Introduction aux bases de données

ITC PC

M. Charles

Introduction

Exemple

```
eleves = {
    "Tristan LETENEBREUX": {
        "options": ["Alld", "HGGSP", "Maths"],
        "contact_parent": "06 92 55 48 12",
    },
    "Geneviève LAJOIE": {
        "options": ["Alld", "SES", "Maths"],
        "contact_parent": "e.lajoie@gmail.com",
    },
    "Adam DURAND": {
        "options": ["Espagnol", "SES",
"Physique"],
        "contact_parent": "jrdurand@orange.fr",
    },
    ...
}

def ajouter_eleve(nom, contact_parent):
    assert(nom not in eleves)
    eleves[nom] = {
        "options": [],
        "contact_parent": contact_parent,
    }
```

Exemple

```
eleves = {
    "Tristan LETENEBREUX": {
        "options": ["Alld", "HGGSP", "Maths"],
        "contact_parent": "06 92 55 48 12",
    },
    "Geneviève LAJOIE": {
        "options": ["Alld", "SES", "Maths"],
        "contact_parent": "e.lajoie@gmail.com",
    },
    "Adam DURAND": {
        "options": ["Espagnol", "SES",
"Physique"],
        "contact_parent": "jrdurand@orange.fr",
    },
    ...
}

def ajouter_eleve(nom, contact_parent):
    assert(nom not in eleves)
    eleves[nom] = {
        "options": [],
        "contact_parent": contact_parent,
    }
```

- Écrire une fonction qui retire un élève du dictionnaire.

Exemple

```
eleves = {
    "Tristan LETENEBREUX": {
        "options": ["Alld", "HGGSP", "Maths"],
        "contact_parent": "06 92 55 48 12",
    },
    "Geneviève LAJOIE": {
        "options": ["Alld", "SES", "Maths"],
        "contact_parent": "e.lajoie@gmail.com",
    },
    "Adam DURAND": {
        "options": ["Espagnol", "SES",
"Physique"],
        "contact_parent": "jrdurand@orange.fr",
    },
    ...
}

def ajouter_eleve(nom, contact_parent):
    assert(nom not in eleves)
    eleves[nom] = {
        "options": [],
        "contact_parent": contact_parent,
    }
```

- Écrire une fonction qui retire un élève du dictionnaire.

```
def retirer_eleve(nom):
    del eleves[nom]
```

Exemple

```
eleves = {
    "Tristan LETENEBREUX": {
        "options": ["Alld", "HGGSP", "Maths"],
        "contact_parent": "06 92 55 48 12",
    },
    "Geneviève LAJOIE": {
        "options": ["Alld", "SES", "Maths"],
        "contact_parent": "e.lajoie@gmail.com",
    },
    "Adam DURAND": {
        "options": ["Espagnol", "SES",
"Physique"],
        "contact_parent": "jrdurand@orange.fr",
    },
    ...
}

def ajouter_eleve(nom, contact_parent):
    assert(nom not in eleves)
    eleves[nom] = {
        "options": [],
        "contact_parent": contact_parent,
    }
```

- Écrire une fonction qui retire un élève du dictionnaire.
- Écrire une fonction qui change le nom de la matière « Alld » en « Allemand ».

Exemple

```
eleves = {
    "Tristan LETENEBREUX": {
        "options": ["Alld", "HGGSP", "Maths"],
        "contact_parent": "06 92 55 48 12",
    },
    "Geneviève LAJOIE": {
        "options": ["Alld", "SES", "Maths"],
        "contact_parent": "e.lajoie@gmail.com",
    },
    "Adam DURAND": {
        "options": ["Espagnol", "SES",
"Physique"],
        "contact_parent": "jrdurand@orange.fr",
    },
    ...
}

def ajouter_eleve(nom, contact_parent):
    assert(nom not in eleves)
    eleves[nom] = {
        "options": [],
        "contact_parent": contact_parent,
    }
```

