Formation SQL, les jointures

m2iformation.fr

Introduction

Les BDD relationnelles sont conçues pour **stocker des informations dans des tables distinctes**, chacune organisée autour d'un ensemble de colonnes.

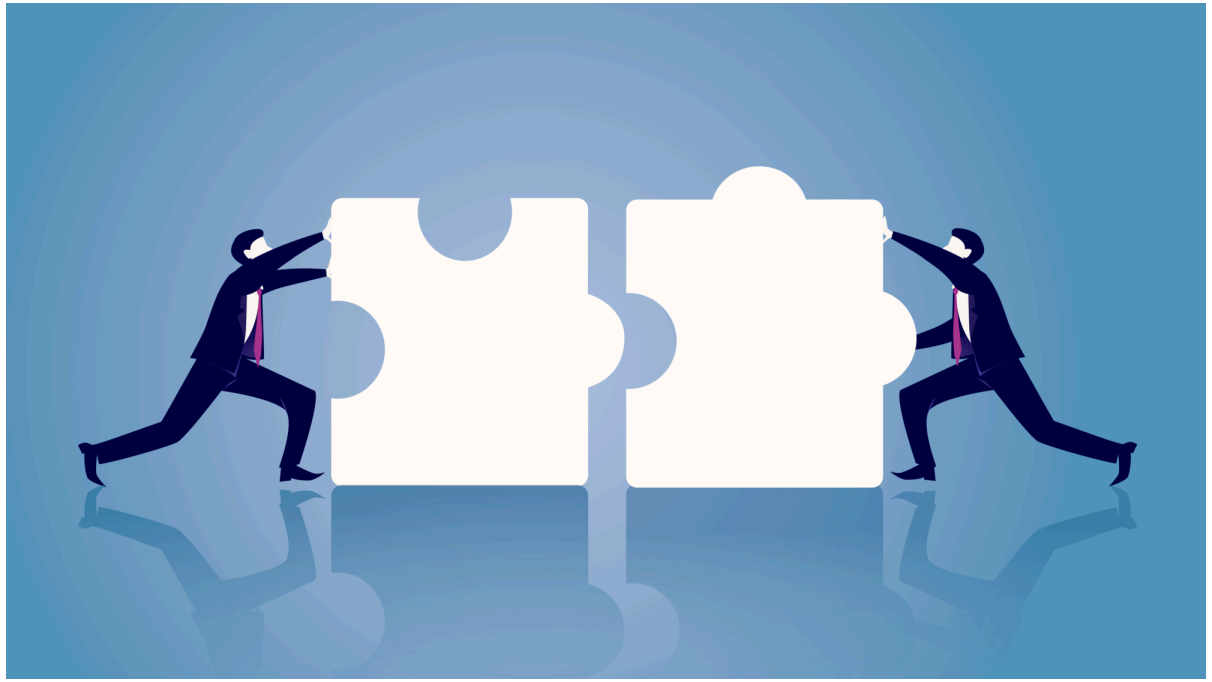
Cependant, ces données ne sont utiles que lorsqu'elles peuvent **être reliées et combinées pour fournir une vue complète des informations.**

C’est là qu’interviennent **les jointures.**

Introduction

Une jointure est une opération qui permet **de relier des tables entre elles en fonction de colonnes communes.**

En d'autres termes, elle permet d’associer des données dispersées sur plusieurs tables, en les combinant en une seule vue logique.



Exemple de jointure

History_Grades

student_id	student_name	history_grade
1	Alice	A
2	Jane	B
3	Julie	A

Math_Grades

student_id	student_name	math_grade
1	Alice	B
4	Roger	A
5	Kate	C

INNER JOIN

student_id	student_name	history_grade	student_id	student_name	math_grade
1	Alice	A	1	Alice	B

UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

Construire des relations entre les tables

Construire des relations

Les relations entre tables (ou entités) **permettent de structurer et d'organiser les données de manière logique et cohérente.**

Une bonne compréhension de ces relations aide à concevoir des bases de données efficaces. Dans sa forme simplifiée, on parle de:

- **Relation One-To-One**
- **Relation One-To-Many**
- **Relation Many-To-Many**

Connaître le type de relation entre deux tables permet de savoir comment utiliser la **foreign key (clé étrangère).**

L'importance de la clé étrangère

La clé étrangère joue un rôle clé dans la structuration des BDDR, et bien qu'elle **ne soit pas strictement nécessaire** pour effectuer des jointures en SQL, la clé étrangère **est cruciale pour assurer l'intégrité des données** dans la BDD:

- **Intégrité des données:** Elles empêchent l'insertion de valeurs incohérentes. Par exemple, on ne peut pas ajouter une commande pour un client qui n'existe pas.
- **Simplification des jointures:** Elles facilitent la création de jointures SQL, en rendant les associations entre tables explicites et robustes.

