Cours - Base de données

Compétences :

- A - Interroger une base de données
- B - Connaître le vocabulaire élémentaire lié aux bases de données : table ou relation, attribut ou colonne, domaine, schéma d'une table, enregistrements ou lignes, types de données.
- C - Comprendre la notion de clé : identifier de manière unique une ligne au moyen d'une ou plusieurs colonnes.
- D - Comprendre la notion d'association un à un : 1-1, un à plusieurs : 1-*, plusieurs à plusieurs : *-* . Associer les clés primaires à la relation 1-1 et les clés étrangères à la relation 1-*.
- E - Établir l'union, l'intersection ou la différence de deux tables possédant le même schéma ;
- F - Notion d'agrégation. Mots clés à connaître : `MIN`, `MAX`, `SUM`, `AVG`, `COUNT` et `GROUP BY`
- G - Filtrer les agrégats (les groupes obtenus par agrégation). Mots clé : `HAVING`. On souligne la différence entre **la restriction** (mot clé : `WHERE`) et le **filtrage d'agrégat** (mot clé : `HAVING`).

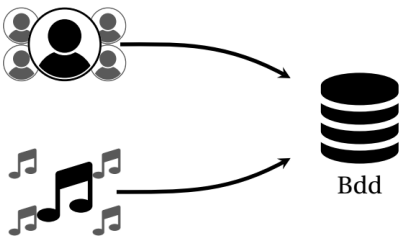
Buts et motivations

Avec l'avènement d'internet et des réseaux sociaux, les données sont devenues une ressource fondamentale. De nombreux logiciels, applications, ou sites internet stockent des informations en utilisant la structure d'une base de données (Pensez à vos photos ou vos vidéos instagram, votre playlist Spotify etc).
Pour ordonner, structurer et accéder à ces données, un pan entier de l'informatique et de l'ingénierie est utilisé. Dans ce contexte, les bases de données sont amenées à prendre une place de plus en plus prégnante dans l'ingénierie.

La base de donnée d'une application pour écouter de la musique :

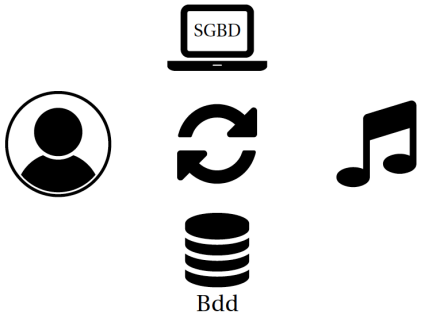
Dans ce cours, pour illustrer la structure d'une base de données relationnelle, on considère la structure simplifiée d'une application pour écouter de la musique (de type spotify, deezer...) qui propose à des utilisateurs d'écouter de la musique.
Dans cette base de donnée on trouvera donc entre autres :

- des musiques ;
- des utilisateurs.



Le SGBD fait l'interface avec la base de données :

Pour interroger la base de données, l'utilisateur doit utiliser un logiciel dédié aux base de données : c'est le SGBD. Il écrit alors ses instructions (on parle de requête) dans un langage compréhensible par le SGBD. Dans le cadre du cours ce sera le langage SQL. Le SGBD va alors récupérer les données dans la base de données et les renvoyer à l'utilisateur sous la forme d'un tableau. Le schéma ci-dessous illustre le rôle du SGBD qui fait le lien entre l'utilisateur et la base de données.



Le langage SQL

Le SQL *(Structured Query Language)* est le langage informatique dédié aux bases de données le plus répandu. C'est donc le langage qui permet de communiquer avec la base de données, par exemple :

- de lui demander d'accéder à certaines données,
- d'en ajouter,
- d'en modifier.

Le SQL a été initialement développé par deux informaticiens : Donald Chamberlin et Raymond Boyce qui travaillaient dans la société IBM.

Base de données relationnelle

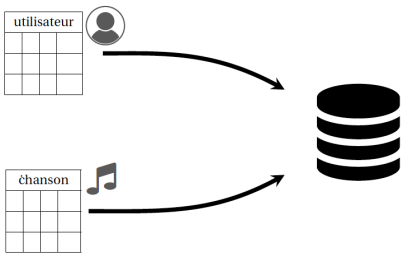
Dans le cours on s'intéresse à des bases de données dites **relationnelles**. Cela signifie que les données sont stockées sous la forme de tables. Dans le cas de notre application on aura donc plusieurs tables comme :

- une table utilisateur qui stocke l'ensemble des utilisateurs de l'application,
- une table chanson qui stocke l'ensemble des titres possibles,
- une table artiste qui stocke des données sur chaque artiste
- etc...

Les SGBD les plus courants

Les SGBD sont les logiciels qui permettent d'interroger une base de données. Parmi les plus répandus on trouve :

- MySQL
- Microsoft Access
- Oracle Database
- SQLite



I Structure des données

Pour structurer les données on les range sous la forme de tables aussi appelées relations (table =relation). Chaque table stocke des objets de même nature : dans l'exemple ci-contre une table stocke chaque utilisateur, une table stocke chaque chanson.
Une table possède des lignes et des colonnes. Chaque ligne d'une table représente un objet stocké dans la table : dans la table utilisateur chaque ligne représente un utilisateur. De même chaque ligne de la table chanson représente une chanson.
Dans le vocabulaire des bases de données on parle d' **enregistrement** pour une ligne, ou encore de tuple ou de vecteur (enregistrement = ligne = tuple = vecteur) et **d'attribut** pour une colonne (attribut = colonne). Dans l'exemple ci-contre les attributs de la table utilisateurs sont : id, nom, prenom, age, ville.
À chaque attribut correspond un **domaine**, c'est à dire un ensemble de valeurs possibles pour l'attribut (des entiers, des chaînes de caractères...). L'ensemble des colonnes d'une table est appelé le **schéma de la table**.

chanson

id	titre	artiste	duree
1	Beautiful	Eminem	392
2	Brothers in Arms	Dire Straits	424
3	The Real Slim Shady	Eminem	284
4	Marché Noir	SCH	210

la table chanson

utilisateur

id	nom	prenom	age	ville
1	Durand	Tom	19	Paris
2	Legrand	Charles	46	Lyon
3	Riolon	Marie	35	Bayonne

le schéma de la table

une ligne = un utilisateur

II Notion de clé

Dans une table, chaque ligne est **unique** : il ne peut pas y avoir de doublons, c'est à dire 2 lignes qui seraient les mêmes dans toutes les colonnes de la table. On cherche maintenant à retrouver une ligne (qui est unique) en utilisant un attribut (ou plusieurs attributs dans certains cas). Pour cela, on définit une clé.

