1. Une base de données en pratique

Une base de données est un ensemble de tables contenant des données généralement du type texte ou numérique.

On a créé un site permettant aux élèves de s'inscrire à des ateliers d'AP.
Ci-après un extrait des tables eleve et atelier de la base de données du site.

id	nom	prenom	classe	atelier
1	BEAUBOURG	Alain	1	2
2	DUBERNARD	Luc	6	1
3	CASTRON	Alissa	4	3
4	GUEPPAN	Hélène	2	3

id	nom	description	nbplace	nbinscrit
1	TICE	Organisé par les …	20	2
2	English Project	Dans le cadre de …	18	1
3	Intercambio …	Echanges en espagnol …	12	3

La table eleve est composée de 5 champs ou attributs. On préfèrera le terme attribut car selon le contexte, le terme « champ » peut désigner une colonne, le titre d'une colonne ou une case de la table. Un enregistrement est une ligne de la table.

Chaque attribut est déterminé par son nom ('idatelier', 'nom', 'nbplace'…) et son type (entier, chaine de caractères…) ou son domaine (ensemble des valeurs prises : $\mathbb{N}$, un intervalle, une liste de noms…).

Une clé est constituée d'un ou plusieurs champs permettant d'identifier chaque enregistrement de manière unique. Une clé primaire est une clé choisie comme clé principale. Le plus souvent, on utilise un champ dédié du type integer qui s'auto-incrémente, un identifiant.

Un logiciel permettant de créer et de gérer une base de données est appelé serveur. L'ordinateur équipé de ce logiciel, hébergeant la base de données, est également appelé serveur.
Un client est un logiciel, ou un ordinateur, permettant d'accéder à une base de données existante située sur un serveur.

Un système de base de données est divisé en trois couches (ou niveaux).

- Couche présentation ou premier niveau : partie visible, interface utilisateur graphique ou texte, transmet les requêtes de l'utilisation à la couche métier, et les réponses de la couche métier à l'utilisateur.
- Couche métier ou second niveau : partie fonctionnelle de l'application, traitement des données, gestion des opérations, communique avec les deux autres couches.
- Couche accès aux données ou troisième niveau : gère l'accès aux données, qui peuvent être stockées localement ou externes, et les transmets à la couche métier.

Dans un système client-serveur, le client gère la couche présentation et une partie de la couche métier, le serveur gère la couche accès aux données et une partie de la couche métier.
Dans un système à trois niveaux, appelé architecture trois-tiers, chaque couche est gérée de manière indépendante, par trois ordinateurs (client, serveur d'application, serveur de données) ou par trois fonctions d'un même programme.

2. Quelques définitions formelles

Soient D_1, D_2, …, D_n des ensembles. Un sous-ensemble de $\prod_{i=1}^{n} D_i$ est appelé relation sur $\prod_{i=1}^{n} D_i$. Une relation est donc un ensemble de n-uplets.
Le domaine D_i est l'ensemble des valeurs prises par l'attribut A_i.
Le schéma d'une relation est l'ensemble $\{A_1 : D_1, \ldots, A_n : D_n\}$. Il spécifie la relation.
Le schéma relationnel d'une base de données est l'ensemble des schémas des relations (liste des tables, de leurs attributs et de leurs domaines). Construire le schéma relationnel est la première étape dans la création d'une base de données.
Une clé est un sous-ensemble minimal d'attributs $\{A_{i_1}, \ldots, A_{i_k}\}$ tel que :

$$\forall (t, s) \in R^2, \left(t_{i_1}, \ldots, t_{i_k}\right) = \left(s_{i_1}, \ldots, s_{i_k}\right) \Rightarrow t = s$$

Exemple : la table eleve précédente est une relation sur $\prod_{i=1}^{5} D_i$ avec
$D_1 = D_4 = D_5 = \mathbb{N}^*$ et $D_2 = D_3 = \{$chaines de caractères$\}$

Le schéma de cette relation est {*id* : int, *nom* : varchar(255), *prenom* : varchar(255), *classe* : int, *atelier* : int}.

int désigne le type integer

varchar(255) désigne une chaine de 255 caractères maximum

L'attribut A_1, id, est une clé primaire.

