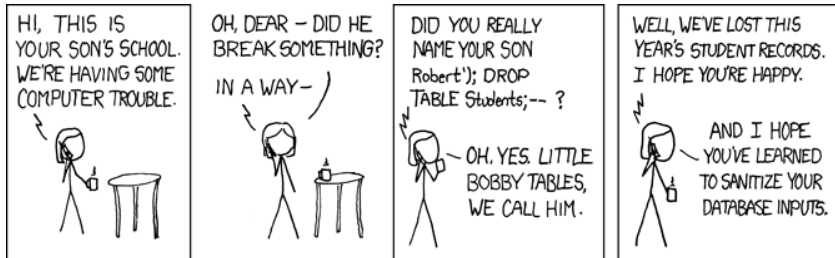


# SQL 2 : Plusieurs tables

## JOIN, UNION, INTERSECT...

Quentin Fortier

July 2, 2022



# Plusieurs tables

Soit  $T$  une table à  $n$  colonnes.

Chaque enregistrement de  $T$  peut être vu comme un  $n$ -uplet.

Donc  $T$  peut être vue comme un ensemble de  $n$ -uplets, c'est-à-dire une **relation** (au sens mathématique).

En voyant les tables comme des ensembles, on peut effectuer des opérations ensemblistes classiques (union, intersection...) ainsi que des opérations plus adaptées (produit cartésien, jointure...).

# Opérations ensemblistes

Si  $R_1$  et  $R_2$  sont des tables ayant le **même schéma relationnel**,  
 $R_1 \cup R_2$  contient les enregistrements dans  $R_1$  ou  $R_2$  :

$R_1$		
$A$	$B$	$C$
$a_1$	$b_1$	$c_1$
$a_2$	$b_2$	$c_2$

$R_2$		
$A$	$B$	$C$
$a_1$	$b_1$	$c_1$
$a_3$	$b_3$	$c_3$

$R_1 \cup R_2$		
$A$	$B$	$C$
$a_1$	$b_1$	$c_1$
$a_2$	$b_2$	$c_2$
$a_3$	$b_3$	$c_3$

En SQL :

**SELECT \* FROM R1 UNION SELECT \* FROM R2;**

⚠ La syntaxe `select * from (R1 UNION R2)` ne fonctionne pas!

# Opérations ensemblistes

Si  $R_1$  et  $R_2$  sont des tables ayant le **même schéma relationnel**,  
 $R_1 - R_2$  contient les enregistrements dans  $R_1$  mais pas dans  $R_2$  :

$R_1$		
$A$	$B$	$C$
$a_1$	$b_1$	$c_1$
$a_2$	$b_2$	$c_2$

$R_2$		
$A$	$B$	$C$
$a_1$	$b_1$	$c_1$
$a_3$	$b_3$	$c_3$

$R_1 - R_2$		
$A$	$B$	$C$
$a_2$	$b_2$	$c_2$

En SQL, on utilise MINUS (MySQL, Oracle) ou **EXCEPT** (PostgreSQL) :

```
SELECT * FROM R1 EXCEPT SELECT * FROM R2;
```

# Opérations ensemblistes

Si  $R_1$  et  $R_2$  sont des tables ayant le **même schéma relationnel**,  
 $R_1 \cap R_2$  contient les enregistrements à la fois dans  $R_1$  et  $R_2$  :

$R_1$		
$A$	$B$	$C$
$a_1$	$b_1$	$c_1$
$a_2$	$b_2$	$c_2$

$R_2$		
$A$	$B$	$C$
$a_1$	$b_1$	$c_1$
$a_3$	$b_3$	$c_3$

$R_1 \cap R_2$		
$A$	$B$	$C$
$a_1$	$b_1$	$c_1$

En SQL :

```
SELECT * FROM R1 INTERSECT SELECT * FROM R2;
```

# Produit cartésien

On peut réaliser le **produit cartésien**  $R_1 \times R_2$  de deux tables :

$R_1$		
$A$	$B$	$C$
$a_1$	$b_1$	$c_1$
$a_2$	$b_2$	$c_2$

$R_2$	
$D$	$E$
$d_1$	$e_1$
$d_2$	$e_2$

$R_1 \times R_2$				
$A$	$B$	$C$	$D$	$E$
$a_1$	$b_1$	$c_1$	$d_1$	$e_1$
$a_1$	$b_1$	$c_1$	$d_2$	$e_2$
$a_2$	$b_2$	$c_2$	$d_1$	$e_1$
$a_2$	$b_2$	$c_2$	$d_2$	$e_2$

En SQL :

**SELECT \* FROM R1, R2;**

On peut aussi sélectionner seulement certaines colonnes de  $R_1 \times R_2$  en écrivant, par exemple, **SELECT A, B FROM R1, R2;**

Considérons une base de donnée bibliotheque avec les tables :

- ❶ livre (titre : CHAR(50), auteur : CHAR(50), pages : INT)
- ❷ emprunteur (id : INT, nom : CHAR(50))
- ❸ emprunt (id\_emprunteur : INT, titre\_livre : CHAR(50))

Dans la table emprunt, id\_emprunteur et titre\_livre sont des **clés étrangères**, ce qui signifie qu'elles font références à une clé primaire d'une autre table.