Définition d'une clé :

Une **clé** est un groupe d'attribut minimum qui permet d'identifier une ligne d'une table **de manière unique**.

Exemples de clés :

Dans la table musique, la durée de la chanson n'est pas une clé. En effet, deux chansons différentes peuvent avoir la même durée. Par contre, l'identifiant est une clé : chaque chanson différente possède un identifiant unique.
En clair, avec une clé on peut définir de manière unique un objet d'une table.

Exemple

Pour illustrer le concept de clé on considère la table suivante :

chanson

id	nom_chanson	artiste	album	genre	duree	annee
1	Beautiful	Eminem	Refill	RAP US	392	2009
2	Brothers in Arms	Dire Straits	Brothers in Arms	ROCK	424	1985
3	The Real Slim Shady	Eminem	The Marshall Matters	RAP US	284	2001
4	Wonderwall	Oasis	Stop The Clocks	ROCK	258	2021
5	Wondervall	Oasis	Morning Glory ?	ROCK	258	2018

question : L'attribut identifiant (id) est il une clé (i.e : la connaissance de l'identifiant permet-il de définir une ligne concernée de manière unique) ?
réponse : Oui, à chaque valeur de l'identifiant correspond une seule et unique ligne.
question : L'attribut nom_chanson est il une clé ?
réponse : Non, on peut voir dans les deux dernières lignes deux versions de la chanson wonderwall : une première dans l'album Stop The Clocks, une seconde dans l'album : Morning Glory. La connaissance du nom de la chanson renvoie ici à deux possibilités.
question : Les deux attributs : nom_chanson et album forment-ils une clé ?
réponse : Oui, on suppose alors qu'il n'est pas possible de stocker dans la table deux versions de la même chanson qui figurent sur le même album.
Remarque : On pourrait imaginer un cas de figure où il y aurait deux versions de la même chanson sur le même album, les deux attributs nom_chanson et album ne formeraient plus alors une clé pour la table.
question : Est ce que les attributs nom_chanson, album et durée forment une clé ?
réponse : Non, on a vu que les attributs nom_chanson et album suffisent déjà à définir une et une seule ligne, or dans une clé le nombre d'attribut doit être minimal, dans ce cas 2.
Conclusion : On gardera à l'idée qu'une clé peut être formée d'un ou plusieurs attributs (en nombre minimal) et que chaque valeur de la clé renvoie à une seule et unique ligne de la table. On parle alors de relation un à un 1-1.

Notion de clé primaire

Parmi l'ensemble des clés possibles (dites clés candidates) on en choisit une qui sera la clé primaire. Parmi les groupes d'attributs qui sont des clés, on a l'habitude de choisir la clé primaire comme :

— ayant le moins d'attributs possible ;

— dont les attributs sont "simples" : des entiers, des chaînes de caractères...

La clé primaire est composée d'un ou plusieurs attributs qui identifient un objet de manière unique dans la table. La clé primaire est très importante, d'ailleurs pour certains logiciels de gestion de base de données, on doit spécifier quelle est la clé primaire dès la création de la table. Dans la plupart des cas, la clé primaire sera la colonne id.

Stocker les données dans une seule ou plusieurs tables ?

Dans une unique table :

Une première idée consiste à stocker dans une unique table toutes les données : ce n'est pas souhaitable. Le principal défaut de cela serait qu'il y aurait beaucoup d'informations redondantes. La redondance implique un coût de stockage élevé et une difficulté supplémentaire pour effectuer des modifications. En effet, si une même donnée apparaît à plusieurs endroits dans la table et qu'on souhaite la modifier, alors il faut effectuer des modification des données à plusieurs endroits : il y a donc plus de risques de commettre une erreur. Pour la modification d'une table : il est donc plus simple de faire une modification à un seul endroit. Pour cela il faut éviter au maximum la redondance.

Dans plusieurs tables :

En pratique, une base de données se compose de plusieurs tables. Entre ces tables on crée des liens. De cette manière, on essaie de limiter au maximum la redondance : on réalise ainsi des économies en termes de **stockage**, de plus il est plus facile de **faire des modifications**. Gardons à l'esprit qu'une base de donnée est un objet qui va évoluer au cours du temps : pensez au fichier client d'une société. En pratique, on aura donc forcément à réaliser des modifications sur une base de données au cours de son utilisation.

Conclusion : il est indispensable de découper l'ensemble des données en plusieurs tables reliées entre elles. Pour créer du lien entre les différentes tables, on introduit les clés étrangères.

Notion de clé étrangère

Définition d'une clé étrangère

Une clé étrangère est un attribut (ou plusieurs) d'une table qui sert de clé primaire dans une autre table. La clé étrangère permet de lier les deux tables. On dit que la clé étrangère de la première table **référence** la clé primaire de la seconde table. Autrement dit dans l'exemple ci-dessous la clé étrangère id_genre de la table chanson référence la clé primaire id de la table genre musical.

Exemple :

On crée une table genre_musical. L'attribut id_genre n'est pas une clé primaire de la table chanson, mais il renvoie à la clé primaire de la table genre_musical.

