOT NULL sur un attribut d'une table impose que l'attribut ait une valeur non nulle

Contrainte NOT NULL

Spécifier des contraintes de NOT NULL en SQL

```
CREATE TABLE Films
(titre      VARCHAR(30) NOT NULL,
 annee      INTEGER NOT NULL,
 realisateur  VARCHAR(30)
);
```

Effet : génère une erreur à l'insertion d'une ligne qui ne spécifie pas la valeur du titre ou la valeur de l'année

Contraintes de clé

- Une clé d'une table:
  - Plus petit sous-ensemble d'attributs permettant d'identifier une ligne de manière unique
- Exemples :
  - nss est une clef de la table Personne(nss, nom, prénom, naissance, pays)
  - (ville, rue, numero) est une clef de la table Bâtiment (ville, rue, numero, #etages)
- Une table peut avoir plusieurs clefs
- Exemple :
  - pseudo est une clef de la table Utilisateur (pseudo, email, nom, prénom, mdp)
  - comme également email

Clés primaires

- Une table a toujours une clé dite
  - clé primaire

    (attributs soulignés ci-dessous)

    Film (titre, année, realisateur)

    Personne(nss, nom, prénom, naissance, pays)

    Utilisateur (pseudo, e-mail, nom, prénom, mdp)

    Bâtiment (ville, rue, numero, #etages)

  - les autres clés sont appelées clefs candidates (ou secondaires)
    - exemple : email pour la table Utilisateur

Spécifier les clés primaires en SQL

- Clé primaire comportant un seul attribut

```
CREATE TABLE Films
(titre      VARCHAR(30) PRIMARY KEY,
 annee      INTEGER NOT NULL,
 realisateur  VARCHAR(30)
);
```

Spécifier les clés primaires en SQL

- Clé primaire comportant plusieurs attributs

```
CREATE TABLE Bâtiment (
    ville VARCHAR(50),
    rue VARCHAR(100),
    numero INTEGER,
    #etages INTEGER,
    PRIMARY KEY (ville, rue, numero)
);
```

- Remarque : PRIMARY KEY implique NOT NULL, pas besoin de le spécifier explicitement pour les attributs d'une clé
- Effet : la tentative d'insertion d'une clé primaire existante génère une erreur
- Chaque table devrait avoir une clé primaire

Clés candidates

- Les autres clés de la table, pas choisies comme clés primaires, peuvent être spécifiées avec la contrainte UNIQUE

```
CREATE TABLE Personne (
      nss VARCHAR(20) PRIMARY KEY,
      nom VARCHAR(50) NOT NULL,
      prenom VARCHAR(50) NOT NULL,
      naissance INTEGER,
      pays VARCHAR(4),
      UNIQUE (nom, prenom, naissance)
);
```

- (nom, prenom, naissance) : clé candidate
- Effet : la tentative d'insertion d'une clé candidate existante génère une erreur
- Remarque : UNIQUE n'implique pas NOT NULL

Clés étrangères

Contrainte entre deux tables : sert à relier des attributs d'un table avec la clef primaire d'une autre table

`Exemple`

Personne(<u>nss</u>, nom, prénom, naissance, pays)

Pays (<u>code</u>, nom) Personne[pays] ⊆ Pays[code]

Impose que la colonne `pays` de la table Utilisateur contienne uniquement des valeurs qui apparaissent dans la colonne `code` de la table Pays

Clés étrangères

Contrainte entre deux tables : sert à relier des attributs d'un table avec la clef primaire d'une autre table

`Exemple`

Personne(<u>nss</u>, nom, prénom, naissance, pays)

Pays (<u>code</u>, nom) Personne[pays] ⊆ Pays[code]

Personne

nss	*nom*	*preno*	*naissan*	*pay*
SDHF45FR9	Dupont	Luc	1943	FR
ERYE75EFHS	White	Alfred	1899	UK
JFB9745HSD	White	Ted	1910	UK
SDHF8465SJ	Brown	John	1946	US
SF745HSFJ7	Smith	Ross	1987	US
SHDGG475	Brown	James	1954	UK

Pays

cod	*nom*
FR	France
UK	Grande Bretagne
US	Etas Unis

Cette instance de la base de données satisfait la contrainte

Clés étrangères

Contrainte entre deux tables : sert à relier des attributs d'un table avec la clef primaire d'une autre table

`Exemple`

Personne(nss, nom, prénom, naissance, pays)

Pays (code, nom) Personne[pays] ⊆ Pays[code]

Personne

nss	*nom*	*preno*	*naissan*	*pay*
SDHF45FR9	Dupont	Luc	1943	FR
ERYE75EFHS	White	Alfred	1899	UK
JFB9745HSD	White	Ted	1910	UK
SDHF8465SJ	Brown	John	1946	US
SF745HSFJ7	Smith	Ross	1987	IT
SHDGG475	Brown	James	1954	UK

Pays

cod	*nom*
FR	France
UK	Grande Bretagne
US	Etas Unis

Cette instance de la base de données VIOLE la contrainte

Clés étrangères

Spécifier une contrainte de clef étrangère en SQL :

```
CREATE TABLE Pays (
      code VARCHARR(4)
          PRIMARY KEY,
      nom VARCHAR(50)
);
```

```
CREATE TABLE Personne (
      nss VARCHAR(20) PRIMARY KEY,
      nom VARCHAR(50) NOT NULL,
      prenom VARCHAR(50) NOT NULL,
      naissance INTEGER,
      pays VARCHAR(4),
      UNIQUE (nom, prenom, naissance),
      FOREIGN KEY (pays) REFERENCES Pays(code)
);
```

Les attributs reliés doivent avoir exactement le même type

Les attribut référencés (ceux qui suivent la clause REFERENCES) doivent être une clef primaire de leur table

Clés étrangères

Conséquences de la contrainte de clé étrangère
(sur les tables Personne/Pays par exemple)

- Lors d'une insertion dans la table Personne :
  - vérification : la valeur de *pays* doit être parmi les valeurs de code dans la table Pays
- Lors de la mise à jour / suppression d'un code dans la table Pays
  - vérification : aucune personne n'appartient à ce pays

- Si ces vérifications n'on pas de succès
  - comportement par défaut : erreur
  - on peut demander explicitement un autre comportement (ON DELETE CASCADE….)

Contraintes génériques

D'autres contraintes sur les données ne peuvent pas être exprimées par les mécanismes vus jusqu'à maintenant

Exemple :

la note d'un examen est entre 0 et 20

le prix soldé d'un article est inférieur au prix entier

le salaire d'un manager est plus élevé que celui des ses subalternes

En SQL :

Contraintes sur une seule table : clause CHECK

Contraintes sur plusieurs tables : Assertions SQL

Clause CHECK

- Exemple : dans la table Article:

```
CREATE TABLE Article (
    id INTEGER PRIMARY KEY,
    nom VARCHAR(50) NOT NULL,
    prix NUMERIC NOT NULL,
    prix_solde NUMERIC,
    CHECK (prix > prix_solde)
);
```