3. Algèbre relationnelle et requêtes SQL

L'algèbre relationnelle est la branche des mathématiques qui définit les opérations sur les relations.

Le langage SQL (Structured Query Language) permet d'effectuer des requêtes sur les bases de données. En pratique, on peut programmer des requêtes SQL en Python (hors programme) ou bien utiliser une interface graphique. L'interface en ligne la plus connue pour travailler sur les bases de données des sites web est phpMyAdmin.

Chaque opération peut être décrite dans le cadre de l'algèbre relationnelle ou en langage SQL.

Dans la suite, R et R' désignent des relations (ou des tables) de schémas respectifs $\{A_1 : D_1, \dots, A_n : D_n\}$ et $\{A'_1 : D'_1, \dots, A'_n : D'_n\}$.
Un attribut de R est noté $R.A_i$ (ou juste A_i s'il n'y a pas d'ambiguïté).
Un élément t de R est sous la forme $(t_1, t_2, \dots, t_n)$.
La valeur de l'attribut A_i est t_i ou $t.A_i$.

3.1 Projection

Soit $A = \{A_{i_1}, \dots, A_{i_k}\}$ un sous-ensemble de $\{A_1, \dots, A_n\}$.
La projection de R sur A, notée $\pi_A(R)$ est la relation obtenue à partir de R en conservant uniquement les attributs $A_{i_1}, \dots, A_{i_k}$.

$$\pi_{\{A_{i_1},\dots,A_{i_k}\}}(R) = \{(t_{i_1}, \dots, t_{i_k}); t \in R\}$$

SQL : SELECT $A_{i_1}, \dots, A_{i_k}$ FROM R

Exemple : SELECT nom, prenom FROM eleve

3.2 Sélection

Une sélection renvoie une partie de R vérifiant un critère C.

$\sigma_C(R)$ est la relation obtenue à partir de R en conservant uniquement les enregistrements dont les attributs vérifient le critère C.

$$\sigma_{A_i=a}(R) = \{t \in R; t_i = a\}$$

$$\sigma_{A_i<a \text{ et } A_j=b}(R) = \{t \in R; t_i < a \text{ et } t_j = b\}$$

SQL : SELECT $*$ FROM R WHERE *condition*
Le critère peut être exprimé à l'aide d'égalités, d'inégalités, des opérateurs AND, OR, NOT, BETWEEN, IN, LIKE, IS NULL, IS NOT NULL.

Exemples : SELECT $*$ FROM eleve WHERE atelier = 3 AND classe = 2
SELECT $*$ FROM eleve WHERE classe BETWEEN 3 AND 6
SELECT $*$ FROM eleve WHERE NOT nom LIKE 'D%'

Remarques : Si le critère utilise AND et OR, on utilisera des parentheses, même si on sait que AND est prioritaire sur OR.
LIKE 'nom' recherche le nom exact; _ peut remplacer un caractère unique et % une chaîne de caractères de longueur quelconque.

3.3 Renommage

On conserve la même relation, mais on modifie le nom d'un attribut.
$\rho_{A_i/B}(R)$ est la relation R avec A_i renommé B, donc de schéma
$\{A_1 : D_1, \dots, A_{i-1} : D_{i-1}, B : D_i, A_{i+1} : D_{i+1}, \dots, A_n : D_n\}$.
SQL : SELECT $A_1, \dots, A_{i-1}, A_i$ *AS* $B, A_{i+1}, \dots, A_n$ FROM R
On peut ainsi renommer plusieurs attributs d'un coup.

Exemple : SELECT id AS ideleve,nom,prenom,classe,atelier FROM eleve

AS permet également de renommer une table, par exemple pour la dupliquer.