Relation One-To-One

Dans un modèle one-to-one, chaque enregistrement de l'une des tables correspond à un enregistrement unique dans l'autre. **Il suffit souvent de définir une foreign key dans l'une des deux tables.**

Exemple :

- **Table Employees:** contient un champ employee_id (PK).
- **Table Passports:** contient passport_id (PK) et employee_id (FK) pointant vers employee_id dans Employees.

Employee_id dans Passports est à la fois **une foreign key et une contrainte UNIQUE, garantissant l'association 1:1.**

Utilisé pour des informations complémentaires, mais indépendantes.

Relation One-To-Many

En One-to-Many, **une seule foreign key est généralement suffisante.** On place la foreign key du côté de la table "enfant" (la table qui peut avoir plusieurs enregistrements pour un enregistrement de la table "parent").
Exemple :

- **Table Authors:** contient un champ author_id comme clé primaire.
- **Table Books:** contient un champ author_id comme foreign key qui pointe vers author_id dans Authors.

Ainsi, chaque livre (Books) est associé à un seul auteur (Authors), mais un auteur peut avoir plusieurs livres.

Relation Many-To-Many

En Many-to-Many, chaque enregistrement peut être associé à plusieurs enregistrements d'une autre table, et vice versa. Pour modéliser cela, on utilise une **table de liaison** qui contient **deux FK.**

Exemple: Chaque Student peut être inscrit à plusieurs Courses, et chaque Course peut avoir plusieurs étudiants.

- **Table Students:** contient student_id comme clé primaire.
- **Table Courses:** contient course_id comme clé primaire.
- **Table de liaison StudentCourses:** contient student_id comme foreign key pointant vers Students, et course_id comme foreign key pointant vers Courses.

Les différentes catégories de jointures

Une fois que la (ou les) Foreign Key a été implémenté, nous pouvons **construire des jointures**.

Il existe principalement 5 catégories de jointures:

- Les jointures internes
- Les jointures externes
- Les jointures naturelles
- Les jointures croisées
- Les auto-jointures

Commençons par la plus populaire d'entre elles, **la jointure interne**

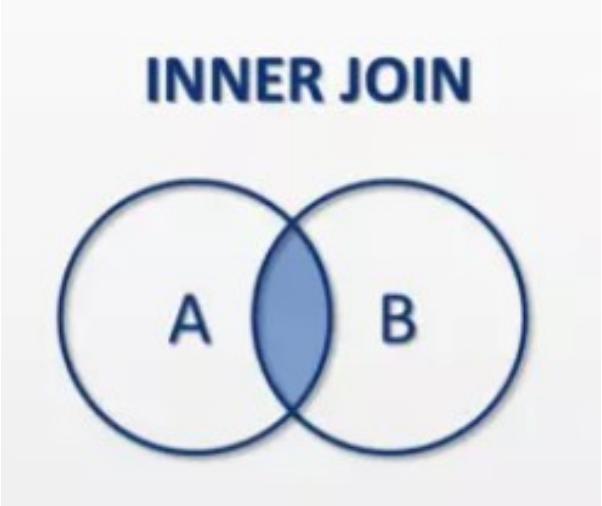
UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

Les jointures internes

Description

Une jointure interne (INNER JOIN) est utilisée pour renvoyer uniquement les lignes qui ont des correspondances entre deux tables. Si une ligne d'une table n'a pas de correspondance dans l'autre table, elle est ignorée dans le résultat final.

C'est le type de jointure le plus utilisé, car il est souvent nécessaire de combiner des informations de plusieurs tables en se basant sur une clé commune.



Ancienne syntaxe (dépréciée)

Dans l'ancienne syntaxe SQL, une jointure interne était réalisée en mentionnant toutes les tables dans la clause FROM, puis en définissant les conditions de jointure dans la clause WHERE. **Aussi appelée "syntaxe implicite"**, car les conditions de jointure et de filtrage étaient mélangées dans la même clause.

Utilisateurs et commandes

```
SELECT users.name, orders.order_date
FROM users, orders
WHERE users.id = orders.user_id;
```

Nouvelle syntaxe SQL3

Dans la nouvelle syntaxe, les jointures sont plus explicites. **Le mot-clé JOIN** est utilisé pour spécifier directement quelle relation entre les tables doit être utilisée pour faire correspondre les lignes.
Il permet de distinguer les conditions de jointure des conditions de filtrage, ce qui améliore la lisibilité.