- ① livre (titre : CHAR(50), auteur : CHAR(50), pages : INT)
- ② emprunteur (id : INT, nom : CHAR(50))
- ③ emprunt (id\_emprunteur : INT, titre\_livre : CHAR(50))

Comment obtenir les emprunteurs qui sont aussi auteurs?

(distinct)

auteur

---

```
SELECT nom FROM emprunteur, livre  
WHERE nom = auteur;
```

---



- ❶ livre (titre : CHAR(50), auteur : CHAR(50), pages : INT)
- ❷ emprunteur (id : INT, nom : CHAR(50))
- ❸ emprunt (id\_emprunteur : INT, titre\_livre : CHAR(50))

Comment obtenir les noms des personnes qui ont emprunté le livre dont le titre est Le Banquet?

---

```
SELECT nom FROM emprunteur, emprunt
WHERE id = id_emprunteur
AND titre_livre = 'Le Banquet';
```

---

# Jointure

La jointure  $R_1 \bowtie_{A=D} R_2$  de deux tables  $R_1$  et  $R_2$  revient à combiner les enregistrements de  $R_1$  et  $R_2$  en identifiant les colonnes  $A$  et  $D$  :

$R_1$		
$A$	$B$	$C$
$a_1$	$b_1$	$c_1$
$a_2$	$b_2$	$c_2$
$a_3$	$b_3$	$c_3$

$R_2$	
$D$	$E$
$a_1$	$e_1$
$a_2$	$e_2$

$R_1 \bowtie_{A=D} R_2$			
$D = A$	$B$	$C$	$E$
$a_1$	$b_1$	$c_1$	$e_1$
$a_2$	$b_2$	$c_2$	$e_2$

En SQL :

```
SELECT ... FROM R1 JOIN R2 ON A = D;
```

## Jointure : LEFT JOIN

Par défaut, un `R1 JOIN R2 ON A = D` est en fait un `INNER JOIN`, ce qui signifie que les enregistrements de `R1` qui ont une valeur de `A` qui n'existe pas dans `D` (et inversement) n'apparaissent pas dans le résultat.

Si on veut conserver tous les enregistrements de `A` (en mettant `NULL` s'il n'y a pas de valeur correspondante de `D`), on peut utiliser un `LEFT JOIN`. Et inversement pour un `RIGHT JOIN`.

# Jointure : LEFT JOIN

$R_1$		
$A$	$B$	$C$
$a_1$	$b_1$	$c_1$
$a_2$	$b_2$	$c_2$
$a_3$	$b_3$	$c_3$

$R_2$	
$D$	$E$
$a_1$	$e_1$
$a_2$	$e_2$

$R_1$ LEFT JOIN $R_2$ ON $A = D$			
$A$	$B$	$C$	$E$
$a_1$	$b_1$	$c_1$	$e_1$
$a_2$	$b_2$	$c_2$	$e_2$
$a_3$	$b_3$	$c_3$	NULL

En SQL :

```
SELECT ... FROM R1 LEFT JOIN R2 ON A = D;
```

# Jointure : Exemples

- ❶ livre (titre : CHAR(50), auteur : CHAR(50), pages : INT)
- ❷ emprunteur (id : INT, nom : CHAR(50))
- ❸ emprunt (id\_emprunteur : INT, titre\_livre : CHAR(50))

Comment obtenir les noms des personnes qui ont emprunté le livre Le Banquet?

On peut aussi utiliser une jointure :

---

```
SELECT nom FROM emprunteur
JOIN emprunt ON id = id_emprunteur
WHERE titre_livre = 'Le Banquet';
```

---

## Jointure : Exemples

- ❶ livre (titre : CHAR(50), auteur : CHAR(50), pages : INT)
- ❷ emprunteur (id : INT, nom : CHAR(50))
- ❸ emprunt (id\_emprunteur : INT, titre\_livre : CHAR(50))

Comment obtenir les titres des livres empruntés par M. Machin?