Exemple : SELECT e1.id,e2.id FROM eleve AS e1 JOIN eleve AS e2 ON e1.atelier = e2.atelier WHERE e1.id < e2.id renvoie les couples d'élèves inscrits dans le même atelier.

3.4 Produit cartésien

Il s'agit de l'opération mathématique habituelle.

$$R \times R' = \{(t, s) / \ t \in R, s \in R'\}$$

Pas grand intérêt si on s'arrête là, car de nombreux couples n'ont pas de sens, mais cette notion va nous permettre de définir les jointures.
SQL : SELECT $*$ FROM R, R'

Exemple : SELECT $*$ FROM eleve,atelier

3.5 Jointure

Une jointure est la composée d'un produit cartésien et d'une sélection. Elle sert à « coller » deux tables ayant des éléments en commun.

$$R \bowtie_C R' = \sigma_C(R \times R')$$

Pour nous, le critère de sélection sera toujours une égalité entre deux clés :

$$R \bowtie_{R.A_i = R'.A'_j} R' = \{(t,s) \in R \times R' / t_i = s_j\}$$

SQL : SELECT $*$ FROM R, R' WHERE $R.A_i = R'.A'_j$

Cette requête colle à la théorie, mais n'est plus utilisée depuis 1992. On utilise maintenant :

SELECT $*$ FROM R JOIN R' ON $R.A_i = R'.A'_j$

En pratique, les deux requêtes sont équivalentes, mais l'utilisation de JOIN permet de mettre en évidence l'opération de jointure, ce qui simplifie la compréhension du code et la maintenance. JOIN offre de plus d'autres possibilités hors programme (INNER JOIN, LEFT OUTER JOIN, …).

Exemple : SELECT $*$ FROM eleve JOIN atelier ON eleve.atelier = atelier.id

Cette requête renvoie un « collage » des tables eleve et atelier en faisant correspondre à chaque élève l'atelier qu'il a choisi.

Remarque : l'attribut eleve.atelier est une clé étrangère, un attribut qui permet d'identifier de manière unique chaque enregistrement (clé) d'une autre table (étrangère).

3.6 Union, intersection, différence

Il s'agit des opérations mathématiques habituelles portant sur des relations ayant le même schéma. On peut réunir deux tables eleve1 et eleve2, ou deux tables atelier1 et atelier2, mais pas une table eleve et une table atelier.

$$R \cup R' = \{t / \ t \in R \ ou \ t \in R'\}$$
$$R \cap R' = \{t / \ t \in R \ et \ t \in R'\}$$
$$R - R' = \{t \in R \ / \ t \notin R'\}$$

SQL : SELECT $*$ FROM R
UNION
SELECT $*$ FROM R'

Certains environnements SQL proposent les commandes INTERSECT et EXCEPT ou MINUS, d'autres pas. Dans ce cas, on les simule à l'aide de sélections ou de jointures.

Exemple : On dispose de deux tables atelier_matin et atelier_aprem contenant chacune la liste des élèves inscrits identifiés par l'attribut nom. On suppose qu'il n'y a pas d'homonyme. On obtient les noms des élèves inscrits le matin et l'après-midi avec :

```
SELECT nom FROM atelier_matin
INTERSECT
SELECT nom FROM atelier_aprem
```

Si on ne dispose pas de la fonction INTERSECT, on utilise :

```
SELECT atelier_matin.nom FROM atelier_matin JOIN atelier_aprem
    ON atelier_matin.nom = atelier_aprem.nom
```

Pour obtenir les noms des élèves inscrits seulement le matin, on utilise :

```
SELECT nom FROM atelier_matin WHERE nom NOT IN
 (SELECT nom FROM atelier_aprem)
```

3.7 Division cartésienne

La division cartésienne de R par R' nécessite que le schéma de R' soit inclus strictement dans celui de R. On a alors $R/R' = \{s / \ \forall t \in R', (t,s) \in R\}$.
En notant A l'ensemble des attributs de R et A' l'ensemble des attributs de R' et $T = \pi_{A-A'}(R)$, on a $R/R' = T - \pi_{A-A'}(T \times R' - R)$.

