Bases de données

Un système de gestion de bases de données est un système permettant de traiter un grand nombre d'informations. La base de données peut alors être complétée et modifiée en évitant les incohérences. L'architecture trois tiers (utilisateur, serveur de traitement, serveur de bases de données) permet l'utilisation de la base de données par plusieurs utilisateurs simultanément depuis des postes informatiques différents.

1 Modèle relationnel

1.1 Tables, attributs, schéma

Considérons l'exemple des données d'une bibliothèque, on souhaite en particulier conserver une trace de tous les emprunts de documents.
Les différentes informations concernant un objet concret ou abstrait sont enregistrées dans des **tables** ou **relations**.
Dans un premier temps, on considère les entités (objet concret ou abstrait à propos duquel on souhaite conserver des informations) suivantes : `Document` : les livres, `Auteur` : les auteurs et `Emprunteur` : les clients de la bibliothèque.
Les informations attachées à chaque objet sont appelés les **attributs** ou **colonnes**.
Les attributs de la table `Document` sont : titre, auteur, genre, date de parution, nombre de pages.
Le **domaine d'un attribut** est l'ensemble des valeurs que peut prendre un attribut.
Pour la table `Document`, les attributs titre, auteur et genre sont des chaînes de caractères ; date et nombre de pages sont des entiers.
Le nom d'une table, ses attributs et les domaines des attributs sont donnés dans le **schéma relationnel** de la table sous la forme suivante :
Nom_table{attribut1(domaine1), attribut2(domaine2), ... }
Le schéma de la table Document est : Document{titre(string), auteur(string), genre(string), date de parution(entier), nombre de pages(entier)}
Chaque description d'un objet dans une table est appelée **enregistrement** ou **ligne**.
Par exemple pour la table Document :

Document				
titre	auteur	genre	parution	pages
La cousine Bette	Honoré de Balzac	Roman	1846	240
Le feu	Henri Barbusse	Roman	1916	435
De la guerre	Carl von Clausewitz	Traité	1832	240
Mrs Dalloway	Virginia Woolf	Roman	1925	240
Cyrano de Bergerac	Edmond Rostand	Théâtre	1897	280
...	...	...	...	...

Remarque :
On retrouve la notion de **relation** vue en mathématiques xRy, étendue ici à un nombre n d'éléments : $R(x_1, x_2, \cdots, x_n)$.
Une table T définit une relation R_T par :

$$R_T(x_1, \ldots, x_n) \Leftrightarrow (x_1, \ldots, x_n) \text{ est un enregistrement la table } T.$$

1.2 Clé primaire

Une **clé** d'une table est un ensemble minimal d'attributs (une partie du schéma) dont la valeur caractérise chaque enregistrement de la table. Parmi les différentes clés possibles (appelées clé candidates), on en choisit une appelée **clé primaire**. Les attributs de la clé primaire sont soulignés dans le schéma relationnel de la table.
Pour la table Document :
Document{titre(string), auteur(string), genre(string), date de parution(date), nombre de pages(entier)}
Pour la table Emprunteur, l'attribut nom seul ne suffit pas, Nom, Prénom non plus.
S'il n'y a pas de clé naturelle, il possible d'ajouter un attribut (souvent appelé identifiant) qui sera la clé primaire, cet attribut prendra une valeur entière, certains systèmes de gestion de bases de données permettent d'incrémenter automatiquement l'attribut choisi comme clé.
Pour la table Emprunteur :
Emprunteur{id(entier), Nom(string), Prénom(string), adresse(string), téléphone(entier), mail(string)}
De même : Auteur{nom(string), prénom(string), date(entier), biographie(string)}

1.3 Associations, clé secondaire

Une **association** est un lien entre deux entités.
Par exemple :

- chaque livre est lié à un auteur, association : "écrit par"
- un emprunteur est lié à des documents, association : "emprunté par".