Remarque : ici on a supposé que pour une chanson donnée ne correspondait qu'un seul genre musical. C'est la limite de cet exemple qui ne sert qu'à illustrer la notion de clé étrangère. Gardons en tête que l'utilité des clés étrangère est de limiter la redondance en séparant en plusieurs tables les données et en reliant les tables entre elles.

chanson

id	nom_chanson	artiste	id_genre (clé étrangère)
1	Brothers in Arms	Dire Straits	2
2	Copines	Aya Nakamura	3
3	The Real Slim Shady	Eminem	1
4	Marché Noir	SCH	3
5	Beautiful	Eminem	1
6	Free	Steevie Wonder	4
7	Golden Lady	Steevie Wonder	4
8	Sous le ciel de Paris	Yves Montand	5
9	L'orage	Georges Brassens	5
10	La vie en rose	Edith Piaff	5

genre_musical

id (clé primaire)	nom_genre	annee_apparition
1	RAP US	1970
2	ROCK	1950
3	RAP FR	1985
4	Funk	1965
5	Chanson française	1000

Remarque : Lorsqu'on souhaite visualiser à la fois :

— la chanson,

— les informations sur le genre musical qui s'y rapportent

alors il faut "coller" les deux tables en faisant correspondre la colonne id_genre de la table chanson et id de la table genre musical : c'est ce qu'on appelle une **jointure**. Les jointures sont très utiles en base de données, on étudie les jointures dans les sections suivantes.

III Les requêtes SQL

Le langage SQL (Structured Query Language) est un langage qui permet d'interroger une base de données. Pour extraire les données souhaitées depuis la base de donnée on écrira des instructions en SQL (on parle de requête). Concrètement, cela permet de répondre à des questions comme "Quels sont les utilisateurs qui ont écouté cette chanson ?" ou bien "Quelle est la chanson la plus écoutée en ce moment ?" etc.

Dans cette section les commandes principales pour interroger une base de donnée sont présentées. Il s'agit de :

— sélectionner des colonnes (projection) d'une table. mot clé : `SELECT ... FROM`

— sélectionner des lignes (restriction) d'une table. mot clé : `WHERE`

— construire le produit cartésien de deux tables. mot clé : `FROM`

III.1 Sélectionner des colonnes avec SELECT

L'opération de projection

Lorsqu'on cherche à ne **conserver qu'une ou plusieurs colonnes d'une table** en ignorant les autres, (on peut aussi sélectionner toutes les colonnes d'une table) on réalise une opération dite de **projection**.
Pour sélectionner les colonnes qui nous intéressent par exemple le nom et le prénom d'un utilisateur, on écrira le nom des colonnes séparés par des virgules : `SELECT nom, prenom`. Le résultat de l'opération de projection sera une table avec un nombre de colonnes inférieur (ou égal) au nombre de colonnes de la table initiale et le même nombre de lignes.
Rappel : on parle d'attribut pour une colonne. Par exemple dans la table chanson ci-contre, chaque chanson possède un attribut : id, un attribut : titre, un attribut : artiste.

Syntaxe SQL :

```
SELECT colonne_1, colonne_2 FROM table_1
;
```

Renvoie la table formée en ne retenant, de la table nommée table_1, que les 2 colonnes nommées : colonne_1 et colonne_2.
Remarque : En SQL, chaque instruction se termine par un point virgule.

Sélectionner toutes les colonnes : l'opérateur *

Pour sélectionner toutes les colonnes on indique une étoile `*` c'est à dire qu'on écrira :
`SELECT * FROM table_1 ;`

Exemple

On souhaite conserver toutes les colonnes de la table utilisateur ci-dessous :

utilisateur

id	nom	prenom
1	Petit	Marie
2	Roy	Farid
3	Dubois	Elvire

Commande SQL :`SELECT * FROM utilisateur ;`
Le SGBD renvoie la table :

```
+----+--------+--------+
| id | nom    | prenom |
+----+--------+--------+
|  1 | Petit  | Marie  |
|  2 | Roy    | Farid  |
|  3 | Dubois | Elvire |
+----+--------+--------+
```

Pour sélectionner uniquement le titre des chansons de la table nommée `chanson` :

chanson

id	titre	artiste
1	Beautiful	Eminem
2	Brothers in Arms	Dire Straits
3	The Real Slim Shady	Eminem
4	Marché Noir	SCH
5	Copines	Aya Nakamura
6	Golden Lady	Steevie Wonder

Commande SQL :`SELECT titre FROM chanson ;`
Le SGBD renvoie la table :

```
+--------------------+
| nom_chanson        |
+--------------------+
| Beautiful          |
| Brother in Arms    |
| The Real Slim Shady |
| Marché noir        |
| Copines            |
| Golden Lady        |
+--------------------+
```

Préciser la table

On peut préciser la table avec la notation : `table.attribut`, par exemple, pour effectuer la même action que dans l'exemple précédent.
Commande SQL : `SELECT utilisateur.nom, utilisateur.prenom FROM utilisateur;`
Remarque : Cette notation deviendra indispensable lorsqu'on manipulera plusieurs tables, par exemple dans les opérations de jointure.

Renommer une table ou un attribut : les alias

Il est très souvent commode de renommer une table ou la colonne d'une table avec un nom plus simple et plus court afin de pouvoir la manipuler plus facilement. On dit alors qu'on lui donne a un alias.

pour une table : mot clé AS

Pour donner un alias au nom de la table on écrit :
`SELECT nouveau_nom.nom, nouveau_nom.prenom FROM utilisateur AS nouveau_nom;`
Remarque : le mot clé `AS` est facultatif. On peut aussi écrire :
`SELECT nouveau_nom.nom, nouveau_nom.prenom FROM utilisateur nouveau_nom;`

pour une colonne : mot clé AS

Pour donner un alias au nom de la colonne on écrit :
`SELECT ancien_nom AS nouveau_nom FROM table_1;`
On considère la table ci-dessous :

chanson

id	nom_chanson	artiste
1	Beautiful	Eminem
2	The Real Slim Shady	Eminem

On souhaite renommer la colonne nom_chanson en titre, on écrit :
`SELECT nom_chanson AS titre FROM chanson;`

```
+---------------------+
| titre               |
+---------------------+
| Beautiful           |
| The Real Slim Shady |
+---------------------+
```

Écrire une expression booléenne en SQL

Les opérateurs booléens

On présente les opérateurs de comparaison.

opérateur	résultat
`A = B`	est vrai si A est égal à B, faux sinon.
`A <>B`	est vrai si A est différent de B et faux si A est égal à B.
`A >= B`	est vrai si A est supérieur ou égal à B, faux sinon.
`A <= B`	est vrai si A est inférieur ou égal à B, faux sinon.
`A > B`	est vrai si A est strictement supérieur à B, faux sinon.
`A >= B`	est vrai si A est supérieur ou égal à B, faux sinon.