Utilisateurs et commandes

```
SELECT users.name, orders.order_date
FROM users
JOIN orders ON users.id = orders.user_id;
```

Gagner en précision: INNER JOIN

Lorsqu'on utilise le mot `JOIN`, on réalise par défaut une **jointure interne (INNER)**. Mais d'autres jointures existent, elles aussi utilisant le mot-clé `JOIN`, donc pour ne pas confondre les différents types de jointure, on privilégiera `INNER JOIN`.

`JOIN` et `INNER JOIN` ont le même rôle, mais le second évite toute confusion.

A partir de maintenant, nous utiliserons la syntaxe **INNER JOIN.**

Exemple

Table `users` :

id	name
1	Alice
2	Bob
3	Carol

Table `orders` :

id	user_id	order_date
1	1	2024-01-05
2	2	2024-01-10
3	1	2024-01-20

Exemple

```sql
SELECT users.name, orders.order_date
FROM users
INNER JOIN orders ON users.id = orders.user_id;
```

name	order_date
Alice	2024-01-05
Bob	2024-01-10
Alice	2024-01-20

Explication du résultat :

- L'utilisateur **Carol** n'apparaît pas, car elle n'a pas passé de commande (pas de correspondance dans la table `orders`).
- **Alice** apparaît deux fois, car elle a deux commandes dans `orders`.

Les clauses ON et USING

- **ON:** Spécifie la condition de jointure de manière explicite.

```
SELECT users.name, orders.amount
FROM users
INNER JOIN orders ON users.id = orders.user_id;
```

- **USING:** Utilisé lorsque la colonne de jointure a le même nom dans les deux tables.

```
SELECT users.name, orders.amount
FROM users
INNER JOIN orders USING(id);
```

Les alias de tables

Lorsqu'on en vient à manipuler plusieurs tables, il devient fastidieux de préciser à chaque fois l'entièreté de la table avant chaque colonne. Mais l'**alias de table** nous permet, comme son nom l'indique, de pouvoir renommer notre table, exactement de la même manière qu'on le faisait pour les colonnes.

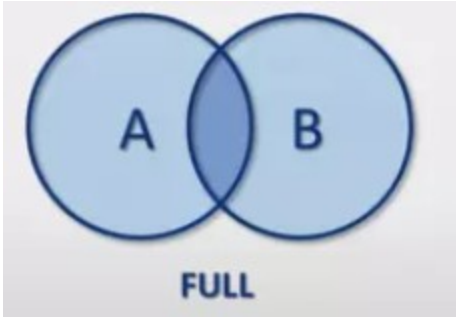
```sql
SELECT u.name, o.amount
FROM users AS u
INNER JOIN orders AS o ON u.id = o.user_id;
```

UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

Les jointures externes

Définition

Les jointures externes (OUTER JOIN) permettent de renvoyer toutes les lignes d'une table, **qu'elles aient une correspondance dans une autre table ou non.**



Contrairement aux jointures internes qui ne renvoient que les lignes ayant une correspondance entre les deux tables, les jointures externes incluent également les lignes sans correspondance.

Les trois types de jointures externes

- **LEFT JOIN:** Renvoie toutes les lignes de la table de gauche (première table dans la jointure), ainsi que les correspondances dans la table de droite.
- **RIGHT JOIN:** Renvoie toutes les lignes de la table de droite, ainsi que les correspondances dans la table de gauche.
- **FULL JOIN:** Renvoie toutes les lignes des deux tables, avec des valeurs nulles pour les colonnes manquantes lorsque les correspondances n'existent pas dans l'une des tables. **ATTENTION: FULL JOIN n'est pas prise en charge dans tous les SGBD.**

Les trois types de jointures externes

Reprenons notre exemple de tout à l'heure avec les élèves et leurs notes.

History_Grades

student_id	student_name	history_grade
1	Alice	A
2	Jane	B
3	Julie	A

Math_Grades

student_id	student_name	math_grade
1	Alice	B
4	Roger	A
5	Kate	C

Les trois types de jointures externes

LEFT JOIN

student_id	student_name	history_grade	student_id	student_name	math_grade
1	Alice	A	1	Alice	B
2	Jane	B			
3	Julie	A			

RIGHT JOIN

student_id	student_name	history_grade	student_id	student_name	math_grade
1	Alice	A	1	Alice	B
			4	Roger	A
			5	Kate	C

Les trois types de jointures externes

FULL JOIN

student_id	student_name	history_grade	student_id	student_name	math_grade
1	Alice	A	1	Alice	B
2	Jane	B			
3	Julie	A			
			4	Roger	A
			5	Kate	C