- Écrire une fonction qui retire un élève du dictionnaire.
- Écrire une fonction qui change le nom de la matière « Alld » en « Allemand ».

```
def changer_nom_matiere(ancien, nouveau):
    for (nom, donnees) in eleves.items():
        liste = donnees["options"]
        for i in len(liste):
            if liste[i] == ancien:
                liste[i] = nouveau

changer_nom_matiere("Alld", "Allemand")
```

Exemple

```
eleves = {
    "Tristan LETENEBREUX": {
        "options": ["Alld", "HGGSP", "Maths"],
        "contact_parent": "06 92 55 48 12",
    },
    "Geneviève LAJOIE": {
        "options": ["Alld", "SES", "Maths"],
        "contact_parent": "e.lajoie@gmail.com",
    },
    "Adam DURAND": {
        "options": ["Espagnol", "SES",
"Physique"],
        "contact_parent": "jrdurand@orange.fr",
    },
    ...
}

def ajouter_eleve(nom, contact_parent):
    assert(nom not in eleves)
    eleves[nom] = {
        "options": [],
        "contact_parent": contact_parent,
    }
```

- Écrire une fonction qui retire un élève du dictionnaire.
- Écrire une fonction qui change le nom de la matière « Alld » en « Allemand ».
- Écrire une fonction qui affiche la liste des élèves en SES.

Exemple

```
eleves = {
    "Tristan LETENEBREUX": {
        "options": ["Alld", "HGGSP", "Maths"],
        "contact_parent": "06 92 55 48 12",
    },
    "Geneviève LAJOIE": {
        "options": ["Alld", "SES", "Maths"],
        "contact_parent": "e.lajoie@gmail.com",
    },
    "Adam DURAND": {
        "options": ["Espagnol", "SES",
"Physique"],
        "contact_parent": "jrdurand@orange.fr",
    },
    ...
}

def ajouter_eleve(nom, contact_parent):
    assert(nom not in eleves)
    eleves[nom] = {
        "options": [],
        "contact_parent": contact_parent,
    }
```

- Écrire une fonction qui retire un élève du dictionnaire.
- Écrire une fonction qui change le nom de la matière « Alld » en « Allemand ».
- Écrire une fonction qui affiche la liste des élèves en SES.

```
def eleve_dans_matiere(nom):
    liste = eleves[nom]["options"]
    for i in len(liste):
        if liste[i] == nom:
            return True
    return False

def liste_matiere(matiere):
    for nom in eleves.keys():
        if eleve_dans_matiere(nom):
            print(nom)

liste_matiere("SES")
```

Exemple : analyse

Quel est le problème ?

Les informations sur les options suivies sont redondantes entre les élèves.

Exemple : analyse

Le problème n'est pas simplifié si on trie par option d'abord :

```
options = {
  "HGGSP": {
    "Tristan TENEBREUX": "06 92 55 48 12",
    ...
  },
  "SES": {
    "Geneviève LAJOIE": "e.lajoie@gmail.com",
    "Adam DURAND": "jrdurand@orange.fr",
    ...
  },
  "Alld": {
    "Tristan TENEBREUX": "06 92 55 48 12",
    "Geneviève LAJOIE": "e.lajoie@gmail.com",
    ...
  },
  "Espagnol": {
    "Adam DURAND": "jrdurand@orange.fr",
    ...
  }
  "Maths": {
    "Tristan TENEBREUX": "06 92 55 48 12",
    "Geneviève LAJOIE": "e.lajoie@gmail.com",
    ...
  },
  "Physique": {
    "Adam DURAND": "jrdurand@orange.fr",
    ...
  }
}
```

- Il est certes facile de changer le nom de « Alld »
- Mais désormais, rajouter un élève est très difficile.
- Changer le contact des parents pose alors le même problème.

