TP : Requêtes avec jointures

Instruction générale : hormis pour les exercices corrigés collectivement, vous ferez valider votre travail par l'enseignant.

1 Requêtes avec jointure

Dans les TP précédents, nous avons effectué des requêtes sur une seule table à l'aide de la syntaxe suivante :

```
SELECT attributs
FROM table
WHERE conditions
+ options (tri, nombre d'éléments, etc)
```

Toutefois, nous avons pas utilisé les différents liens entre les tables de la base de données. Ces liens nous permettent de donner du sens à nos données. En effet, autant l'usage de clés étrangères évite des redondances, allège les tables (un identifiant sous forme numérique est souvent plus léger que du texte) et permet de contrôler l'intégrité des données entre les tables ; autant les clés ne sont pas très parlantes comme on peut le voir dans l'exemple ci-dessous.

Exemple : Le tableau ci-dessous donne le résultat de la requête sur la table `morceaux` au sujet du premier morceau à l'aide du script suivant.

```
SELECT *
FROM morceaux
WHERE id_morceau=1 ;
```

id_morceau	nom_morceau	id_artiste	annee_morceau
1	Money	1	1973

Ce résultat n'est pas parlant en ce qui concerne l'artiste. Il serait pratique de faire le lien avec la table `artistes` afin de savoir qui est l'artiste d'identifiant 1 et d'afficher son nom.

Pour faire ces liens, nous utilisons des **requêtes avec jointure** grâce à la commande `JOIN` et la syntaxe suivante :

```
SELECT table_1.attribut_1, table_2.attribut_2
FROM table_1
INNER JOIN table_2
ON table_1.clé_étrangère_table_1 = table_2.clé_primaire_table_2
WHERE conditions
+ options (tri, nombre d'éléments, etc)
```

— `JOIN` est précédé du type de la jointure, ici `INNER` qui correspond à l'intersection entre deux ensembles en mathématiques ; nous verrons plus loin les autres types de jointures.
— `ON` énonce où se fait la jointure, autrement dit, quel attribut de la table une (clé étrangère) correspond à quel attribut de la table deux (clé primaire).

Exemple : le script suivant permet d'effectuer la même requête que précédemment mais en affichant cette fois-ci le nom de l'artiste à la place de son identifiant.

```
SELECT morceaux.id_morceau, morceaux.nom_morceau, artistes.nom_artiste,
morceaux.annee_morceau
FROM morceaux
INNER JOIN artistes
ON morceaux.id_artiste = artistes.id_artiste
WHERE id_morceau = 1 ;
```

On obtient alors

id_morceau	nom_morceau	id_artiste	annee_morceau
1	Money	Pink Floyd	1973

Remarque : comme on effectue une requête sur plusieurs tables, la syntaxe `table.attribut` permet de préciser dans quelle table on cherche l'attribut voulu.

Exercice 1.

1. Afficher pour toutes les attaques leur nom et leur type.
2. Même question mais seulement avec les attaques de la première génération.
3. Même question en gardant les quinze attaques les plus puissantes.

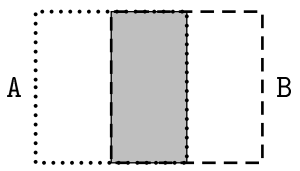
Exercice 2. La table `pokemons` a pour clé étrangère le nom des types au lieu de leurs identifiants. Corriger cela puis afficher les noms des pokémons avec leurs types.

2 Types de jointures

Il existe en tout sept types de jointures. Elles correspondent à des opérations mathématiques ensemblistes comme l'intersection, l'union et le complémentaire. Dans tous les schémas ci-dessous, la table `A` sera représentée en pointillés et la table `B` en tirets ; la jointure sera en gris délimitée par un trait plein.

2.1 INNER JOIN

On pourrait assimiler cela à l'intersection $A \cap B$. Cela renvoie tous les enregistrements (ou du moins les attributs sélectionnés) qui satisfont les conditions de jointures `ON` et éventuellement celle du `WHERE`.



