SQL 1
Introduction, vocabulaire et requêtes exigibles

L'administration, les banques, les assurances, les secteurs de la finance utilisent des **bases de données**, systèmes d'informations qui stockent dans des fichiers les nombreuses données qui leur sont nécessaires.

Une base de données relationnelle permet d'organiser, de stocker, de mettre à jour et d'interroger des données structurées volumineuses utilisées simultanément par différents programmes ou différents utilisateurs. Un logiciel, le **système de gestion de bases de données relationnelles** (SGBDR), est utilisé pour la gestion (lecture, écriture, cohérence, actualisation...) des bases de données dans lesquelles sont stockées les données. Le principal avantage des bases de données relationnelles est le SQL, acronyme de Structured Query Language, un langage fonctionnant avec des requêtes permettant de rechercher, ajouter, modifier ou supprimer des données dans les bases de données relationnelles.

Nous commençons par une présentation du vocabulaire lié à l'analyse de données.

I Vocabulaire de base

Définitions 1 — Table, base de données relationnelle

D1 Une **table** est un tableau de données à deux dimensions dans lequel chaque ligne représente un objet ; tous les objets étant de même nature.

D2 Une **base de données relationnelle** (*relational database* en anglais) est un ensemble de tables et de relations entre ces tables.

Petite remarque
L'ordre des lignes n'a aucune importance.

Exemple 1

On considère la table `etudiants_ECG_chartreux` suivante :

identifiant	nom	prenom	date_de_naissance	classe	ESH_ou_HGG	appli_ou_appro	LVB
1	PTIPEU	Justin	2004-12-31	ECG1	ESH	Appli	Espagnol
2	ZEMMOIS	Elie	2004-08-21	ECG2	ESH	Appro	Italien
3	MINEE	Elie	2004-05-01	ECG3	HGG	Appro	Allemand
4	HAIRE	Axelle	2004-12-31	ECG3	ESH	Appli	Chinois
5	HAUPIED	Djamal	2004-08-21	ECG4	ESH	Appli	Espagnol
6	HAUPIED	Geoffroy	2004-05-01	ECG4	HGG	Appli	Allemand
...	...	...	...	...	...	...	...

Ainsi que la table `classes_ECG_chartreux` :

classe	professeur_principal	maths_appli	maths_appro	ESH	HGG	philo	français	...
ECG1	J. Rigaud-Poole	R. Pellerin	Z. Zaoui	G. Perrin	F. Bouteille	M. Caillou	P. Cournault	...
ECG2	E. Berrucaz		F. Doré	E. Bahrini	T. Merle	M. Caillou	P. Cournault	...
ECG3	F. Jehl	R. Pellerin	Z. Zaoui			M. Collin	F. Jehl	...
ECG4	J. Legendre	J. Legendre		E. Bahrini	T. Merle	M. Caillou	F. Jehl	...

Définitions 2

D1 Les lignes d'une table sont parfois appelées **enregistrements** ou **occurrences**.

D2 Les colonnes d'une table sont parfois appelées `attributs`. L'ensemble des colonnes forme les **descripteurs** de l'objet étudié.

D3 Le **schéma de la table** est constitué d'un nom suivi de la liste des colonnes et de leur type.

Petites remarques
- Les valeurs contenues dans une même colonne doivent toutes être du même type parmi : INTEGER, TEXT, DATE, FLOAT, BOOLEAN.
- On utilisera le mot attribut pour désigner l'information contenue dans une colonne (autrement dit, son nom).

Exemples 2

E1 La table `classes_ECG_chartreux` comporte 4 lignes.

E2 La table `etudiants_ECG_chartreux` comporte 8 colonnes. On peut résumer son schéma ainsi :
`etudiants_ECG_chartreux(identifiant:INTEGER,nom:TEXT,prenom:TEXT,date_de_naissance:DATE,classe:TEXT,ESH_ou_HGG:TEXT,appli_ou_appro:TEXT,LVB:TEXT)`

II Notion de clé

Dans l'exemple de la table `etudiants_ECG_chartreux`, il est tout à fait possible que deux étudiants aient les mêmes noms, ou prénoms, ou dates de naissance et soient dans la même classe. Comment identifier de manière unique chaque ligne ?