Exemple : solution

Solution ? Il faut deux tableaux de données.

```
eleves = {
  0: {
    "nom": "Tristan TENEBREUX",
    "contact_parent": "06 92 55 48 12"
  },
  1: {
    "nom": "Geneviève LAJOIE",
    "contact_parent": "e.lajoie@gmail.com"
  },
  2: {
    "nom": "Adam DURAND",
    "contact_parent": "jrdurand@orange.fr"
  },
  ...
}
```

```
options = {
  "HGGSP": [0, ...],
  "SES": [1, 2, ...],
  "Alld": [0, 1, ...],
  "Espagnol": [2, ...],
  "Maths": [0, 1, ...],
  "Physique": [2, ...],
  ...
}
```

Exemple : solution

Il reste de nombreuses difficultés :

- Comment attribuer des numéros aux élèves de manière unique ?
- Comment valider au mieux les références croisées ?
- Les indirections dans la lecture des données sont sources d'erreur.
- Comment gérer des données dont le volume dépasse la capacité de la mémoire ?

La solution à tout ça : les **bases de données relationnelles**.

Principe des BDD(R)

Les **bases de données (relationnelles)** forment un paradigme d'organisation des données qui permet de **maximiser la cohérence** des informations tout en **minimisant la redondance**. Il est alors bien plus facile de lire, ajouter ou modifier des données.

Principe des BDD(R)

Les **bases de données (relationnelles)** forment un paradigme d'organisation des données qui permet de **maximiser la cohérence** des informations tout en **minimisant la redondance**. Il est alors bien plus facile de lire, ajouter ou modifier des données.

Au lieu d'un unique tableau global monolithique, une **base de données** (relationnelle) est constituée d'un ensemble de « petites » **tables** qui permettent de mieux organiser les données.

Exemple

`table Eleve`

idEleve	Nom	Contact parent
1	Tristan TENEBREUX	06 92 55 48 12
2	Geneviève LAJOIE	e.lajoie@gmail.com
3	Adam DURAND	jrdurand@orange.Fr

`table Option`

idOption	Nom
1	HGGSP
2	ESH
3	Alld
4	Espagnol
5	Maths
6	Physique

`table Affectations`

idEleve	idOption
1	1
1	3
1	5
2	2
2	3
2	5
...	...

Vocabulaire spécifique

Tables etc.

Une **table** est un ensemble de **lignes**, chaque ligne contenant les valeurs de différents **attributs**.

Chaque attribut possède un **domaine**, qui correspond à l'ensemble des valeurs qu'il peut prendre.

Vocabulaire intuitif	Vocabulaire BDD	Ex.
Table	Relation	`Eleve`
Colonne	Attribut	`Nom`
Ligne	Enregistrement	
Type	Domaine	`text`

Schémas de tables et de BDD

- Le **schéma** relationnel d'une table est l'ensemble ordonné de ses attributs et des domaines associés. **Exemples :**
  - `table Véhicule(idVéhicule int, Modèle text, Km int)`
  - `table Produit(idProduit int, Ref text, Prix money)`
- L'ensemble des schémas de toutes les tables est appelé **schéma de la base de données**.

Exercice

Donner le schéma de la base de données : *(on pourra utiliser les domaines `int` et `text`)*

`table Eleve`

idEleve	Nom	Contact parent
1	Tristan TENEBREUX	06 92 55 48 12
2	Geneviève LAJOIE	e.lajoie@gmail.com
3	Adam DURAND	jrdurand@orange.Fr

`table Option`

idOption	Nom
1	HGGSP
2	ESH
3	Alld
4	Espagnol
5	Maths
6	Physique

`table Affectations`

idEleve	idOption
1	1
1	3
1	5
2	2
2	3
2	5
…	…

Exercice : solution

```
Eleve(idEleve int, Nom text, Contact_Parent text)
Option(idOption int, Nom text)
Affectations(idEleve int, idEleve int)
```

Exercice : solution

```
Eleve(idEleve int, Nom text, Contact_Parent text)
Option(idOption int, Nom text)
Affectations(idEleve int, idEleve int)
```

- **Clé primaire** : colonne dans une table permettant d'identifier de manière unique les lignes de la table. Exemples :
  - dans la table `Eleve` : `idEleve` ;
  - dans la table `Option` : `idOption`
- **Clé étrangère** : colonne dans une table dont les valeurs correspondent à un identifiant unique dans une *autre* table.