Les opérateurs logiques

Pour combiner des expressions booléennes on peut utiliser les opérations classiques :

— OU logique, mot clé : `OR`

— ET logique, mot clé : `AND`

— NON, mot clé : `NOT`

III.2 Sélectionner des lignes avec WHERE

L'opération de restriction

L'opération de restriction consiste à **sélectionner certaines lignes de la table** qui vérifient une condition à spécifier. Cette condition consiste en une expression booléenne qui sera évaluée pour chaque ligne. Cette condition est introduite après le mot clé `WHERE`.
Le résultat de l'opération de restriction est une table qui possède les mêmes colonnes que la table de départ mais un nombre de ligne inférieur (ou égal) au nombre de ligne de la table initiale.
Dans une opération de restriction, on filtre les lignes de la table avec une condition : si la condition est remplie on conserve la ligne, sinon on l'ignore.

Syntaxe SQL :

On indique la condition de restriction après le mot clé `WHERE` :

```
SELECT * FROM table_1 WHERE condition ;
```

Pour sélectionner les lignes de la table_1 pour lesquelles l'attribut identifiant est supérieur ou égal à 3 on écrira :

```
SELECT * FROM table_1 WHERE id >= 3 ;
```

Pour sélectionner les lignes de la table_1 pour lesquelles l'attribut artiste prend la valeur Eminem on écrira :

```
SELECT * FROM table_1 WHERE artiste =
'Eminem';
```

Exemple

On considère la table chanson ci-dessous dont on souhaite sélectionner le nom et l'artiste de chaque chanson dont la durée est supérieure à 180s.

chanson

id	titre	artiste	duree
1	Beautiful	Eminem	392
2	Brothers in Arms	Dire Straits	424
3	The Real Slim Shady	Eminem	284
4	Marché Noir	SCH	210
5	Copines	Aya Nakamura	171
6	Free	Steevie Wonder	251
7	She loves you	Les beattles	142
8	Sous le ciel de Paris	Yves Montand	173
9	L'orage	Georges Brassens	198
10	Help	Les Beattles	138

Commande SQL :

```
SELECT titre, artiste FROM chanson
WHERE duree>180;
```

Le SGBD renvoie la table :

```
+------------------+----------------+
|titre             |artiste         |
+------------------+----------------+
|Beautiful         |Eminem          |
|Brothers in Arms  |Dire Straits    |
|The Real Slim Shady|Eminem         |
|Marché Noir       |SCH             |
|Free              |Steevie Wonder  |
|L'Orage           |Georges Brassens|
+------------------+----------------+
```

III.3 Faire une recherche avec une chaîne de caractères : mot clé LIKE

Si on écrit une condition qui porte sur une chaînes de caractères, on peut écrire après le mot clé `WHERE` une égalité comme : `WHERE nom= 'Eminem'`. Cependant, si l'on veut par exemple sélectionner tous les artistes dont le nom commence par la lettre 'e', on n'a pas de moyen simple de le faire. Pour résoudre ce problème, on utilise le mot clé `LIKE` qui permet d'écrire des conditions sur des chaînes de caractères incomplètes en utilisant les commandes indiquées dans le tableau ci-dessous :

caractère	fonction	condition	valeur
_	remplace **exactement 1** caractère	`"Vive le SQL" LIKE "V_ve l_ SQL"`	`TRUE`
		`"Vive le SQL" LIKE "V_ive le SQL"`	`FALSE`
%	remplace **0, 1, plusieurs** caractères	`"Vive le SQL" LIKE "V%L"`	`TRUE`
		`"Vive le SQL" LIKE "V%ive le SQL"`	`TRUE`

Pour sélectionner les artistes dont le nom commence par la lettre e, la commande SQL est : `SELECT artiste WHERE nom LIKE 'e%'`

III.4 Le produit cartésien avec le mot clé FROM

Principe

Le produit cartésien de deux tables consiste à construire une table formée de toutes les combinaisons d'une ligne de la première table avec une ligne de la seconde table. Le produit cartésien de deux tables contenant n lignes pour la première, et p lignes pour la seconde, renvoie donc une table de $n \times p$ lignes.

♣			=	♣	V
♢	×	V		♣	As
♡		As		♢	V
♠				♢	As
				♡	V
				♡	As
				♠	V
				♠	As

Le produit cartésien est peu utilisé tel quel, par contre c'est une des deux opérations de base de la **jointure** dans laquelle on cherche à recoller deux tables (voir sections suivantes). On commence cette opération par un produit cartésien puis on opére une restriction.
On peut construire de la même manière le produit cartésien de 3 tables. Il sera obtenu comme l'ensemble des combinaisons d'une ligne de la première table, une ligne de la seconde, et d'une ligne de la troisième etc.

Syntaxe

Pour construire le produit cartésien de deux tables on utilise le mot clé `FROM` que l'on fait suivre du nom des deux tables séparé par une virgule.

```
SELECT * FROM table_1, table_2 ;
```

Exemple

On considère une table d'utilisateurs et une table de chansons. On veut construire la table contenant toutes les associations possibles entre un utilisateur et une chanson.

utilisateur

id_u	prenom	nom
1	Thomas	Goya
2	Héléna	Dupont
3	Charline	Picard

chanson

id_c	titre	groupe
1	Help	The beattles
2	Hurt	Johny Cash

Commande SQL :

```
SELECT * FROM utilisateur, chanson ;
```

Le SGBD renvoie la table :

```
+----+--------+------+----+-----+------------+
|id\_u|prenom  |nom   |id\_c|titre|groupe      |
+----+--------+------+----+-----+------------+
|   1|Thomas  |Goya  |   2|Hurt |Johny Cash  |
|   1|Thomas  |Goya  |   1|Help |The Beattles|
|   2|Héléna  |Dupont|   2|Hurt |Johny Cash  |
|   2|Héléna  |Dupont|   1|Help |The Beattles|
|   3|Charline|Picard|   2|Hurt |Johny Cash  |
|   3|Charline|Picard|   1|Help |The Beattles|
+----+--------+------+----+-----+------------+
```

formée de toutes les associations possibles de toutes les lignes de la table utilisateur avec chaque ligne de la table chanson.