Définitions 3 — Clé, clé primaire

D1 Une **clé** d'une table est un attribut ou un ensemble d'attributs permettant d'identifier de manière unique chaque ligne de la table.

D2 Choisir une **clé primaire** (*primary key*, PK, en anglais) c'est choisir définitivement *une seule clé* de la table.

Important !
Quand on choisit une clé primaire d'une table, elle doit le rester même si des lignes sont ajoutées à la table...
Une table doit toujours contenir une clé primaire.

Exemples 3

E1 Reprenons la table `etudiants_ECG_chartreux` :
- La colonne `nom` n'est pas une clé, car deux étudiants ont le même nom. De même, les colonnes `prenom` et `date_de_naissance` ne sont pas des clés.
- L'ensemble `{nom,prenom}` est une clé si on restreint la table au tableau que nous voyons. Il est possible que pour la table complète, cet ensemble ne soit pas une clé si deux étudiants ont même nom et même prénom.
- De façon générale, on aime choisir une clé primaire simple (qui contient le moins de colonnes possible), quitte à ajouter une colonne à la table. C'est ce que nous avons fait en créant une colonne `identifiant`, qui est bien une clé.

☞ Pour info...
On a ajouté une colonne artificielle, qui est l'identifiant, pour avoir une clé primaire simple. Bien souvent, cet identifiant est un entier incrémenté automatiquement à chaque ligne de la table.

E2 Reprenons la table `classes_ECG_chartreux` :
- La colonne `ESH` n'est pas une clé.
- Les colonnes `classe` et `professeur_principal` sont des clés.

E3 La base de données de la sécurité sociale concernant les bénéficiaires comporte une clé primaire : le numéro de sécurité sociale.

C'est bien souvent une mauvaise idée de vouloir regrouper toutes les données dans une même table. Avec les exemples de nos deux tables, nous allons pouvoir extraire les informations sur les professeurs d'un étudiant. Si l'enseignant d'ESH en ECG4 change, il suffit de le changer dans la table `classes_ECG_chartreux`. Alors que s'il avait été mis en colonne dans le tableau des étudiants, il faudrait le changer pour tous... au risque de se tromper ou d'en oublier. Dans une base de données, il est impératif de limiter la redondance des données, quitte à augmenter le nombre de tables et les relations entre elles. Nous verrons ensuite que cela ne pose pas de problème, bien au contraire !
L'intérêt est de pouvoir croiser les informations et les extraire d'une table sur l'autre.

On peut aussi imaginer qu'à chaque étudiant on attribue un numéro de badge ; et que l'on a une table `badges` indiquant à qui il est attribué, quelles doivent être ses autorisations...

Définition 4 — Clé étrangère

Une **clé étrangère** (*foreign key*, FK, en anglais) est un attribut d'une table qui est une clé primaire pour une autre table de la même base de données.

Exemple 4

L'attribut `classe` de `etudiants_ECG_chartreux` n'est pas une clé de cette table, mais c'est une clé étrangère, puisqu'il s'agit d'une clé primaire de la table `classes_ECG_chartreux`.

Important !
C'est cette clé étrangère qui va permettre de faire le lien entre les deux tables. En effet, on pourra extraire les informations relatives à la classe d'un étudiant grâce à elle.

III Table d'association

On considère maintenant la table suivante nommée `info_ecoles` :

identifiant (PK)	ecole	rang	frais_scolarité_non_boursiers	...
1	HEC	1	64500	...
2	ESSEC	2	60280	...
3	ESCP	3	62140	...
...	...	...	...	...

Important !
Puisque le rang peut changer d'une année sur l'autre, il ne constitue pas une bonne PK... Le nom de l'école peut également changer ! D'où la création d'un identifiant pour chaque école.

