

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 données nombreuses 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 – RELATION, BASE DE DONNÉES RELATIONNELLE

- D1#** Une **table** ou **table de relation** est un tableau à deux dimensions de données 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 tables et de relations entre ces tables.

#### 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
...	...	...	...	...	...	...	...

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

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 appelées **enregistrements** ou **occurrences**.
- D2#** Les colonnes d'une table sont appelées **attributs**. L'ensemble des attributs forme les **descripteurs** de l'objet étudié.
- D3#** Le **domaine d'un attribut** (ou champ) est le type des valeurs possibles de cet attribut (ces valeurs doivent toutes être du même type) : INTEGER, TEXT, DATE, FLOAT, BOOLEAN.
- D4#** Le **schéma de la table** est constitué d'un nom suivi de la liste des attributs et de leur type.

#### EXEMPLES 2

- E1** La table **classes\_ECG\_chartreux** comporte 4 enregistrements.

**E2** La table `etudiants_ECG_chartreux` comporte 8 attributs. 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...

### EXEMPLES 3

**E1** Reprenons la table `etudiants_ECG_chartreux` :

- L'attribut `nom` n'est pas une clé, car deux étudiants ont le même nom. De même, les attributs `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 d'attributs possible), quitte à ajouter un attribut à la table. C'est ce que nous avons fait en créant un attribut `identifiant`, qui est bien une clé.

**E2** Reprenons la table `classes_ECG_chartreux` :

- L'attribut `ESH` n'est pas une clé.
- Les attributs `classe` et `professeur_principal` sont des clés. Mais on choisit plutôt l'attribut `classe` comme clé primaire, qui est un attribut plus explicite.

#### Pour info...

On a ajouté un attribut artificiel, 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.

C'est bien souvent une mauvaise idée de vouloir regrouper tous les attributs 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 attribut 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 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 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_scolarite_non_boursiers	...
1	HEC	1	64500	...
2	ESSEC	2	60280	...
3	ESCP	3	64780	...
...	...	...	...	...

On pourrait imaginer une relation contenant les tarifs de chaque école pour les différents échelons de bourse.

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** qui est une clé étrangère vers la table **ecoles**. Et si 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 un étudiant va être admissible, tous les étudiants ne sont pas admissibles au même nombre d'écoles... On peut s'en sortir, mais cette solution n'est pas élégante.

La stratégie inverse ne fonctionne pas non plus. On pourrait envisager de mettre dans la table **ecoles** un attribut **etudiant\_admissible** qui serait une clé étrangère vers la table **etudiants**. Mais cela voudrait dire qu'une école ne peut sélectionner qu'un seul étudiant admissible (fâcheux!).

Ce problème motive la définition suivante.

### DÉFINITION 5 - TABLE D'ASSOCIATION

Une **table d'association** est une relation contenant en attributs au moins deux clés étrangères vers d'autres relations 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
...	...

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 l'attribut **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 l'attribut **ecole**.

**Question** : peut-on donner une PK à partir des attributs déjà existants ?

Il est possible d'ajouter d'autres attributs à cette table ; du moment qu'elles contiennent au moins deux FK.

## IV COMMANDES SQL EXIGIBLES

Dans toute cette partie, les commandes seront énoncées pour des relations nommées **tableau1**, **tableau2** dont les attributs seront **attribut1\_1**, **attribut1\_2**,...

### COMMANDE 1 - CRÉATION D'UNE TABLE ET DÉCLARATION DES CLÉS

La requête :

```
CREATE TABLE tableau1 (
    attribut1_1 DOMAINE,
    attribut1_2 DOMAINE,
    ...
    attribut1_n DOMAINE,
    PRIMARY KEY (attribut1_i),
    FOREIGN KEY (attribut1_k) REFERENCES tableau2(attribut2_j)
);
```

permet de :

- créer la table **tableau1** dont les attributs sont **attribut1\_1**, **attribut1\_2**,... **attribut1\_n**;
- déclarer **attribut1\_i** comme clé primaire de cette table;
- déclarer **attribut1\_j** comme clé étrangère dans cette table, faisant référence à la clé primaire **attribut2\_j** de la table **tableau2**.