IV Trier les données

IV.1 Ordonner ses données : le mot clé ORDER BY

Pour trier par ordre croissant des données on utilise le mot clé `ORDER BY` suivi de l'attribut selon lequel on classe les lignes.
On peut classer par ordre **croissant** : mot clé `ASC` (**ascending** en anglais) ou par ordre **décroissant** : mot clé `DESC`. Par défaut, l'ordre est **implicitement croissant** s'il n'est pas précisé.

chanson

id	titre	artiste	annee
1	Beautiful	Eminem	2009
2	Brothers in Arms	Dire Straits	1985
3	The Real Slim Shady	Eminem	2001
4	Still D.R.E	Dr. Dre	1999
5	My Name Is	Eminem	1999

Exemple

On souhaite classer les titres d'Eminem du plus récent au plus ancien.
Commande SQL :

```
SELECT titre, annee FROM chanson WHERE artiste='Eminem' ORDER BY annee DESC ;
```

Le SGBD renvoie :

```
+---------------------+-------+
| titre               | annee |
+---------------------+-------+
| Beautiful           |  2009 |
| The Real Slim Shady |  2001 |
| My Name Is          |  1999 |
+---------------------+-------+
```

ORDER BY fonctionne aussi avec les chaînes de caractères

Les chaînes de caractère ont aussi un ordre, l'ordre alphabétique. La commande `ORDER BY` peut ainsi ranger par ordre alphabétique avec `ASC` ou dans l'ordre alphabétique inversé en ajoutant le mot clé `DESC` (**descending** en anglais).

Exemple

On souhaite classer les artistes par ordre alphabétique (ou lexicographique).
Commande SQL :

```
SELECT artiste FROM chanson ORDER BY artiste ASC ;
```

Le SGBD renvoie :

```
+--------------+
| artiste      |
+--------------+
| Dire Straits |
| Dr. Dre      |
| Eminem       |
| Eminem       |
| Eminem       |
+--------------+
```

Supprimer les doublons avec DISTINCT

On reprend l'exemple précédent, on ajoute le mot clé `DISTINCT` après le mot clé `SELECT` pour éliminer les doublons.
Commande SQL :

```
SELECT DISTINCT artiste FROM chanson ORDER BY artiste ASC ;
```

Le SGBD renvoie :

```
+--------------+
| artiste      |
+--------------+
| Dire Straits |
| Dr. Dre      |
| Eminem       |
+--------------+
```

Sélectionner les premières lignes : mot clé LIMIT et/ou suivi du mot clé OFFSET

Pour sélectionner les 5 premières lignes de la table, on peut ajouter la commande `LIMIT 5` à la fin de la requête. Si on souhaite débuter à un indice qui n'est pas le premier, par exemple 10, on indiquera `OFFSET 9` (pour commencer au début on "écrirait" `OFFSET 0`).
Remarque : La commande `OFFSET` doit être indiquée **après** la commande `LIMIT`.
On considère la table chanson du paragraphe on souhaite sélectionner **4** artistes en commençant à partir du **deuxième** artiste. On écrira :

```
SELECT artiste FROM chanson LIMIT 4 OFFSET 1;
```

V Opérations sur les tables : les opérations ensemblistes

Union, intersection et différence de deux tables :

Pour réaliser des opérations ensemblistes, il est nécessaire que les deux tables possèdent le même schéma : c'est à dire qu'**elles aient les mêmes colonnes**. Si jamais les colonnes ne sont pas identiques dans les deux tables, alors on devra sélectionner uniquement des colonnes qui se retrouvent dans les deux tables.
Entre des tables qui ont les mêmes colonnes on peut réaliser des opérations :

- d'union : `UNION`
- d'intersection : `INTERSECT`
- de différence : `EXCEPT`

L'opération d'UNION de deux tables renvoie une table qui contient l'ensemble des lignes des deux tables.
L'opération d'INTERSECTION de deux tables renvoie une table qui contient les lignes présentes dans les deux tables à la fois.
L'opération de différence d'une table A par la table B renvoie la table formée des lignes de A qui ne sont pas présentes dans la table B.

Remarque : langage SQL $\neq$ SGBD

Attention, les logiciels de base de données (les SGBD) n'implémentent pas exactement toutes les fonctions du langage SQL. Par exemple dans le logiciel MySQL les opérateurs `INTERSECT` et l'opérateur `EXCEPT` du langage SQL ne sont pas implémentés. Cependant ces opérateurs sont implémentés dans le logiciel SQLite Studio par exemple.

Pour illustrer les opérations ensemblistes on considère les deux tables ci-dessous :

playlist_deezer

titre	artiste
Beautiful	Eminem
Creep	Radiohead
One	U2
Je dis aime	M

playlist_spotify

titre	artiste
Beautiful	Eminem
Stan	Eminem
Marché Noir	SCH
La boulette	Diams
Je dis aime	M

On souhaite faire l'union de deux playlist qui sont stockées dans deux tables de même schéma nomées : playlist_deezer et playlist_spotify.
Commande SQL :

```
SELECT * FROM playlist_deezer UNION SELECT * FROM playlist_spotify ;
```

On souhaite connaître les chansons qui se trouvent dans les deux playlist.
Commande SQL :

```
SELECT * FROM playlist_deezer INTERSECT SELECT * FROM playlist_spotify ;
```

Pour éviter les doublons on aimerait générer la liste obtenue en retirant à la playlist_deezer les lignes qui se trouvent dans playlist_spotify :

Commande SQL :

```
SELECT * FROM playlist_deezer EXCEPT SELECT * FROM playlist_spotify ;
```

Le SGBD renvoie :