On souhaite, dans un premier temps, lister les admissibilités de tous les étudiants d'ECG. Nous pouvons donc ajouter une colonne dans notre table d'étudiants avec l'attribut **admissibilites** (indiquant l'identifiant de l'école dans laquelle l'étudiante ou l'étudiant est admissible) qui est une clé étrangère vers la table **ecoles**.
Et si une étudiante ou un étudiant est admissible dans 2 écoles, comment fait-on ?
On pourrait mettre deux colonnes avec les attributs **admissibilite1** et **admissibilite2** qui seraient toutes les deux des clés étrangères vers la table **ecoles**. Mais ce raisonnement n'est pas très bon : on ne peut pas savoir à l'avance dans combien d'écoles les étudiantes seront admissibles, tous les étudiants ne sont pas admissibles au même nombre d'écoles... On peut s'en sortir, mais cette solution n'est pas élégante.

Ce problème motive la définition suivante.

Définition 5 — Table d'association

Une **table d'association** est une table contenant en colonnes au moins deux clés étrangères vers d'autres tables de la base de données.

Exemple 5

La table d'association `admissibilites` listant les admissibilités de chaque étudiant :

etudiant	ecole
1	3
1	4
1	5
2	1
2	3
2	4
...	...

Petite remarque
Il est possible d'ajouter d'autres colonnes à cette table ; du moment qu'elle contient au moins deux FK.

On a alors une table simple à remplir et il sera tout aussi simple d'extraire les informations voulues.
Par exemple, si l'on veut les admissibilités de Justin PTIPEU, il suffira d'aller chercher les lignes contenant 1 dans la colonne `etudiant`. Dans l'autre sens, si l'on souhaite la liste de tous les étudiants admissibles à HEC, il nous suffira d'aller chercher les lignes contenant 1 dans la colonne `ecole`.

Question : peut-on donner une PK à partir des colonnes déjà existants ?
L'ensemble `{etudiant,ecole}` est une clé primaire de cette table.

IV Commandes SQL exigibles

✘ Attention !
Même si l'usage consiste à écrire en `MAJUSCULE` les mots-clés du langage SQL et en `minuscule` le reste, le langage SQL ne fait pas la différence entre les deux. Par conséquent, un mot-clé ne pourra pas être utilisé par l'utilisateur pour autre chose que ce à quoi il est destiné.

Dans toute cette partie, les commandes seront énoncées pour des tables nommées `tableau1` et `tableau2` dont les colonnes seront `colonne1_1`, `colonne1_2`,...

Commande 1 — Création d'une table et déclaration des clés

La requête :

```
CREATE TABLE tableau1 (
    colonne1_1 TYPE,
    colonne1_2 TYPE,
    ...
    colonne1_n TYPE,
    PRIMARY KEY (colonne1_i),
    FOREIGN KEY (colonne1_k) REFERENCES tableau2(colonne2_j)
);
```

permet de :

- créer la table `tableau1` dont les colonnes sont `colonne1_1`, `colonne1_2`,... `colonne1_n` ;
- déclarer `colonne1_i` comme clé primaire de cette table ;
- déclarer `colonne1_j` comme clé étrangère dans cette table, faisant référence à la clé primaire `colonne2_j` de la table `tableau2`.

Exemple 6

Écrivons la requête de création de la table `etudiants_ECG_chartreux`.

```
CREATE TABLE etudiants_ECG_chartreux (
    identifiant INTEGER,
    nom TEXT,
    prenom TEXT,
    date_de_naissance DATE,
    classe TEXT,
    ESH_ou_HGG TEXT,
    appli_ou_appro TEXT,
    LVB TEXT,
    PRIMARY KEY (identifiant)
);
```

Commandes 2 — Insertion de lignes

C1 La requête :

```
INSERT INTO tableau VALUES ('valeur1','valeur2',...,'valeurn')
```

permet d'insérer une ligne telle que pour tout i, `valeuri` est la valeur associée à `colonne_i`.