Les associations entre deux entités E_1 et E_2 peuvent être de type :

- $1-1$: chaque élément de E_1 est lié à au plus un élément de E_2 et réciproquement ;
- $1-*$: quitte à échanger E_1 et E_2, chaque élément de E_1 est lié à au plus un élément de E_2, mais un élément de E_2 peut être associé à plusieurs éléments de E_1 ;
- $*-*$: chaque élément de E_1 peut être lié à plusieurs de E_2 et réciproquement.

Dans le cas d'une association $1-*$ ou $1-1$, elle peut être représentée par un attribut dans l'une des tables.
L'association "écrit par" est de type $1-*$: chaque livre a un seul auteur, mais un auteur peut avoir écrit plusieurs livre. Cette association est représentée par l'attribut `auteur` de la table `Document`.
Pour la relation "emprunté par" est également de type $1-*$, mais on veut avoir des informations complémentaires comme la date d'emprunt et la date de retour prévue. On crée alors une nouvelle table `Emprunt` pour représenter cette association, dont le schéma est :
`Emprunt` : { idEmprunteur(entier), idDocument(entier), nom_auteur(string), dateEmprunt(string), dateRetour(string) }.
On appelle **clé étrangère** d'une table un ensemble d'attributs qui font partie de la clé primaire d'une autre table.
Par exemple, dans la table `Document` l'attribut auteur est une clé étrangère : c'est la clé primaire de la table `Auteur`. Dans la table `Emprunt` : id_emprunteur est une clé étrangère (clé primaire de `Emprunteur`) et {titre, auteur} est une autre clé étrangère (clé primaire de `Document`).

1.4 Un exemple d'association $*-*$

On est parti du principe qu'un document n'a qu'un seul auteur qui peut être caractérisé par son nom. Ce qui n'est pas très réaliste. Si l'on veut pouvoir indiquer plusieurs auteurs pour un même document, l'association "écrit par" est alors de type $*-*$.
Pour la représenter, on peut passer par une table `ecrit_par`. Avant cela on change de modèle pour la table Document{idDocument(entier), titre(string), Auteur(string), genre(string), date de parution(string), genre(string),nombre de pages(entier)} et
Auteur{idAuteur(entier), nom(string), prénom(string), date(string), biographie(string)}.
On propose alors le schéma suivant pour `ecrit_par` : {idDocument, idAuteur}.

2 Langage SQL

Le langage utilisé est une implémentation du SQL (Structured Query Language).

2.1 Pour utiliser une base de données

On commence toujours par renseigner la base de données sur laquelle on travaille :
pour travailler avec une base de données existante

```
USE <database_name>;
```

Par exemple :

```
.
```

2.2 Requête simple

Pour récupérer tous les attributs de tous les enregistrements d'une table

```
SELECT * FROM <table_name>;
```

Par exemple :

```
.


.
```

```
+----+-----------+-------------+---------------------+------------+-----+
| id | nom       | prénom      | adresse             | telephone  | mail|
+----+-----------+-------------+---------------------+------------+-----+
|  1 | Martin    | Jacques     | 11 avenue de Wagram |  123456789 | ... |
|  2 | Lagaffe   | Gaston      | Boulevard Pachéco   |  381000000 | ... |
|  3 | Martin    | Jean-Pierre | Toulon              | 2147483647 | ... |
+----+-----------+-------------+---------------------+------------+-----+
```

2.3 Projection

Pour récupérer certains attributs de tous les enregistrements d'une table

```
SELECT <column_name,...> FROM <table_name>;
```

Par exemple

```
.


.
```

```
+-----------+-------------+
| nom       | prénom      |
+-----------+-------------+
| Martin    | Jacques     |
| Lagaffe   | Gaston      |
| Martin    | Jean-Pierre |
+-----------+-------------+
```

2.4 Sélection

Pour récupérer tous les attributs de certains enregistrements d'une table

```
SELECT * FROM <table_name> WHERE <condition>;
```