UNION

```
+-----------+---------+
|titre      |artiste  |
+-----------+---------+
|Beautiful  |Eminem   |
|Creep      |Radiohead|
|One        |U2       |
|Je dis aime|M        |
|Stan       |Eminem   |
|Marché Noir|SCH      |
|La boulette|Diams    |
+-----------+---------+
```

INTERSECTION

```
+-----------+---------+
|titre      |artiste  |
+-----------+---------+
|Beautiful  |Eminem   |
|Je dis aime|M        |
+-----------+---------+
```

DIFFERENCE

```
+-----------+---------+
|titre      |artiste  |
+-----------+---------+
|Creep      |Radiohead|
|One        |U2       |
+-----------+---------+
```

VI Les jointures avec le mot clé JOIN

Objectif

Lorsque l'on souhaite réunir les informations contenues dans deux tables comme la table chanson et genre musical (voir chapitre II), on peut "coller" ces deux tables selon un critère. On appelle cette opération une **jointure**. En pratique, c'est très utile car on a expliqué qu'on avait découpé l'ensemble des données en plusieurs tables de façon à limiter la redondance. Lorsque l'on souhaite rassembler en une seule table les informations contenues dans 2 tables (ou plus), on réalise une jointure.

Syntaxe SQL

Pour faire une jointure en SQL, il y a 2 méthodes :

- écrire `table_1 JOIN table_2 ON condition` qui introduit la condition pour la jointure.
- faire un produit cartésien `FROM table_1, table_2`, puis une restriction avec `WHERE` où on ne garde parmi toutes les associations de lignes, que celles qui vérifient la condition de jointure.

```
SELECT * FROM table_1 JOIN table_2 ON condition ;
SELECT * FROM table_1, table_2 WHERE condition ;
```

En pratique on a souvent une condition qui prend la forme d'une colonne de table_1 qui renvoie vers une colonne de table_2 (cf clé étrangère). La requête SQL prend alors la forme :

```
SELECT * FROM table_1 JOIN table_2 ON table_1.champ_1 = table_2.champ_2 ;
SELECT * FROM table_1, table_2 WHERE table_1.champ_1 = table_2.champ_2 ;
```

Exemple :

On considère une table chanson et une seconde table genre musical données ci-dessous :

clé étrangère clé primaire

chanson

id	titre	groupe	id_genre
1	Beautiful	Eminem	1
2	Brothers in Arms	Dire Straits	2
3	Stan	Eminem	1
4	Marché Noir	SCH	3
5	Copines	Aya Nakamura	3

genre_musical

id	genre	annee
1	RAP US	1970
2	ROCK	1950
3	RAP FR	1985

On aimerait construire la table qui pour chaque chanson ajoute des informations sur le genre musical. On doit donc réaliser la jointure en faisant correspondre la clé primaire genre_musical.id avec la clé étrangère chanson.id_genre.
Commande SQL avec les mots clés : JOIN ON (version au programme) :

```
SELECT * FROM chanson JOIN genre_musical ON chanson.id_genre=genre_musical.id ;
```

Commande SQL avec un produit cartésien et une restriction (variante) :

```
SELECT * FROM chanson, genre_musical WHERE chanson.id_genre=genre_musical.id ;
```

Le SGBD renvoie alors :

```
+----+------------------+--------------+----------+----------+--------+-------+
| id | titre            | groupe       | id\_genre | id       | genre  | annee |
+----+------------------+--------------+----------+----------+--------+-------+
|  1 | Beautiful        | Eminem       |        1 |        1 | RAP US |  1970 |
|  2 | Brothers in Arms | Dire Straits |        2 |        2 | ROCK   |  1950 |
|  3 | Stan             | Eminem       |        1 |        1 | RAP US |  1970 |
|  4 | Marché Noir      | SCH          |        3 |        3 | RAP FR |  1985 |
|  5 | Copines          | Aya Nakamura |        3 |        3 | RAP FR |  1985 |
+----+------------------+--------------+----------+----------+--------+-------+
```

VI.1 L'autojointure

Dans certains cas on peut souhaiter joindre deux fois la même table. On considère la table chanson ci-contre. Pour l'exemple on souhaite écouter **deux** chansons à la suite mais on veut que la durée totale de l'écoute soit inférieure à 9 minutes (540 secondes).

On peut alors faire une jointure entre la table chanson et elle même avec la condition de jointure étant que la somme des durées des chansons respecte la condition.

chanson

id	titre	groupe	duree
1	Beautiful	Eminem	392
2	Brothers in Arms	Dire Straits	424
3	Stan	Eminem	399
4	Marché Noir	SCH	210
5	Copines	Aya Nakamura	171

Le SGBD renvoie la table :

```
+--------------+--------------+
| titre        | titre        |
+--------------+--------------+
| Copines      | Marché Noir  |
| Marché Noir  | Marché Noir  |
| Copines      | Copines      |
| Marché Noir  | Copines      |
+--------------+--------------+
```

Commande SQL :

```
SELECT a.titre, b.titre FROM chanson AS a, chanson AS b WHERE a.duree + b.duree<=540 ;
```

Effectuer des calculs

Il est possible d'envoyer une requête pour compter le nombre de lignes, le nombre de lignes distinctes qui vérifient un certain critère, on bien de faire des opérations sur les colonnes. On présente les principales commandes qui permettent de réaliser les opérations classiques.