C2 On peut insérer une ligne ne contenant pas des valeurs pour chaque colonne. Deux possibilités :

- soit on laisse une valeur vide pour la colonne correspondante ;
- soit on liste entre parenthèse les colonnes à compléter avant la commande `VALUES`

Petite remarque

On met les valeurs entre '...', sauf éventuellement s'il s'agit de nombres (ATTENTION aux majuscules dans les différentes valeurs sous forme de texte...).

Exemple 7

Écrivons la requête d'insertion de la première ligne de la table `etudiants_ECG_chartreux`.

```
INSERT INTO etudiants_ECG_chartreux VALUES ('1','PTIPEU','Justin','2004-12-31','ECG1','ESH','Appli','Espagnol')
```

Commandes 3 — Suppression de lignes

C1 La requête :

```
DELETE FROM tableau WHERE condition
```

permet de supprimer les lignes de `tableau` qui vérifient `condition`.

C2 La requête

```
DELETE FROM tableau
```

supprime toutes les données contenues dans `tableau`.

Opérateurs

- de comparaison : `A=B` : A est égal à B (on écrit 'B' en cas de texte)
  `A<>B` : A est différent à B
  `A>B` : A est supérieur à B (< existe aussi)
  `A BETWEEN B AND C` : A est compris entre B et C
  `A IN (B1,B2,...) 'B'` : A est dans la liste (B1,B2,...)
  `A IS NULL` : A n'a pas de valeur
- logiques : `AND,OR,NOT`
- arithmétiques : +,-,*.

☞ Pour info...

Pour supprimer la table de la base de données, on utilise `DROP TABLE tableau1`.

Exemple 8

On suppose que la table `info_ecoles` est complétée par toutes les écoles de la BCE et Ecricome.
Écrivons les requêtes permettant de :

- supprimer BSB :
  `DELETE FROM info_ecoles WHERE ecole=BSB`
- supprimer les écoles dont le rang est compris entre 15 et 20 :
  `DELETE FROM info_ecoles WHERE rang BETWEEN 15 AND 20`
- supprimer les écoles dont les frais de scolarité sont strictement supérieurs à 60000 euros :
  `DELETE FROM info_ecoles WHERE frais_scolarite_non_boursiers > 60000`

Commande 4 — Modification

La requête :

```
UPDATE tableau SET colonne_i='valeur' WHERE condition
```

remplace la valeur, située à l'intersection de la colonne **colonne_i** et des lignes désignées (une ou plusieurs) par **condition**, par la valeur **valeur**.

Petites remarques

- Il est possible de modifier les valeurs de plusieurs colonnes.
- Si on ne précise pas de condition, on attribue la même valeur à toute la colonne modifiée.

Exemple 9

On suppose que la table **info_ecoles** est complétée par toutes les écoles de la BCE et Ecricome.

- Écrivons des requêtes permettant de modifier le classement EDHEC et EML en 5 et 4 respectivement.
  `UPDATE info_ecoles SET rang='5' WHERE ecole=EDHEC`
  `UPDATE info_ecoles SET rang='4' WHERE ecole=EML`
- Écrivons une requête permettant de modifier les frais de scolarité des non boursiers de toutes les écoles hors TOP5 en un coût fixe de 30000 euros.
  `UPDATE info_ecoles SET frais_scolarite_non_boursiers='30000' WHERE rang>5`
- Écrivons une requête permettant de modifier les frais de scolarité des non boursiers de toutes les écoles en un coût fixe de 15000 euros.
  `UPDATE info_ecoles SET frais_scolarite_non_boursiers='15000'`
- Écrivons une requête permettant de modifier le nom et les frais de scolarité d'HEC.
  UPDATE info_ecoles SET ecole='achessai',frais_scolarite_non_boursiers='0' WHERE ecole='HEC'

Commandes 5 — Projection et restriction

C1 **Projection.**

- La requête

  ```
  SELECT colonne_i FROM tableau
  ```

  permet l'affichage de **colonne_i** de la table **tableau**.
- La requête

  ```
  SELECT colonne_1,colonne_2,... FROM tableau1
  ```

  permet l'affichage de **colonne_1**,**colonne_2**,... de la table **tableau**.
- La requête

  ```
  SELECT * FROM tableau1
  ```

  permet l'affichage des toutes les colonnes de la table **tableau**.