Syntaxe :

```
SELECT *
FROM table_A
INNER JOIN table_B
ON table_A.clé = table_B.clé
```

Exemple : Ajoutons l'artiste Helloween à la table `artistes`.

```
INSERT INTO artistes (nom_artiste)
VALUES ("Helloween") ;
```

Affichons les artistes ayant des morceaux présents dans la table dédiées et les noms de ceux-ci.

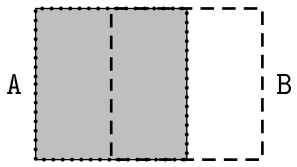
```
SELECT morceaux.nom_morceau, artistes.nom_artiste
FROM artistes
INNER JOIN morceaux
ON morceaux.id_artiste = artistes.id_artiste
```

On peut constater que le groupe Helloween n'ayant pas de morceau enregistré n'apparaît pas dans les résultats de la recherche. Par ailleurs, on peut constater que les rôles des tables `artistes` et `morceaux` ont été inversés par rapport à l'exemple précédent ; cela n'a pas d'importance car la jointure `INNER` est symétrique.

2.2 LEFT JOIN

2.2.1 Incluant B

On pourrait assimiler cela à l'ensemble A. Cela renvoie tous les enregistrements de la table `A` qu'ils aient une correspondance ou non avec la table `B`. Bien évidemment, s'il y a correspondance, on récupérera les données associées.



Syntaxe :

```
SELECT *
FROM table_A
LEFT JOIN table_B
ON table_A.clé = table_B.clé
```

Exemple : Cette fois-ci, l'artiste Helloween apparaît dans les résultats, même s'il n'a pas de morceaux associés.

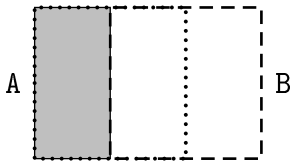
```
SELECT morceaux.nom_morceau, artistes.nom_artiste
FROM artistes
LEFT JOIN morceaux
ON artistes.id_artiste = morceaux.id_artiste
```

Cette opération n'est pas symétrique. On peut s'en convaincre en exécutant le script suivant où les rôles de `artistes` et `morceaux` sont échangés et Helloween n'apparaît plus dans les résultats.

```
SELECT morceaux.nom_morceau, artistes.nom_artiste
FROM morceaux
LEFT JOIN artistes
ON morceaux.id_artiste = artistes.id_artiste
```

2.2.2 Excluant `B`

On pourrait assimiler cela à l'ensemble $A \cap \overline{B}$. Cela renvoie tous les enregistrements de la table `A` qui n'ont pas correspondance avec la table `B`. D'une certaine façon, on pourrait le voir comme le contraire de `INNER` côté `A`.



Syntaxe :

```
SELECT *
FROM table_A
LEFT JOIN table_B
ON table_A.clé = table_B.clé
WHERE table_B.clé IS NULL
```

Exemple : Cette fois-ci, seul l'artiste Helloween apparaît dans les résultats car il n'a aucun morceau associé.

```
SELECT morceaux.nom_morceau, artistes.nom_artiste
FROM artistes
LEFT JOIN morceaux
ON artistes.id_artiste = morceaux.id_artiste
WHERE morceaux.id_artiste IS NULL
```

À nouveau, cette opération n'est pas symétrique. On peut s'en convaincre avec l'exemple suivant.

Exemple : Ajoutons le morceau « Age of Machine » à la table `morceaux`.

```
INSERT INTO morceaux(nom_morceau)
VALUES ("Age of Machine")
```

Puis effectuons la requête

```
SELECT morceaux.nom_morceau, artistes.nom_artiste
FROM morceaux
LEFT JOIN artistes
ON morceaux.id_artiste = artistes.id_artiste
WHERE artistes.id_artiste IS NULL
```

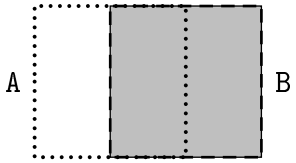
Cette fois-ci, c'est le morceau « Age of Machine » qui est obtenu en résultat car c'est le seul à ne pas avoir de valeur pour l'attribut `id_artiste`.