Les opérateurs classiques

On peut réaliser les opérations suivantes :

Commande SQL	Opération
`COUNT(*)`	compte les lignes renseignées entre parenthèses
`AVG`	calcule la moyenne des valeurs sélectionnées
`SUM`	calcule la somme des valeurs sélectionnées
`MAX`	renvoie le maximum des valeurs sélectionnées
`MIN`	renvoie le minimum des valeurs sélectionnées.
`+,-,*,/`	opérations somme, différence, produit, division

Exemples :

On considère la table chanson ci-dessous :

chanson

id	titre	artiste	duree	annee
1	Beautiful	Eminem	392	2009
2	Brothers in Arms	Dire Straits	424	1985
3	The Real Slim Shady	Eminem	284	2001
4	Marché Noir	SCH	210	2021
5	Copines	Aya Nakamura	171	2018
6	Free	Steevie Wonder	251	1987
7	Golden Lady	Steevie Wonder	298	1973
8	Sous le ciel de Paris	Yves Montand	173	1964
9	L'orage	Georges Brassens	198	1960
10	La vie en rose	Edith Piaff	185	1947

On souhaite compter combien d'artistes différents il existe dans la table chanson on écrira :
```
SELECT COUNT(DISTINCT artiste) FROM
chanson;
```

Pour compter le nombre de chansons total on peut utiliser la commande `COUNT` :
```
SELECT COUNT(*) FROM chanson ;
```

Pour connaître la chanson dont la durée est maximale on peut utiliser la commande `MAX` :
```
SELECT nom,MAX(duree) FROM chanson ;
```

Pour compter combien de chanson d'Eminem font partie de la table chanson on ajoute une restriction `WHERE` :
```
SELECT COUNT(*) FROM chanson WHERE
artiste='Eminem';
```

Pour connaître le temps mis pour écouter 5 fois chaque chanson on peut former la colonne : `5*duree` :
```
SELECT nom,5*duree FROM chanson ;
```

Pour connaître la durée de toutes les chansons mises bout à bout on utilise la commande `SUM`.
```
SELECT SUM(duree) FROM chanson ;
```

Pour connaître la durée moyenne d'une chanson de Steevie Wonder on utilise la commande `AVG`.
```
SELECT AVG(duree) FROM chanson
WHERE nom='Steevie Wonder';
```

VII Faire des groupes : l'agrégation

Très souvent on a besoin de constituer des groupes avant d'effectuer des calculs sur ces groupes. Pour résoudre ce problème en SQL on utilise l'agrégation.

Principe

L'opération d'agrégation consiste en deux étapes :

— le partitionnement,

— la fonction d'agrégation.

Première étape : on constitue des groupes dans une table. Dans le jargon des bases de données on parle de **partitionnement** et les groupes s'appellent des **agrégats**.
Deuxième étape : on effectue des calculs sur chacun des groupes : on parle de **fonction d'agrégation**. La fonction d'agrégation s'applique à chaque groupe et renvoie une valeur.

Syntaxe :

Pour constituer des groupes on utilise le mot clé : `GROUP BY` suivi de l'attribut pour constituer les groupes appelé attribut de partitionnement.

Exemple

On considère la table ci-dessous. On pourrait se poser la question : pour chaque artiste, combien y a-t-il de chanson dans la table ? On aura alors un agrégat par artiste qui sera constitué de ses chansons, puis on devra compter le nombre de chansons avec la fonction `COUNT(*)`.

chanson

id	titre	artiste	genre	duree	annee
1	Beautiful	Eminem	RAP US	392	2009
2	Brothers in Arms	Dire Straits	ROCK	424	1985
3	The Real Slim Shady	Eminem	RAP US	284	2001
4	Marché Noir	SCH	RAP FR	210	2021
5	Copines	Aya Nakamura	RAP FR	171	2018
6	Free	Steevie Wonder	Funk	251	1987
7	Golden Lady	Steevie Wonder	Funk	298	1973
8	Sous le ciel de Paris	Yves Montand	Chanson française	173	1964
9	L'orage	Georges Brassens	Chanson française	198	1960
10	La vie en rose	Edith Piaff	Chanson Française	185	1947

Commande SQL : `SELECT artiste,COUNT(*) FROM chanson GROUP BY artiste ;`
De la même manière on peut se poser la question : quelle est la durée moyenne d'une chanson pour chaque artiste.
Commande SQL : `SELECT artiste, AVG(duree) FROM chanson GROUP BY artiste ;`
Combien y a-t-il de chanson dans chaque genre musical ?
Commande SQL : `SELECT genre, COUNT(*) FROM chanson GROUP BY genre ;`

À chaque fois qu'on veut répondre à ces questions, on se rend compte qu'avant d'effectuer les calculs, on doit partitionner la table en groupes. Pour écrire la requête on doit alors penser à l'agrégation.

artiste	COUNT(*)	AVG(duree)
Eminem	2	338.0000
Dire Straits	1	424.0000
SCH	1	210.0000
Aya Nakamura	1	171.0000
Steevie Wonder	2	274.5000
Yves Montand	1	173.0000
Georges Brassens	1	198.0000
Edith Piaf	1	185.0000

genre	COUNT(*)
RAP US	2
ROCK	1
RAP FR	2
Funk	2
Chanson francaise	3

Règle pour l'agrégation

Si on souhaite appeler une fonction d'agrégation après le `SELECT`, on ne pourra appeler que des fonctions d'agrégation (qui prennent en argument un agrégat et renvoie un résultat), et/ou l'attribut de partitionnement. On ne peut pas afficher les autres attributs (qui ne sont pas des attributs de partitionnement). On **ne peut pas** écrire par exemple : `SELECT artiste, titre, COUNT(*) FROM chanson GROUP BY artiste ;` car titre n'est pas un attribut de partitionnement.

Ajouter des critères de restriction après l'agrégation : le mot clé HAVING

Pour rappel les critères de restrictions introduits avec le mot clé `WHERE` opèrent des restrictions : on ne considère plus que certaines lignes **avant d'avoir agrégé les données**.
On peut souhaiter faire des restrictions **après l'agrégation des données**, pour cela on ajoute le mot clé `HAVING`.

Exemple

On souhaite sélectionner les artistes et le nombre de leurs chansons dans la table, qui appartiennent au genre RAP US, et parmi ces artistes ne retenir que ceux qui ont au moins 2 chansons dans la table. Alors on comprend bien qu'il faut d'abord éliminer tous les artistes dont le genre n'est pas du RAP US (mot clé `WHERE`) puis constituer un groupe par artiste (mot clé `GROUP BY`) puis enfin retenir ceux qui ont plus de deux chansons dans la base (mot clé `HAVING`).
Commande SQL :

```
SELECT artiste,count(*) FROM chanson WHERE genre = 'RAP US' GROUP BY artiste HAVING COUNT(*)>=2 ;
```

Remarque : on vérifie la règle pour l'agrégation à savoir que : si on fait une agrégation alors on trouve après l'instruction `SELECT` :

— des fonctions d'agrégation,

— des attributs qui sont dans le "`GROUP BY`" (ici l'attribut artiste).