Petite remarque

Ne pas hésiter à écrire les requêtes sur plusieurs lignes pour mieux identifier les arguments...

C2 **Restriction (ou sélection).**

- La requête

  ```
  SELECT colonne_i FROM tableau WHERE condition
  ```

  permet l'affichage de **colonne_i** de la table **tableau** en ne retenant que les lignes vérifiant **condition**.
- La requête

  ```
  SELECT colonne_1,colonne_2,... FROM tableau WHERE condition
  ```

  permet l'affichage de **colonne1**,**colonne2**,... de la table **tableau** en ne retenant que les lignes vérifiant **condition**.
- La requête

  ```
  SELECT * FROM tableau WHERE condition
  ```

  permet l'affichage des toutes les colonnes de la table **tableau** en ne retenant que les lignes vérifiant **condition**.

Exemple 10

On suppose la table **etudiants_ECG_chartreux** créée.

- Écrivons une requête permettant d'obtenir le nom et prénom des étudiants faisant ESH.
  `SELECT nom,prenom FROM etudiants_ECG_chartreux WHERE ESH_ou_HGG='ESH'`
- Écrivons une requête permettant d'obtenir toutes les informations des étudiants faisant chinois.
  `SELECT * FROM etudiants_ECG_chartreux WHERE LVB='Chinois'`
- Écrivons une requête permettant d'obtenir toutes les informations des étudiants faisant ESH et chinois.
  `SELECT * FROM etudiants_ECG_chartreux WHERE ESH_ou_HGG='ESH' AND LVB='Chinois'`

Commandes 6 — Jointure

C1 La requête

```
SELECT * FROM tableau1 JOIN tableau 2
```

permet l'affichage du produit cartésien de `tableau1` et `tableau2`.

C2 La requête

```
SELECT * FROM tableau1 JOIN tableau 2 ON tableau1.colonne1_i=tableau2.colonne2_j
```

permet l'affichage du produit cartésien de `tableau1` et `tableau2` pour les seules lignes où la valeur de `colonne1_i` est égale à celle de `colonne2_j`.

Petite remarque

Si on ne veut que certaines colonnes, on remplace * par les colonnes des deux tables qui nous intéressent.

Exemple 11

Considérons les tables `etudiants` et `ecoles` suivantes :

`etudiants`

id	nom	admission
1	PTIPEU	HEC
2	ZEMMOIS	ESSEC
3	MINEE	ESSEC

`ecoles`

ecole	cout
HEC	64500
ESSEC	60280
ESCP	62140

- La requête `SELECT * FROM etudiants JOIN ecoles` permet d'obtenir :

id	nom	admission	ecole	cout
1	PTIPEU	HEC	HEC	64500
1	PTIPEU	HEC	ESSEC	60280
1	PTIPEU	HEC	ESCP	62140
2	ZEMMOIS	ESSEC	HEC	64500
2	ZEMMOIS	ESSEC	ESSEC	60280
2	ZEMMOIS	ESSEC	ESCP	62140
3	MINEE	ESSEC	HEC	64500
3	MINEE	ESSEC	ESSEC	60280
3	MINEE	ESSEC	ESCP	62140

- La requête `SELECT nom,admission,cout FROM etudiants JOIN ecoles ON etudiants.admission=ecoles.cout` permet d'obtenir :

nom	admission	cout
PTIPEU	HEC	64500
ZEMMOIS	ESSEC	60280
MINEE	ESSEC	60280