Syntaxe

- LEFT JOIN

```
SELECT colonne(s)
FROM table1
LEFT JOIN table2 ON table1.colonne = table2.colonne;
```

- RIGHT JOIN

```
SELECT colonne(s)
FROM table1
RIGHT JOIN table2 ON table1.colonne = table2.colonne;
```

La jointure FULL JOIN

Ce type de jointure n'est pas toujours supporté nativement dans certains systèmes, comme MySQL, **mais peut être simulé avec UNION.**

```
SELECT colonne(s)
FROM table1
LEFT JOIN table2 ON table1.colonne = table2.colonne
UNION
SELECT colonne(s)
FROM table1
RIGHT JOIN table2 ON table1.colonne = table2.colonne;
```

Les jointures naturelles

Les jointures naturelles

La NATURAL JOIN est une jointure qui se base **automatiquement** sur les colonnes ayant le même nom dans les deux tables. **C'est un raccourci rapide, mais dangereux** si les noms de colonnes ne sont pas strictement identiques.

Exemple :

```
SELECT users.name, orders.amount
FROM users
NATURAL JOIN orders;
```

Cette requête est équivalente à un INNER JOIN basé sur la colonne commune id.

UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

Auto-jointures

Définition

Une auto-jointure (self-join) est une jointure d'une table avec elle-même.

Elle est utilisée lorsque vous souhaitez comparer les lignes d’une même table entre elles.

Une auto-jointure fonctionne exactement comme une jointure classique, à la différence que vous allez joindre la table à elle-même. Pour que cela fonctionne correctement, vous devez utiliser des alias de table afin de distinguer les instances différentes de la même table dans la requête.

Syntaxe

Syntaxe générale avec `INNER JOIN`:

```sql
SELECT alias1.colonne, alias2.colonne
FROM table AS alias1
INNER JOIN table AS alias2
ON alias1.colonne = alias2.colonne;
```

Vous pouvez aussi utiliser d'autres types de jointures (`LEFT JOIN`, etc.), en fonction du résultat que vous voulez obtenir.

Exemple concret

Table `employees`:

id	name	manager_id
1	Alice	NULL
2	Bob	1
3	Carol	1
4	David	2
5	Eve	2

Dans cet exemple, **Alice** est la superviseure de **Bob** et **Carol**, tandis que **Bob** supervise **David** et **Eve**.
L'objectif de l'auto-jointure ici est de lister les employés avec le nom de leur superviseur.

Exemple concret

```sql
SELECT e1.name AS Employee, e2.name AS Manager
FROM employees e1
INNER JOIN employees e2
ON e1.manager_id = e2.id;
```

- **Alias de table** : Nous avons utilisé `e1` et `e2` comme alias pour représenter deux instances de la table `employees`.
- La condition de jointure `ON e1.manager_id = e2.id` compare les employés avec leurs superviseurs respectifs en utilisant la relation entre `manager_id` et `id`.

Résultat de la jointure

Employee	Manager
Bob	Alice
Carol	Alice
David	Bob
Eve	Bob

Le résultat montre chaque employé avec le nom de son superviseur.

Les produits cartésiens (CROSS-JOIN)

Description du CROSS JOIN

Le CROSS JOIN (ou jointure croisée) est un type de jointure en SQL qui renvoie le produit cartésien des lignes entre deux tables. Cela signifie que chaque ligne de la première table est associée à chaque ligne de la deuxième table. **Le nombre total de lignes du résultat est donc le produit du nombre de lignes des deux tables.**

```
SELECT *
FROM table1
CROSS JOIN table2;
```

Particularités

Les CROSS JOIN **ne nécessitent aucune condition de jointure** entre les deux tables, contrairement aux autres types de jointures (comme INNER JOIN ou LEFT JOIN). Ce type de jointure est rarement utilisé dans les requêtes classiques, car le produit cartésien **peut rapidement devenir très grand si les tables contiennent beaucoup de lignes.**

Bien que puissant pour créer des combinaisons, il est à utiliser avec précaution dans des tables volumineuses, car cela peut entraîner une explosion exponentielle du nombre de résultats.

Exemple concret

Prenons deux tables, `colors` et `shapes`, qui contiennent respectivement des couleurs et des formes géométriques. Nous voulons créer toutes les combinaisons possibles entre les couleurs et les formes.

id	color
1	Red
2	Blue
3	Green

id	shape
1	Circle
2	Square

Nous voulons obtenir toutes les combinaisons possibles entre les couleurs et les formes.