Par exemple

```
.



.
```

```
+----+-----------+-------------+--------------------+------------+-----+
| id | nom       | prénom      | adresse            | telephone  | mail|
+----+-----------+-------------+--------------------+------------+-----+
|  1 | Martin    | Jacques     | 11 avenue de Wagram |  123456789 | ... |
|  3 | Martin    | Jean-Pierre | Toulon             | 2147483647 | ... |
+----+-----------+-------------+--------------------+------------+-----+
```

Pour créer les conditions de sélection, on peut utiliser les opérateurs : `+`, `-`, `*`, `/` sur les entiers et les flottants, `=`, `<>`, `<`, `<=`, `>`, `>=` sur les entiers, flottants et chaînes de caractères (ordre lexicographique) et `AND`, `OR`, `NOT` sur le booléens.
Projections et sélections peuvent être combinées pour récupérer certains attributs de certains enregistrements d'une table

```
SELECT <column_name...> FROM <table_name> WHERE <condition>;
```

Par exemple

```
.


.
```

```
+-----------+-------------+------------+
| nom       | prénom      | telephone  |
+-----------+-------------+------------+
| Martin    | Jacques     |  123456789 |
| Martin    | Jean-Pierre | 2147483647 |
+-----------+-------------+------------+
```

2.5 Renommage

Pour renommer un attribut :

```
SELECT <column_name> AS <nouveau_nom> FROM <table_name>;
```

Par exemple

```
.



.
```

```
+-----------+-------------+
| NOM       | Prénom      |
+-----------+-------------+
| Martin    | Jacques     |
| Lagaffe   | Gaston      |
| Martin    | Jean-Pierre |
+-----------+-------------+
```

2.6 Mots-clés

DISCTINCT : permet de supprimer les doublons.

```
SELECT DISTINCT <column_name,...> FROM <table_name>;
```

Par exemple

```
SELECT nom FROM Emprunteur;
```

```
+-----------+
| nom       |
+-----------+
| Martin    |
| Lagaffe   |
| Martin    |
+-----------+
```

```
SELECT DISTINCT nom
   FROM Emprunteur;
```

```
+-----------+
| nom       |
+-----------+
| Martin    |
| Lagaffe   |
+-----------+
```

ORDER BY permet de trier les résultats suivant la valeur d'une colonne dans l'ordre croissant (on peut préciser `ORDER BY ... ASC`), si l'on veut trier par ordre décroissant on utilisera `ORDER BY ... DESC`.

```
SELECT nom, prénom FROM Emprunteur ORDER BY nom;
```

```
+-----------+-------------+
| nom       | prénom      |
+-----------+-------------+
| Lagaffe   | Gaston      |
| Martin    | Jacques     |
| Martin    | Jean-Pierre |
+-----------+-------------+
```

On peut trier sur plusieurs critères, par exemple `ORDER BY nom, prénom` permet de trier suivant le nom, et ensuite sur le prénom si le nom est le même.

LIMIT : permet de spécifier le nombre maximum de résultats que l'on souhaite obtenir.

```
SELECT nom, prénom
FROM Emprunteur
ORDER BY nom LIMIT 5;
```

```
+-----------+-------------+
| nom       | prénom      |
+-----------+-------------+
| Lagaffe   | Gaston      |
| Martin    | Jacques     |
| Martin    | Jean-Pierre |
+-----------+-------------+
```

```
SELECT nom, prénom
FROM Emprunteur
ORDER BY nom LIMIT 2;
```

```
+-----------+-------------+
| nom       | prénom      |
+-----------+-------------+
| Lagaffe   | Gaston      |
| Martin    | Jacques     |
+-----------+-------------+
```

OFFSET associé à `LIMIT` permet de spécifier le rang de la première ligne à prendre en compte, **en comptant à partir de 0**. Cela revient à sauter le nombre de lignes indiqué avec le mot-clé `OFFSET`.