---

```
SELECT titre_livre FROM emprunteur  
JOIN emprunt ON id = id_emprunteur  
WHERE nom = 'Machin';
```

---

# Jointure : Exemples

- ❶ livre (titre : CHAR(50), auteur : CHAR(50), pages : INT)
- ❷ emprunteur (id : INT, nom : CHAR(50))
- ❸ emprunt (id\_emprunteur : INT, titre\_livre : CHAR(50))

Comment obtenir les noms des personnes ayant emprunté un livre écrit par Stephen King?

---

```
SELECT nom FROM emprunteur
JOIN emprunt ON id = id_emprunteur
JOIN livre ON titre_livre = titre
WHERE auteur = 'Stephen King';
```

---

## Jointure : Auto-jointure

- ❶ livre (titre : CHAR(50), auteur : CHAR(50), pages : INT)
- ❷ emprunteur (id : INT, nom : CHAR(50))
- ❸ emprunt (id\_emprunteur : INT, titre\_livre : CHAR(50))

Comment obtenir les plus gros livres empruntés avec leur nombre de pages?

---

```
SELECT titre, pages FROM livre
JOIN emprunt ON titre_livre = titre
ORDER BY pages DESC;
```

---



## Jointure : Auto-jointure

- ❶ livre (**id** : INT, titre : CHAR(50), auteur : CHAR(50), pages : INT)
- ❷ emprunteur (**id** : INT, nom : CHAR(50))
- ❸ emprunt (id\_emprunteur : INT, titre\_livre : CHAR(50))

Problème : comment savoir, dans livre  $\times$  emprunteur, à quelle table **id** fait référence?

# Jointure : Auto-jointure

- ❶ livre (id INT, titre CHAR(50), auteur CHAR(50), pages INT)
- ❷ emprunteur (id INT, nom CHAR(50))
- ❸ emprunt (id\_emprunteur INT, titre\_livre CHAR(50))

Tentative :

```
SELECT id FROM livre, emprunteur;
```

Résultat :

Column 'id' in field list is ambiguous

## Jointure : Auto-jointure

- ❶ livre (id INT, titre CHAR(50), auteur CHAR(50), pages INT)
- ❷ emprunteur (id INT, nom CHAR(50))
- ❸ emprunt (id\_emprunteur INT, titre\_livre CHAR(50))

Solution :

```
SELECT livre.id FROM livre, emprunteur;
```

## Jointure : Auto-jointure

- ❶ livre (id INT, titre CHAR(50), auteur CHAR(50), pages INT)
- ❷ emprunteur (id INT, nom CHAR(50))
- ❸ emprunt (id\_emprunteur INT, titre\_livre CHAR(50))

Problème 2 : afficher tous les couples de livres ayant le même nombre de pages.

```
SELECT titre, titre FROM livre, livre WHERE pages = pages;
```

Ne marche pas du tout!

## Jointure : Auto-jointure

- ❶ livre (id INT, titre CHAR(50), auteur CHAR(50), pages INT)
- ❷ emprunteur (id INT, nom CHAR(50))
- ❸ emprunt (id\_emprunteur INT, titre\_livre CHAR(50))

Solution : renommer les tables.

---

```
SELECT liv1.titre, liv2.titre
FROM livre AS liv1, livre AS liv2
WHERE liv1.pages = liv2.pages;
```

---

# Jointure : Auto-jointure

On modélise ici un réseau routier par un ensemble de *croisements* et de *voies* reliant ces croisements. Les voies partent d'un croisement et arrivent à un autre croisement. Ainsi, pour modéliser une route à double sens, on utilise deux voies circulant en sens opposés.

La base de données du réseau routier est constituée des relations suivantes :

- Croisement(id, longitude, latitude)
- Voie(id, longueur, id\_croisement\_debut, id\_croisement\_fin)

Dans la suite on considère  $c$  l'identifiant (id) d'un croisement donné.

□ **Q26** – Écrire la requête SQL qui renvoie les identifiants des croisements atteignables en utilisant une seule voie à partir du croisement ayant l'identifiant  $c$ .

□ **Q27** – Écrire la requête SQL qui renvoie les longitudes et latitudes des croisements atteignables en utilisant une seule voie, à partir du croisement  $c$ .

□ **Q28** – Que renvoie la requête SQL suivante ?

```
1  SELECT V2.id_croisement_fin
2  FROM   Voie as V1
3  JOIN   Voie as V2
4  ON     V1.id_croisement_fin = V2.id_croisement_debut
5  WHERE  V1.id_croisement_debut = c
```

Q26: SELECT id-croisement-fin  
FROM Voie JOIN Croisement ON croisement.id = id-croisement-fin  
WHERE id-croisement-debut = c.

Q27: SELECT longitude, latitude  
FROM Voie JOIN Croisement ON croisement.id = id-croisement-fin  
WHERE id-croisement-debut = c

Q28: