Base de données

Cours 01

En informatique, il est important de manipuler de nombreuses données de manière rapide et efficace. La plupart des langages disposent de méthodes pour ce faire, plus ou moins efficaces : un langage s'est imposé ces dernières décennies, SQL. L'objectif principal cette année sera d'étudier les *requêtes* en base de données : vous disposez de nombreuses informations, comment faire pour les récupérer ?

1 Définitions

Définition 1

Une **relation** (ou table) est une table assimilable à une matrice dont les lignes correspondent à des **enregistrements**. Chaque enregistrement dispose de plusieurs **attributs**, correspondant aux colonnes de la relation.
Une base de données est un ensemble de relations.

Ci-après se trouvent deux relations, formant ensemble une base de données. La première relation dispose de 8 enregistrements, la seconde de 12.

La relation ci-dessous, appelée `Auteurs`, recense plusieurs écrivains ; la seconde, appelée `Livres`, recense des livres.

prenom	**nom**	**age**	**ville**
William	Shakespeare	52	Londres
Anna	Akhmatova	76	Moscou
Octavio	Paz	84	Mexico
Toni	Morrison	88	New York
Jane	Austen	41	Winchester
Yukio	Mishima	45	Tokyo
Roger	Munier	86	Vesoul
Harold	Morrison	62	Springfield

auteur	**annee**	**titre**
Shakespeare	1606	Le Roi Lear
Shakespeare	1623	Hamlet
Akhmatova	1921	Le plantain
Mishima	1949	Confession d'un masque
Austen	1816	Emma
Akhmatova	1963	Requiem
Mishima	1956	Le Pavillon d'Or
Paz	1963	L'Arc et la Lyre
Austen	1813	Orgueil et Préjugés
Morrison	1973	Sula
Paz	1950	Le Labyrinthe de la Solitude
Munier	1989	Requiem

Dans la première table, les attributs sont `prenom`, `nom`, `age` et `ville`. Si on voulait rajouter un auteur, il devrait avoir ces trois caractéristiques (en particulier, il devrait être mort). Dans la seconde table, les attributs sont `auteur`, `annee` et `titre`. On remarque que les attributs `nom` de `Auteurs` et `auteur` de `Livres` sont très similaires.

Un attribut a généralement un **type** associé : par exemple, dans `Auteurs`, un `age` sera toujours un entier positif. De même, un prénom est toujours une chaîne de caractères.

On peut aussi préciser une plage de valeurs possibles : c'est le **domaine** d'un attribut. Par exemple, ici, l'âge sera toujours compris entre 1 et 100. L'intérêt est qu'ainsi, il est plus facile de stocker la base de données, en quantifiant régulièrement l'espace nécessité par les valeurs à remplir. Il est aussi usuel, dans le cas de chaînes de caractères, de donner une limite à la taille du texte : ici par exemple, on pourrait fixer qu'un prénom ne peut contenir plus de 50 caractères.

Définition 2 : clé primaire

Pour une relation, une **clé primaire** est un ensemble d'attributs qui permettent de définir de manière unique toute entrée.

Dans notre exemple, dans `Auteurs`, `age` est une clé primaire : il n'y a pas, dans notre base, deux auteurs morts au même âge. On remarque que `nom` n'est pas une clé primaire : Toni Morrison et Harold Morrison ont le même nom. On pourrait utiliser comme clé primaire `(prenom,nom)`.

Il est très, **très** usuel d'avoir un attribut utilisé tout spécialement pour servir de clé primaire. Il s'agit généralement d'une colonne appelée `id` (pour « identifiant »). Il faut alors faire très attention : l'id d'une table n'est pas l'id d'une autre.

Définition 3 : schéma d'une relation

Le schéma relationnel d'une relation est la liste de ses attributs.

Remarque. On distinguera, plus tard, le schéma relationnel d'une *relation* et d'une *base de données*.

Le schéma d'une relation peut être complété par les types et domaines des différents attributs. Par exemple, `Auteurs` a pour schéma `(prenom : string, nom : string, age : int, ville : string )`.

Il est usuel de souligner la clé primaire : on a alors le schéma (prenom, nom, age, ville)

2 Requêtes simples

L'une des tâches fondamentales du SQL est d'effectuer des requêtes (d'où le nom : *Structured **Query** Language*) : à partir de nos bases de données, comment récupérer l'information qui nous intéresse ?

2.1 SELECT ...FROM ...

De manière générale, il est usuel (mais non nécessaire !) d'écrire les mots-clés de SQL en majuscule.

La commande `SELECT` est la commande essentielle, permettant de récupérer des informations. Le mot-clé `FROM` permet de préciser la table où récupérer de l'information.

> `SELECT prenom FROM Peintres` renvoie
> `[Claude,Gustav,Frida,Marc,Salvador,Tamara,Vassily]`.

Remarque. SQL renvoie en fait plutôt une nouvelle base de données avec les informations demandées : on se fiche un peu du format précis en sortie (qui dépend fortement du logiciel utilisé pour faire tourner SQL).

> `SELECT ville FROM Musees` renvoie
> `[Paris, New York, New York, New York, Paris]`.

S'il y a des doublons, les doublons sont renvoyés. Le mot-clé `DISTINCT` permet de ne pas avoir de doublons.

> `SELECT DISTINCT ville FROM Musees` renvoie
> `[Paris, New York]`.

On peut sélectionner des tuples : `SELECT prenom, nom FROM Peintres` renvoie
`[(Claude,Monet),(Gustav,Klimt),(Frida,Kahlo),…]`.

2.2 WHERE ...

Plus intéressant, on peut préciser des conditions sur ce qu'on veut renvoyer. Le mot-clé pour exiger une condition est `WHERE`.

> `SELECT nom FROM Peintres WHERE annee_naissance = 1904` renvoie
> `[Dali]`.

> `SELECT nom FROM Peintres WHERE annee_naissance >= 1900` renvoie
> `[Kahlo,Dali]`.

Les conditions à préciser peuvent s'écrire à l'aide de `=`, `<>` (différent de), `<`, `<=`, `AND`, `OR`, `NOT`.

Question. Écrire une requête SQL renvoyant les noms des peintres nés entre 1850 et 1900.

Question. Écrire une requête SQL renvoyant toutes les informations des musées situés à New York.

2.3 `ORDER BY ...`

Il est possible d'ordonner les résultats obtenus selon un critère en écrivant `<requete> ORDER BY attribut`.

> `SELECT nom FROM Peintres ORDER BY annee_naissance` renvoie
> `[Monet, Klimt, Kandinsky, Chagall, de Lempicka, Dali, Kahlo]`.

Le mot-clé `DESC` permet d'avoir les informations dans l'ordre décroissant ; la syntaxe est `SELECT nom FROM Peintres ORDER BY annee_naissance DESC`.

On peut parfaitement combiner une condition `WHERE` et un `ORDER BY` : d'abord la condition, puis l'ordre.

Question. Écrire une requête SQL renvoyant les noms des tableaux de Chagall dans l'ordre croissant des années.

2.4 `LIMIT ..., OFFSET ...`

Parfois, on ne veut que les 2 premiers éléments : on peut alors utiliser le mot-clé `LIMIT`. L'ordre choisi est par défaut celui de la relation (qui correspond, le plus souvent, à l'ordre selon la clé primaire).

> `SELECT nom FROM Peintres LIMIT 3` renvoie
> `[Monet, Klimt, Kahlo]`.

Le mot-clé `OFFSET` permet, combiné à un `LIMIT`, d'exclure les premiers éléments.

> `SELECT nom FROM Peintres LIMIT 3 OFFSET 1` renvoie
> `[Klimt, Kahlo, Chagall]` : on offset Monet, puis on récupère les 3 premiers suivants.

3 Fonctions d'agrégats

3.1 `MAX`, `MIN`, `SUM`, `AVG`, `COUNT`

Pour effectuer des études statistiques sur les bases de données, plusieurs fonctions sont déjà implémentées et standardisées : on les appelle **fonctions d'agrégat** (car elles agrègent les informations pour les étudier).

> `SELECT MAX(annee) FROM Oeuvres` renvoie
> 1939 : il s'agit de l'œuvre la plus tardive de la base de données.

Question. Écrire une requête SQL renvoyant l'année moyenne des œuvres de Kandinsky.

La fonction `COUNT` est légèrement différente : la valeur `COUNT(attribut)` comptera les éléments selon la colonne en question. Cela peut être utile dans certains contextes précis, mais la plupart du temps, on se fiche de l'attribut compté : on écrit donc le plus souvent `COUNT(*)` (l'étoile signifiant à SQL « compte, peu importe l'attribut »).

> `SELECT COUNT(*) FROM Oeuvres WHERE musee = 2` renvoie
> 5 : il y a 5 peintures dans notre BDD au Metropolitan.

3.2 `GROUP BY ...`

La dernière requête nous a permis de connaître le nombre d'œuvres au Metropolitan : mais on voudrait connaître le nombre d'œuvres dans *tous* les musées. Si on procède comme avant, il faut donc une requête par musée, c'est long.

Le mot-clé `GROUP BY` permet de découper une table en « sous-tables » selon un attribut commun ; en l'associant à une fonction d'agrégat, on peut alors faire des statistiques sur chacune des sous-tables.

> `SELECT COUNT(*) FROM Oeuvres GROUP BY musee` renvoie
> `[(1 : 1 , 2 : 5 , 3 : 1, 4 : 3, 5 : 5)]`

3.3 HAVING ...

Jusque-là, on peut faire pas mal de choses ...mais pas tout. En particulier, SQL contraint les conditions WHERE à avoir lieu avant les GROUP BY. Parfois, on veut cependant imposer une condition *après* avoir fait un GROUP BY : c'est ce à quoi sert HAVING.

`SELECT peintre, COUNT(*) FROM Oeuvres GROUP BY peintre HAVING COUNT(*) >=2;` renvoie `[Chagall : 3 , Dali : 2, Kandinsky : 3, Klimt : 2, Monet : 2, de Lempicka : 3]` : on calcule le nombre d'œuvres de chaque artiste, puis on renvoie ceux qui ont au moins 2 œuvres.

Le plus important est de retenir l'ordre suivant :

agrégats → WHERE → GROUP BY → HAVING

On veut les artistes et nombres d'œuvres desdits artistes ayant au moins 2 tableaux *qui ne sont pas affichés au Metropolitan*. On étudie la table `Oeuvres`, en se restreignant aux tableaux affichés en dehors du Metropolitan :
`SELECT ??? FROM Oeuvres WHERE musee <> 2`
Puis, on fait un agrégat pour avoir les artistes et leurs nombres d'œuvres :
`SELECT peintre, COUNT(*) FROM Oeuvres WHERE musee <> 2 GROUP BY peintre`
Enfin, on filtre pour n'avoir que ceux qui ont au moins deux peintures :
`SELECT peintre, COUNT(*) FROM Oeuvres WHERE musee <> 2 GROUP BY peintre HAVING COUNT(*) >= 2`

4 Bases de données à plusieurs relations

Dans le cas où une base de données contient plusieurs relations, décrire le schéma relationnel de chaque table n'est pas toujours suffisant pour saisir les différents liens entre les différentes tables. On utilise alors le modèle entité-association.

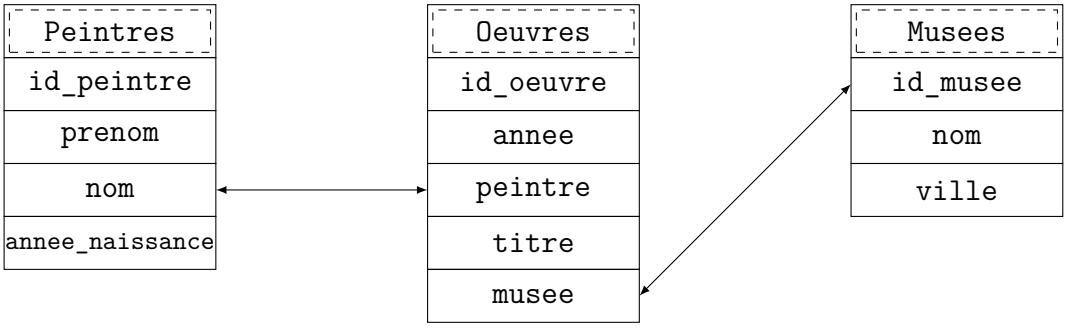
Définition 4 : schéma d'une base de données

Le schéma relationnel d'une base de données consiste en la liste des attributs de chaque table, ainsi qu'à l'association entre certains attributs de certaines tables.

Dans la table jouet donnée en annexe, on a les trois schémas de relations suivants :

- `Peintres : (id_peintre, prenom, nom, annee_naissance);`
- `Musees : (id_musee, nom, ville);`
- `Oeuvres : (id_oeuvres, annee, peintre, titre, musee).`

Mais ces listes ne suffisent pas à décrire complètement la base de données. En effet, certains attributs seront reliés, et on pourra utiliser les liens entre ces attributs pour écrire de futures requêtes : par exemple, l'attribut `peintre` dans la relation `Oeuvres` correspondra à des `nom` dans la relation `Peintres`. Pour représenter complètement cette table, on dessine généralement un diagramme entité association :



Ces flèches permettent de définir le concept de clé étrangère :

Définition 5 : clé étrangère

Pour deux relations R_1 et R_2, un attribut a_1 de R_1 et un attribut a_2 de R_2 définissent une clé étrangère de R_2 référençant a_1 si pour tout enregistrement de R_2, son attribut a_2 correspond à un enregistrement de R_1 tel que $a_1 = a_2$.

Ici, c'est le cas pour `Oeuvres.peintre` et `Peintres.nom` : chaque œuvre dans `Oeuvres` a un peintre, dont le nom apparaît dans `Peintres`. On dit que `Oeuvres.peintre` définit une clé étrangère de `Oeuvres` référençant `Peintres.nom` dans `Peintres`.

Le modèle entité-association peut être rendu plus précis en quantifiant les flèches.