Exercice 3. Dans le script précédent, cela change-t-il quelque chose si on remplace `artistes.id_artiste` par `morceaux.id_artiste` dans le `WHERE` ? Pourquoi ?

2.3 RIGHT JOIN

2.3.1 Incluant A

On pourrait assimiler cela à l'ensemble B. Cela renvoie tous les enregistrements de la table `B` qu'ils aient une correspondance ou non avec la table `A`. Bien évidemment, s'il y a correspondance, on récupérera les données associées.



Syntaxe :

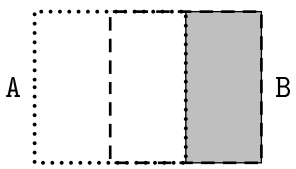
```
SELECT *
FROM table_A
RIGHT JOIN table_B
ON table_A.clé = table_B.clé
```

Remarque : `RIGHT JOIN` n'est pas disponible dans DB Browser.

Exercice 4. Comment faire une jointure droite sans la commande `RIGHT JOIN` ?

2.3.2 Excluant A

On pourrait assimiler cela à l'ensemble $\overline{A} \cap B$. Cela renvoie tous les enregistrements de la table `B` qui n'ont pas correspondance avec la table `A`. D'une certaine façon, on pourrait le voir comme le contraire de `INNER` côté `B`.



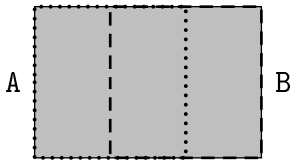
Syntaxe :

```
SELECT *
FROM table_A
RIGHT JOIN table_B
ON table_A.clé = table_B.clé
WHERE table_B.clé IS NULL
```

2.4 FULL JOIN

2.5 Avec intersection

On pourrait assimiler cela à l'ensemble $A \cup B$. Cela renvoie tous les enregistrements des tables `A` et `B`, qu'ils aient ou non des correspondances entre eux.



Syntaxe :

```
SELECT *
FROM table_A
FULL JOIN table_B
ON table_A.clé = table_B.clé
```

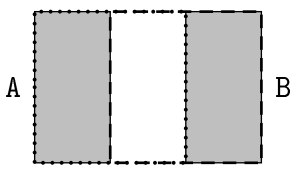
Remarque : `FULL JOIN` n'est pas disponible dans DB Browser.

Exemple : Le script suivant renvoie tous les enregistrement des deux tables, même si le lien entre les deux n'est pas fait.

```
SELECT morceaux.nom_morceau, artistes.nom_artiste
FROM artistes
FULL JOIN morceaux
ON artistes.id_artiste = morceaux.id_artiste
```

2.6 Sans intersection

On pourrait assimiler cela à l'ensemble $(A \cap \overline{B}) \cup (\overline{A} \cap B) = A \oplus B$. Cela renvoie seulement les enregistrements des tables `A` et `B` qui n'ont pas de correspondance avec l'autre table.



Syntaxe :

```
SELECT *
FROM table_A
FULL JOIN table_B
ON table_A.clé = table_B.clé
WHERE table_A.clé IS NULL OR table_B.clé IS NULL
```

Exemple : Le script suivant renvoie seulement les enregistrement correspondant à Helloween et Age of Machine car seuls ces deux là n'ont pas de correspondance avec l'autre table.

```
SELECT morceaux.nom_morceau, artistes.nom_artiste
FROM artistes
FULL JOIN morceaux
ON artistes.id_artiste = morceaux.id_artiste
WHERE artistes.id_artiste IS NULL OR morceaux.id_artiste IS NULL
```

3 Ressources supplémentaires

— Vidéo sur les requêtes avec jointures
— Cours du W3 Schools sur le SQL
— Cours du site sqh.sh
— Mémo du site sql.sh
— Plus de fichiers CSV sur les pokémons sont disponibles sur ce Github.