Par exemple :

```
SELECT nom, prénom
FROM Emprunteur
ORDER BY nom
LIMIT 1 OFFSET 2;
```

Donne au plus 1 ligne en sautant les 2 premières :

```
+-----------+-------------+
| nom       | prénom      |
+-----------+-------------+
| Martin    | Jean-Pierre |
+-----------+-------------+
```

3 Opérateurs ensemblistes

3.1 Produit cartésien

Le **produit cartésien** de deux relations $\mathscr{R}$ et $\mathscr{S}$ est la relation $\mathscr{T}$ ayant pour schéma la concaténation des schéma de $\mathscr{R}$ et $\mathscr{S}$ et dont les enregistrements sont ceux de $\mathscr{R} \times \mathscr{S}$: concaténation des enregistrements de $\mathscr{R}$ et de $\mathscr{S}$. On note $\mathscr{T} = \mathscr{R} \times \mathscr{S}$

Réalisation SQL :

```
SELECT * FROM R,S
```

3.2 Union

Deux relations sont dites **union-compatibles** lorsqu'elles ont le même schéma relationnel (donc les mêmes attributs avec les mêmes domaines).
L'**union** deux relations $\mathscr{R}$ et $\mathscr{S}$ union-compatibles est une relation $\mathscr{T}$ de même schéma que $\mathscr{R}$ et $\mathscr{S}$ et dont les enregistrements sont ceux qui sont dans $\mathscr{R}$ ou dans $\mathscr{S}$.

Réalisation SQL :

```
SELECT ...
UNION
SELECT ...
```

3.3 Intersection

L'**intersection** de deux relations union-compatibles $\mathscr{R}$ et $\mathscr{S}$ est la relation $\mathscr{T}$ de même schéma que $\mathscr{R}$ et $\mathscr{S}$ et dont les enregistrements sont ceux qui sont à la fois dans $\mathscr{R}$ et dans $\mathscr{S}$.

```
SELECT ...
INTERSECT
SELECT ...
```

Remarque : n'est pas supporté par tous les systèmes de gestions de bases de données.

3.4 Différence

La **différence** entre deux relations union-compatibles $\mathscr{R}$ et $\mathscr{S}$ est la relation $\mathscr{T}$ de même schéma que $\mathscr{R}$ et $\mathscr{S}$ et dont les enregistrements sont ceux qui sont dans $\mathscr{R}$ sans être dans $\mathscr{S}$.

```
SELECT ...
EXCEPT
SELECT ...
```

Remarque : n'est pas supporté par tous les systèmes de gestions de bases de données.

3.5 Jointure

La **jointure** de deux relations $\mathscr{R}$ et $\mathscr{S}$ suivant une formule de sélection P est la relation $\mathscr{T}$ dont le schéma est la concaténation des schéma de $\mathscr{R}$ et $\mathscr{S}$ et dont les enregistrements sont la concaténation des enregistrements de $\mathscr{R}$ et $\mathscr{S}$ qui vérifient la condition P.
Réalisation SQL :

```
SELECT * FROM R JOIN S ON P
```

On utilise le plus souvent les clés étrangères pour faire la condition de jointure.
Si un attribut d'une table a le même nom qu'un attribut d'une autre table, on lève l'ambiguité avec la syntaxe suivante : `table.attribut`.
Par exemple `idDocument` apparait dans la table `Document` et dans la table `Emprunt`. On utilisera soit `Document.idDocument`, soit `Emprunt.idDocument` pour désigner ces attributs.
En reprenant l'exemple de la bibliothèque, pour éviter les redondance, on a réparti les informations sur plusieurs tables. Les jointures vont nous permettre de regrouper des informations issues de différentes tables.

```
+----+-----------+-------------+
| id | nom       | prénom      |
+----+-----------+-------------+
|  1 | Martin    | Jacques     |
|  2 | Lagaffe   | Gaston      |
|  3 | Liègeois  | Jean-Pierre |
|  4 | Colluci   | Michel      |
|  5 | Dupont    | Jean        |
+----+-----------+-------------+
+------------+-----------------------+--------------------+---------+
| idDocument | Titre                 | Auteur             | Genre   |
+------------+-----------------------+--------------------+---------+
|          1 | Ruy Blas              | Victor Hugo        | Roman   |
|          2 | Les fleurs du mal     | Charles Baudelaire | Poésie  |
|          3 | Don Quichotte         | Cervantès          | Roman   |
|          4 | Les lauriers de César | Gosciny            | BD      |
+------------+-----------------------+--------------------+---------+
+--------------+------------+-------------+------------+
| idEmprunteur | idDocument | DateEmprunt | DateRetour |
+--------------+------------+-------------+------------+
|            1 |          3 | 2013-05-02  | 2013-05-22 |
|            1 |          4 | 2013-05-04  | 2013-05-25 |
|            2 |          1 | 2013-06-08  | 2013-06-29 |
+--------------+------------+-------------+------------+
```