VIII Cardinalité : les relations 1-1, 1-*,*-*

Relation 1 à 1

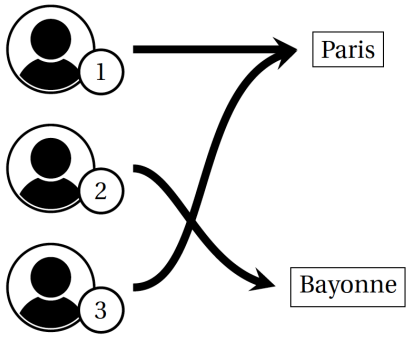
Pour illustrer la relation 1 à 1 on pense à la clé d'une table qui fait correspondre à une valeur de la clé, une unique ligne. De manière réciproque, une ligne donnée renvoie à une seule valeur de la clé. On a bien une relation **1** à **1**.

Relation 1 à plusieurs

`Pour illustrer la relation 1 à plusieurs, on supposera qu'un artiste ne peut avoir qu'un seul genre musical`

Un artiste appartient à **un seul** genre musical, mais un genre musical peut être associé à **plusieurs** artistes. C'est donc une relation un à plusieurs.

De même à chaque utilisateur on associe **une seule** ville mais pour une ville donnée **plusieurs** utilisateurs peuvent être associés. C'est encore une relation un à plusieurs.



utilisateur

id	nom	prénom	age	ville
1	Durand	Tom	19	Paris
2	Legrand	Charles	46	Bayonne
3	Riolon	Marie	35	Paris

chanson

id	titre	artiste	genre
1	Beautiful	Eminem	RAP US
2	Brothers in Arms	Dire Straits	ROCK
3	The Real Slim Shady	Eminem	RAP US

L'utilisation d'une clé étrangère permet de représenter une relation 1 à plusieurs entre deux tables. On souhaite réunir les informations réunies dans la table artiste et dans la table genre_musical présentées ci-dessous. Pour cela on réalise une opération de jointure en écrivant :

Commande SQL :

```
SELECT * FROM chanson JOIN genre_musical ON chanson.id_genre = genre_musical.id ;
```

La clé étrangère de la table chanson fait correspondre à **un** genre musical **plusieurs** artistes. C'est donc une relation un à plusieurs.

chanson (clé étrangère : id_genre)

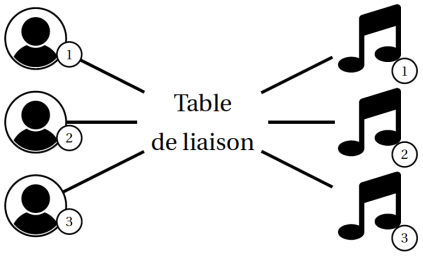
id	nom_chanson	artiste	id_genre
1	Brothers in Arms	Dire Straits	2
2	Copines	Aya Nakamura	3
3	Golden Lady	Steevie Wonder	4
4	Marché Noir	SCH	3
5	Free	Steevie Wonder	4

genre_musical (clé primaire : id)

id	nom_genre	annee_apparition
1	RAP US	1970
2	ROCK	1950
3	RAP FR	1985
4	Funk	1965
5	Chanson française	1000

Relation plusieurs à plusieurs : *-*

On souhaite créer un historique des morceaux qui ont été écoutés par chaque utilisateur, pour savoir par exemple : qui écoute le plus de musique ou bien quelle chanson est la plus écoutée. Dans ce cas :
Une même chanson peut avoir été écoutée par **plusieurs** utilisateurs.
Un même utilisateur peut avoir écouté **plusieurs** chansons.
C'est donc une relation **plusieurs** à **plusieurs**.
Dans les exemples précédents on a vu qu'une clé étrangère d'une table A permet de renvoyer à une clé primaire ("unique") de la table B. La clé étrangère permet ainsi de représenter une association un à plusieurs. Pour représenter une relation plusieurs à plusieurs on ne peut pas utiliser une seule clé étrangère.



Pour représenter cette relation plusieurs à plusieurs on introduit une table intermédiaire qu'on appelle de façon générale une **table d'association** ou **table de liaison** qui va stocker, une à une, l'ensemble des chansons écoutées par chaque utilisateur. Pour notre exemple on appelle cette table : écoute car chaque ligne correspondra à un utilisateur qui écoute une chanson. En fait cette table est constituée de **deux clés étrangères** : une qui va référencer la table utilisateur et l'autre qui va référencer la table chanson.
En définitive on a modélisé une relation plusieurs à plusieurs en utilisant 2 relations 1 à plusieurs.

utilisateur

id	nom	prénom
1	Durand	Tom
2	Legrand	Charles
3	Riolon	Marie

écoute

id_utilisateur	id_chanson
1	3
2	3
1	1
2	3
2	1

chanson

id	titre	artiste
1	Beautiful	Eminem
2	Brothers in Arms	Dire Straits
3	The Real Slim Shady	Eminem

Rappel : jointures et table d'association

Si on souhaite effectuer une requête SQL pour visualiser ces trois tables on va devoir faire une jointure double.

Avec la syntaxe avec JOIN ON :

```
SELECT * FROM utilisateur JOIN écoute ON utilisateur.id=écoute.id_utilisateur JOIN chanson ON écoute.id_chanson = chanson.id ;
```

Ou bien en utilisant le produit cartésien puis la restriction :

```
SELECT * FROM utilisateur, écoute, chanson WHERE utilisateur.id=écoute.id_utilisateur AND écoute.id_chanson = chanson.id ;
```

Bibliographie

[1] Initiez-vous à l'algèbre relationnelle avec le SQL. Nicolas Rangeon, Openclassroom

[2] Implémentez vos bases de données relationnnelles avec SQL. Quentin Durantay, Openclassroom

[3] Ressources wikipedia sur les thèmes abordés (musique et base de données)