Exemple concret:

```
SELECT colors.color, shapes.shape
FROM colors
CROSS JOIN shapes;
```

Résultat de la jointure :

color	shape
Red	Circle
Red	Square
Blue	Circle
Blue	Square
Green	Circle
Green	Square

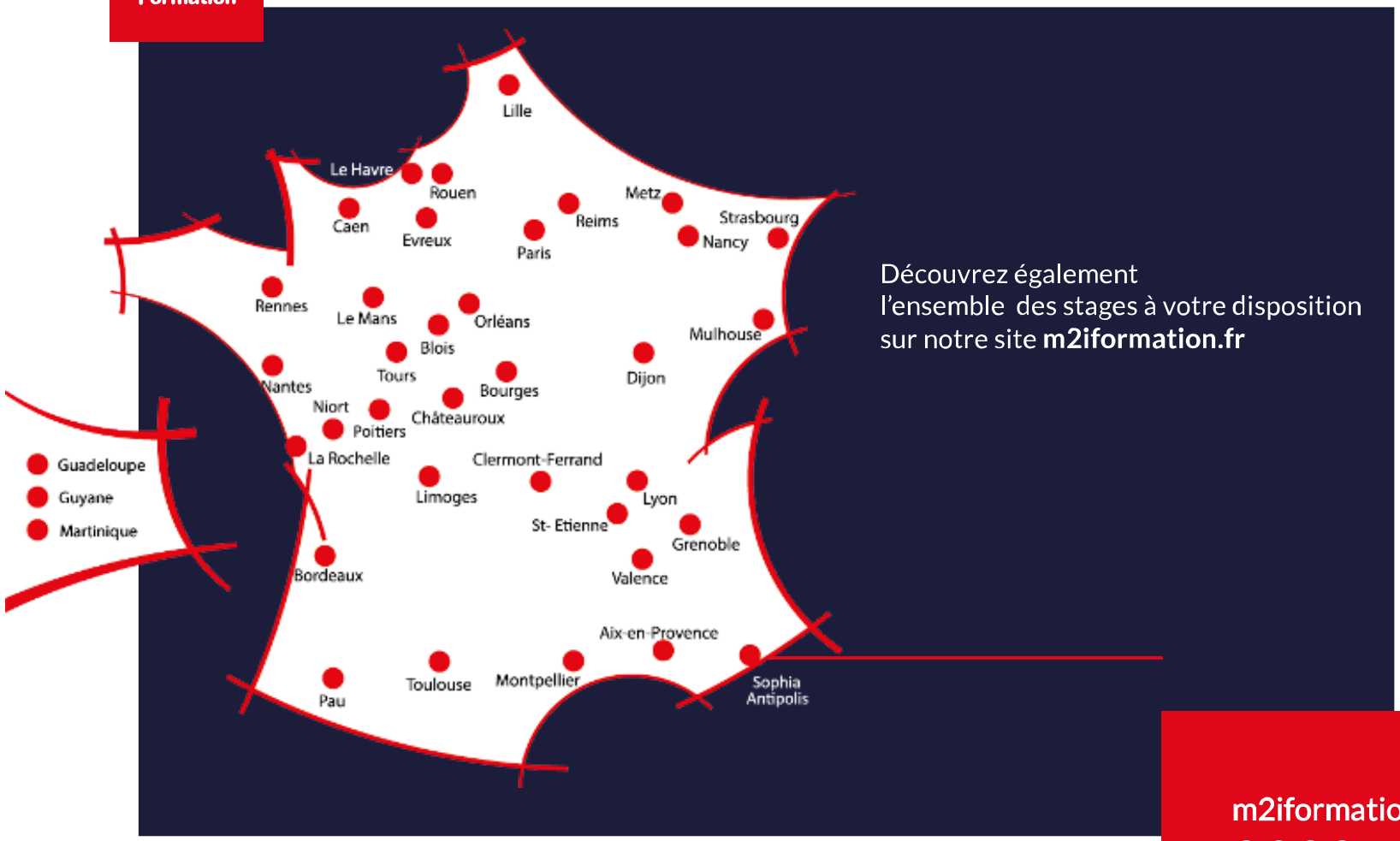
Explication :

- **Produit cartésien** : Chaque ligne de la table `colors` est associée à chaque ligne de la table `shapes`. Étant donné que `colors` contient 3 lignes et `shapes` en contient 2, le résultat final contient **3 × 2 = 6** combinaisons.
- **Aucune condition de jointure** : Le `CROSS JOIN` n'a pas besoin de condition de correspondance entre les tables, car il associe simplement toutes les lignes de la première table avec celles de la deuxième.

UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

Merci pour votre attention

Des questions ?



Découvrez également
l'ensemble des stages à votre disposition
sur notre site **m2iformation.fr**

m2iformation.fr