Pour obtenir les noms, prénoms des emprunteurs avec la date de retour prévue pour chaque document emprunté :

```
+---------+---------+------------+
| nom     | prénom  | DateRetour |
+---------+---------+------------+
| Martin  | Jacques | 2013-05-22 |
| Martin  | Jacques | 2013-05-25 |
| Lagaffe | Gaston  | 2013-06-29 |
+---------+---------+------------+
```

Pour obtenir en plus les titres, information présente dans la table `Document`, on fait une deuxième jointure.

```
+---------+---------+-----------------------+-------------+
| nom     | prénom  | Titre                 | DateRetour  |
+---------+---------+-----------------------+-------------+
| Lagaffe | Gaston  | Ruy Blas              | 2013-06-08  |
| Martin  | Jacques | Don Quichotte         | 2013-05-02  |
| Martin  | Jacques | Les lauriers de César | 2013-05-04  |
+---------+---------+-----------------------+-------------+
```

Pour retrouver le numéro de téléphone de l'emprunteur de Ruy Blas :

```
.




.
```

3.6 Attention

Une requête `SELECT` renvoie une table, mais cette table n'est pas stockée dans la base de données. On ne peut pas la réutiliser dans une deuxième requête. Pour cela on utilise les requêtes imbriquées.

4 Fonctions agrégatives

4.1 Syntaxe générale

Modèle complet (nous concernant) d'une requête `SELECT` avec options disponibles :

```
SELECT [DISTINCT]
    <select_expression,...>
    FROM <table_references>
      [WHERE <condition> ]
      [GROUP BY <att> ]
      [HAVING <condition>]
      [ORDER BY <att> [ASC | DESC] ,...]
      [LIMIT <n> [OFFSET <p>]]
```

On remplace `<select_expression,...>` par le nom des attributs (projection) éventuellement renommés (`AS`) et/ou des fonctions agrégatives : `COUNT`, `MIN`, `MAX`, `...`
et `<table_references>` par une table éventuellement renommée ou des tables jointées.
L'option `WHERE` permet de faire une sélection et l'option `DISTINCT` permet de supprimer les doublons.
Quelques fonctions agrégatives (`<att>` étant à remplacer par le nom d'un attribut) :

- `COUNT(*)` ou `COUNT(<att>)` : nombre de d'enregistrements
- `COUNT(DISTINCT <att>)` : nombre de valeurs distinctes prises par l'attribut `<att>` ;
- `SUM(<att>)` : calcule la somme des valeurs de l'attribut `<att>`
- `MIN(<att>)` et `MAX(<att>)` : minimum et maximum
- `AVG(<att>)` : moyenne.

4.2 L'option `GROUP BY`

Appliquées sans l'option `GROUP BY` les fonctions agrégatives s'appliquent à l'ensemble des enregistrements de la table. On peut les appliquer à des groupements d'enregistrements ayant la même valeur pour un attribut, la fonction s'applique alors sur chaque groupement.