3.8 Fonctions d'agrégation

Ces fonctions permettent d'effectuer les opérations que l'on devine sur un ensemble d'enregistrements. Dans la suite, F désigne une des fonctions AVG, COUNT, MAX, MIN, SUM.
AVG et SUM s'appliquent uniquement à des données numériques.

COUNT(∗) renvoie le nombre de lignes dans la table, COUNT(*A*) renvoie le nombre de lignes dans la colonne *A* dont le contenu n'est pas NULL.

3.8.1 Utilisation simple

SQL : SELECT $F(A_i)$ FROM *R*

Exemple : SELECT AVG(nbinscrit) FROM atelier
renvoie le nombre moyen d'inscrits par atelier.

3.8.2 Utilisation avec GROUP BY

GROUP BY permet de regrouper les enregistrements ayant une valeur commune (ou plusieurs) avant d'utiliser une fonction d'agrégation sur chaque groupe.

SQL : SELECT $F(A_i)$ FROM *R* GROUP BY A_j

Exemple : SELECT nom, COUNT(nom) FROM eleve GROUP BY nom
renvoie le nombre d'occurrences de chaque nom dans la table

3.8.3 Utilisation avec GROUP BY … HAVING

HAVING permet de sélectionner des groupes vérifiant une condition exprimée à l'aide de fonctions d'agrégation.

SQL : SELECT $F(A_i)$ FROM *R* GROUP BY A_j HAVING *condition*

Exemple : SELECT nom, COUNT(nom) FROM eleve GROUP BY nom HAVING COUNT(nom) >= 2

Remarque : ces fonctions s'utilisent uniquement après SELECT ou après HAVING, jamais après WHERE. Ainsi, pour obtenir l'atelier ayant le plus d'inscrits, on utilisera :

SELECT nom FROM atelier WHERE nbinscrit =
(SELECT MAX(nbinscrit) FROM atelier)

Exercice : écrire la requête renvoyant l'atelier avec le plus de places libres.

3.9 Compléments

SELECT DISTINCT supprime les doublons
'SELECT DISTINCT nom FROM table' va afficher tous les noms distincts.
'SELECT nom FROM table GROUP BY nom' aura le même effet.
On peut utiliser DISTINCT à l'intérieur de Count :
'SELECT Count(DISTINCT nom) FROM table' est different de
'SELECT count(*) FROM table GROUP BY nom' qui va afficher le nombre d'occurrences de chaque nom.

LIMIT *n* à la fin d'une requête renvoie uniquement les *n* premiers résultats.
LIMIT *p*, *n* renvoie *n* résultats à partir du (*p*+1)-ième.

ORDER BY à la fin d'une requête permet de trier suivant les attribut(s) dans l'ordre croissant (ASC ou par défaut) ou décroissant (DESC).

Exemple : SELECT * FROM eleve ORDER BY classe, nom DESC

EXISTS (requête) renvoie True si la requête contient des enregistrements, False sinon et s'utilise de la manière suivante :
(requête 1) WHERE EXISTS (requête 2)
La requête 1 ne renvoie que les enregistrements pour lesquels la requête 2 contient des enregistrements.
(requête 1) WHERE NOT EXISTS (requête 2)
Ici, la requête 1 ne renvoie que les enregistrements pour lesquels la requête 2 n'en contient pas.

Exemple : SELECT * FROM atelier WHERE EXISTS (SELECT * FROM eleve WHERE eleve.atelier = atelier.id)

Comme dans l'exemple, les tables sélectionnées dans la requête 1 peuvent être utilisées dans la requête 2.

Un bon site d'initiation et d'entrainement : http://sqlzoo.net