Dans le schéma relationnel précédent, les flèches ont en fait « une direction » : chaque `nom` de peintre peut apparaître plusieurs fois dans la table `Oeuvres`. Pour une apparition dans `Peintres`, il y a plusieurs apparitions dans `Oeuvres` : c'est une association $1 - *$.
Question. Compléter la quantification de l'autre association du schéma précédent.

Il existe trois principales quantifications :

— $1 - *$: une apparition dans une table correspond à plusieurs dans l'autre ;

— $1 - 1$: une apparition dans une table correspond à une seule apparition dans une autre ;

— $* - *$: il y a des apparitions multiples d'un côté et de l'autre.

Vous gérez la base de données d'une association de jeux de plateau. Vous avez les données de joueurs et de jeux, ce qui donne le schéma suivant :

Vous voulez maintenant enregistrer les données permettant de savoir quel joueur possède quel jeu. Un même joueur peut posséder plusieurs jeu ; un même jeu peut être possédé par plusieurs joueurs : c'est une association $* - *$.

— si vous créez un attribut `jeu_possede` dans `Joueurs`, vous ne pourrez mettre qu'un seul jeu. Il n'est pas possible de mettre une liste comme attribut d'une relation.

— symétriquement, créer un attribut `possede_par` dans `Jeux` ne résout pas le problème.

— En multipliant les attributs, vous pourriez combler temporairement le défaut, mais vous imposeriez une limite au nombre de jeux possédés / de joueurs possédant un jeu particulier.

De manière générale, il n'est pas possible, stricto sensu, de décrire directement une association $* - *$ dans une base de données. La solution consiste à introduire une relation intermédiaire, qui permet de découper une association $* - *$ en deux associations $* - 1$ et $1 - *$. Dans notre exemple, cela consiste à créer une table `Achats`, qui décrit chaque couple `(id_joueur, id_jeu)` de possessions de jeux par des joueurs.

5 Requêtes imbriquées et jointures

5.1 Requêtes imbriquées

Parfois, il devient très compliqué d'écrire notre requête avec un seul `SELECT` (voire impossible). SQL autorise tout à fait à avoir des requêtes imbriquées, pourvu qu'elles aient du sens.

`SELECT musee, COUNT(*)/(SELECT COUNT(*) FROM Oeuvres) FROM Oeuvres GROUP BY musee` renvoie, pour chaque musée, le pourcentage total que chacun possède parmi toutes les œuvres.

Remarque. En fait, la division effectuée est entière; pour forcer SQL à faire une division flottante, on peut écrire `SELECT musee, COUNT(*)*(1.0)/(SELECT COUNT(*) FROM Oeuvres) FROM Oeuvres GROUP BY musee`.

5.2 `JOIN ...ON`

Il arrive régulièrement que la requête demandée nécessite de récupérer des informations dans plusieurs tables différentes. Une idée centrale est alors de joindre deux (ou plus!) tables ensemble en coordonnant des attributs communs.

`SELECT * FROM Oeuvres JOIN Musees ON musee = id_musee` renvoie

id_oeuvre	annee	peintre	titre	musee	id_musee	nom	ville
1	1872	Monet	Impression, soleil levant	1	1	Musée Marmottan	Paris
2	1899	Monet	Le bassin aux nymphéas	2	2	Metropolitan	New York
3	1907	Klimt	Portrait d'Adele Bloch-Bauer	3	3	Neue Galerie	New York
4	1923	Kandinsky	Composition VIII	4	4	Guggenheim	New York
5	1930	Chagall	Dans la neige	4	4	Guggenheim	New York
6	1931	Dali	La naissance des désirs liquides	4	4	Guggenheim	New York

Quelques remarques sur les jointures :

— Les attributs identifiés par le `ON` coexistent dans la jointure : c'est certes un peu coûteux, mais cela signifie aussi que les deux attributs existent *après* la jointure, sans que vous vous posiez la question duquel a disparu.

— Il est toujours mieux de s'assurer que les attributs des différentes tables sont tous différents. Si ce n'est pas le cas, pour désambigüiser, on précise l'attribut sous la forme `Table.attribut`; cette construction est toujours possible.

— Si la jointure se fait sur des attributs qui ne sont pas des clés primaires, la jointure contiendra *toutes* les combinaisons possibles qui remplissent la condition du `ON`.

5.3 Opérations ensemblistes sur les relations

Enfin, il est possible d'effectuer 4 opérations ensemblistes classiques sur les relations

— `UNION` et `INTERSECT` permettent de faire l'union et l'intersection de tables de mêmes schémas;

— `EXCEPT` permet de faire la différence entre deux tables de mêmes schémas;

— `CROSS JOIN` permet de faire le produit cartésien de deux tables, peu importe leurs schémas.

Seul EXCEPT est vraiment utile.