Exemple :

pour connaître le nombre d'habitants sur chaque continent à partir de la table `Pays`, on peut utiliser la requête :

```
.


.
```

Attention : ne pas demander la valeur d'un attribut qui n'est pas commune à tous les enregistrements d'un groupement.
Remarque : L'option `ORDER BY <att>` permet de trier l'ordre dans lequel est affiché le résultat de la requête suivant la valeur de l'attribut ou du résultat de la fonction agrégative `<att>`, par défaut l'ordre est croissant (`ASC`) et peut être remplacé par décroissant (`DESC`).

4.3 Filtrage des agrégats avec HAVING

Le résultat d'une fonction d'agrégation ne peut pas servir dans une condition de sélection (`WHERE`). Pour filtrer suivant des agrégats on utilise `HAVING`.
Par exemple : pour ne tenir compte que des continents dont la population n'est pas nulle :

```
.



.
```

5 Requêtes imbriquées

Dans une condition de sélection introduite par `WHERE`, on compare la valeur d'un attribut à une valeur de référence qui peut être : un attribut, une expression ou une constante. Mais pas une fonction agrégative (`SUM`, `COUNT` etc.).
Si l'on veut comparer la valeur d'un attribut au résultat d'une requête, on aurait envie d'enregistrer ce résultat. On procéderait de la manière suivante pour obtenir les villes de la région de Paris :

```
CirconscriptionParis = SELECT Circonscription FROM Ville WHERE Nom = "Paris";
SELECT Nom FROM Ville WHERE Circonscription = CirconscriptionParis;
```

Mais nous avons vu que les systèmes de gestion de bases de données (comme MySQL) ne le permettent pas. Le langage SQL offre la possibilité d'utiliser le résultat d'une requête `SELECT ...` dans une comparaison sous le modèle suivant :
. WHERE <attribut> <comparaison> (SELECT ...)
On appelle **requête imbriquée** ou **sous-requête** la requête intermédiaire : `SELECT ...`
La requête qui utilise ce résultat est la **requête principale**.
Attention : les parenthèses autour de la requête imbriquée sont obligatoires.

On distingue deux cas :

les requêtes qui renvoient une unique valeur : cette valeur est alors utilisée comme valeur de référence pour une comparaison classique (=, <, >, <=, >=, !=, etc.)
Cela donne dans l'exemple précédent :

```
.



.
```

les requêtes qui renvoient une colonne éventuellement vide : on a alors deux opérateurs

- `EXISTS` permet de tester si la requête imbriquée renvoie au moins un enregistrement.
  La condition s'écrit alors : `... WHERE EXISTS (SELECT ...);`
  ce n'est pas une comparaison avec la valeur d'un attribut.
- `IN` permet de tester si la valeur d'un attribut est un élément de la colonne obtenue par la requête imbriquée qui doit donc ne contenir qu'un seul attribut (mais possiblement plusieurs enregistrements).
  La condition s'écrit alors : `... WHERE <attribut> IN (SELECT ...);`

Ces deux opérateurs peuvent être combinés avec `NOT`.

Remarque : On peut même utiliser le résultat d'une requête `SELECT` comme table (dont on pourra utiliser plusieurs colonnes). Il faut alors lui donner un nom qui remplacera `<NomTableIntermediaire>` dans :

```
SELECT... FROM (SELECT ...) <NomTableIntermediaire> [...] ;
```

Exemples :

1. Une requête qui donne le nom et la population de la ville la plus peuplée du monde.

   ```
   SELECT Nom, Population FROM Ville WHERE Population = MAX(Population);
   ```

   ne convient pas : pas de fonction agrégative dans une condition.

   ```
   SELECT Nom, MAX(Population) FROM Ville;
   ```

   ne convient pas : Nom n'est pas commun à l'agrégat.
   en deux fois :

   ```
   .

   .
   ```

   puis après lecture du résultat

   ```
   .

   .
   ```

   on en déduit la solution avec requête imbriquée :

   ```
   .


   .
   ```

   Une autre solution par tri et limitation :

   ```
   .


   .
   ```

2. Une requête qui donne les villes du même Circonscription (région) que Paris.

   ```
   .


   .
   ```

   ou par autojointure :

   ```
   .


   .
   ```