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.

#### EXEMPLE 6

Écrivons la requête de création de la table **etudiants\_ECG\_chartreux**.

## COMMANDES 2 - INSERTION DE LIGNES

C1# La requête :

```
INSERT INTO tableau1 VALUES ('valeur1','valeur2',...,'valeurn')
```

permet d'insérer une ligne telle que pour tout  $i$ , **valeuri** est la valeur associée à **attributi**.

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

- soit on laisse une valeur vide pour l'attribut correspondant;
- soit on liste entre parenthèse les attributs à compléter avant la commande **VALUES**

### Petite remarque

On met les valeurs entre '...', sauf éventuellement s'il s'agit de nombres.

### EXEMPLE 7

Écrivons la requête d'insertion de la première ligne de la table **etudiants\_ECG\_chartreux**.

## COMMANDES 3 - SUPPRESSION DE LIGNES

C1# La requête :

```
DELETE FROM tableau1 WHERE condition
```

permet de supprimer les lignes de **tableau1** qui vérifie **condition**.

C2# La requête

```
DELETE FROM tableau1
```

supprime toutes les données contenues dans **tableau1**.

### 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 Ericome. Écrivons les requêtes permettant de :

- supprimer BSB :
- supprimer les écoles dont le rang est compris entre 15 et 20 :
- supprimer les écoles dont les frais de scolarité sont strictement supérieurs à 60000 euros :

## COMMANDE 4 - MODIFICATION

La requête :

```
UPDATE tableau1 SET attributi='valeur' WHERE condition
```

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

### EXEMPLE 9

On suppose que la table **info\_ecoles** est complétée par toutes les écoles de la BCE et Ericome.

- Écrivons une requête permettant de modifier le classement EDHEC et EML en 5 et 4 respectivement.
- 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.
- Écrivons une requête permettant de modifier les frais de scolarité des non boursiers de toutes les écoles en un coût fixe de 15000 euros.

## COMMANDES 5 - PROJECTION ET RESTRICTION

### C1# Projection.

- La requête

```
SELECT attributi FROM tableau1
```

permet l'affichage de la colonne correspondant à **attributi** de la table **tableau1**.

- La requête

```
SELECT attribut1,attribut2,... FROM tableau1
```

permet l'affichage des colonnes correspondant à **attribut1,attribut2,...** de la table **tableau1**.

- La requête

```
SELECT * FROM tableau1
```

permet l'affichage des toutes les colonnes de la table **tableau1**.

### C2# Restriction (ou sélection).

- La requête

```
SELECT attributi FROM tableau1 WHERE condition
```

permet l'affichage de la colonne correspondant à **attributi** de la table **tableau1** en ne retenant que les lignes vérifiant **condition**.

- La requête

```
SELECT attribut1,attribut2,... FROM tableau1 WHERE condition
```

permet l'affichage des colonnes correspondant à **attribut1,attribut2,...** de la table **tableau1** en ne retenant que les lignes vérifiant **condition**.

- La requête

```
SELECT * FROM tableau1 WHERE condition
```

permet l'affichage des toutes les colonnes de la table **tableau1** en ne retenant que les lignes vérifiant **condition**.

#### Petite remarque

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

### EXEMPLE 10

On suppose la table **etudiant\_ECG\_chartreux** créée.

- Écrivons une requête permettant d'obtenir le nom et prénom des étudiants faisant ESH.
- Écrivons une requête permettant d'obtenir toutes les informations des étudiants faisant chinois.

- Écrivons une requête permettant d'obtenir toutes les informations des étudiants faisant HGG et italien.
- Écrivons une requête permettant d'obtenir toutes les informations relatives à Justin PTIPEU.

### 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.attribut1_i=tableau2.attribut2_j
```

permet l'affichage du produit cartésien de `tableau1` et `tableau2` pour les seules lignes où la valeur de `attribut1_i` est égale à celle de `attribut2_j`.

#### EXEMPLE 11

Considérons les `etudiants` et `ecoles` suivantes :

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

ecole	cout
HEC	64500
ESSEC	60280
ESCP	64780

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

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