Exercice 2

Que représente et comment fonctionne la base de données définies par le schéma suivant ?

```
Client(idClient int, Nom text, Prénom text, Adresse text)
Produit(idProduit int, Ref text, Prix money)
Livreur(idLivreur int, Nom text, Prénom text)
Véhicule(idVéhicule int, Modèle text, Kms int)
Commande(
    id int, idClient int, idProduit int, idLivreur int,
    idVéhicule int, Jour date, Heure time, Livré time
)
```

Requêtes SQL simples

Opérations unaires

Les **opérations unaires** travaillent sur une seule table. Elles peuvent être vues comme des fonctions qui prennent une table en entrée et produisent une table en sortie.

Langages de requête

Les bases de données sont en général situées sur un serveur qui implémente un certain **système de gestion de base de données** (SGBD). Parmi les exemples les plus connus, citons MySQL, Oracle, Microsoft SQL Server, PostgreSQL, etc.

Lorsqu'on souhaite accéder aux données d'un serveur de bases de données, on l'**interroge** à l'aide d'une **requête**. Les requêtes sont formulées dans un **langage de requête** spécifique.

Le programme des CPGE préconise l'apprentissage du langage de requête **SQL**. En particulier, il s'agit d'être capable de traduire des calculs « simples » en requêtes SQL.

Développement par IBM dans les **années 70** :

1. SQUARE (*Specifying Queries in A Relational Environment*)
2. SEQUEL
3. SQL
4. *Structured Query Language*

Projection avec `SELECT … FROM`

La **projection** est une opération permettant de conserver certaines **colonnes** d'une table. En SQL :

```
SELECT c1, ..., ck FROM T ;
```

En général, on utilise une projection en fin de calcul, pour éliminer les colonnes que l'on ne souhaite pas voir dans le résultat.

Exemple. La table des noms et prénoms des clients peut être obtenue par la requête

```
SELECT Nom, Prénom FROM Client ;
```

qui a pour résultat la table

Nom	Prénom
Dumbledore	Albus
Weasley	Fred
McGonagall	Minerva
Riddle	Tom
Weasley	Georges
Diggory	Cédric

Sélection avec `WHERE`

La **sélection** est une opération permettant de conserver les **lignes** d'une table qui vérifient un certain critère. En SQL :

```
SELECT * FROM T WHERE condition·s ;
```

C'est l'opération la plus souvent utilisée.

Sélection avec `WHERE`

Exemple. La table des pizzas coûtant moins de 11 euros peut être obtenue par la requête

```
SELECT * FROM Produit WHERE Prix <= 11 ;
```

qui a pour résultat la table

idProduit	Ref	Prix
2	Royale	9.00
4	Bretonne	11.00
6	Végétarienne	11.00

Renommage avec `AS`

Il est parfois utile de pouvoir renommer un ou des attributs. En SQL :

```
SELECT c1 AS a1, ..., ck AS ak FROM T ;
```

Renommage avec AS

Exemple. Si on souhaite renommer `Ref` par `Pizza` dans la table `Produit`, on écrira

```
SELECT Ref AS Pizza FROM Produit ;
```

Combinaisons de mots-clé

Une requête peut être construite en combinant différents mots-clé. Exemple :

```
SELECT Prix, Ref AS Pizza
FROM   Produit
WHERE  Prix >= 12 ;
```

Résultat :

Prix	Pizza
13.00	Savoyarde
12.00	Forézienne
12.00	Océane
13.00	Canicatti