Ressource R2.06
Exploitation d'une base de données

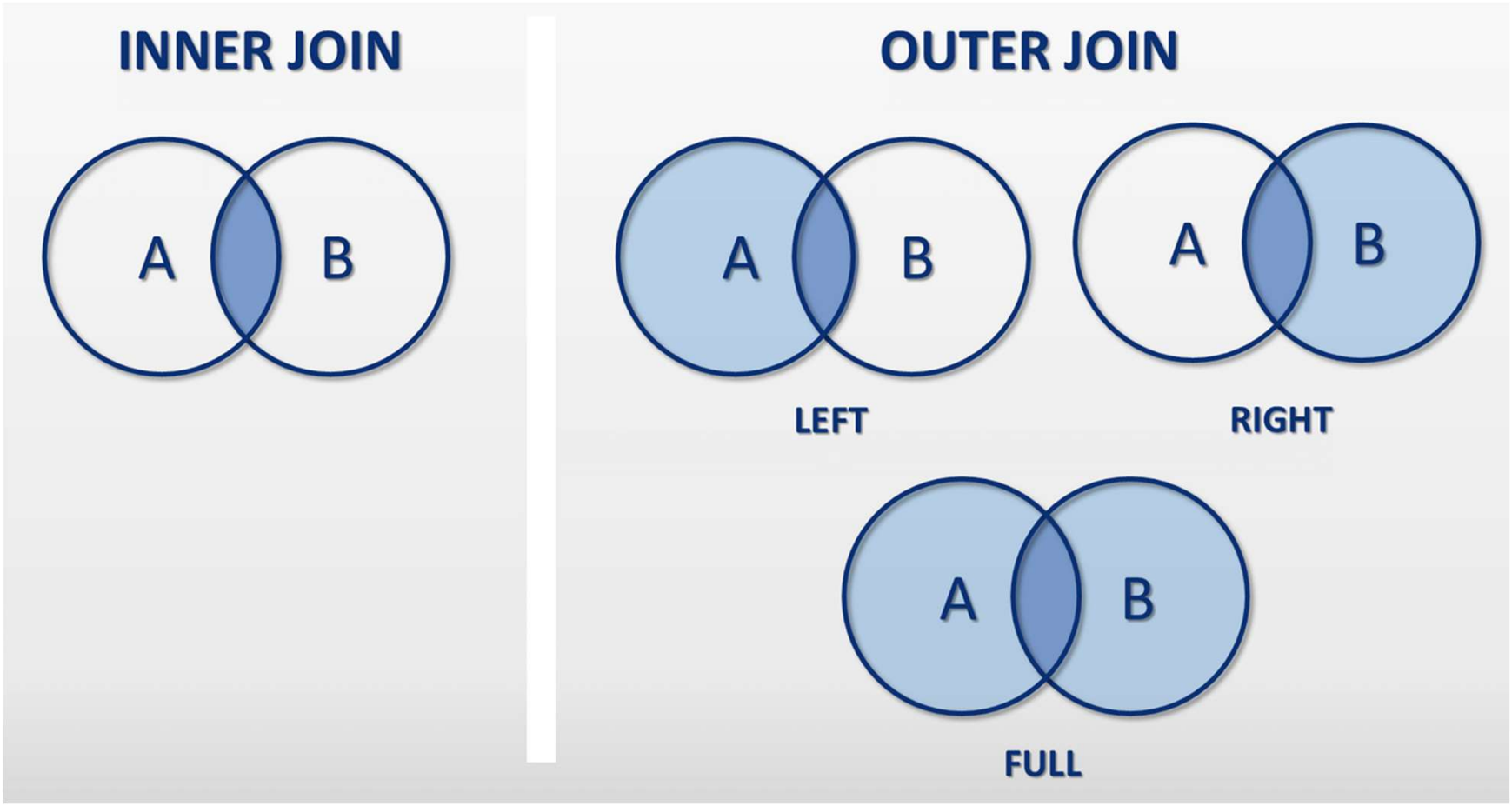
Chapitre 1
SQL Avancé

Mathieu PALOSSE
Professeur de Numérique Sciences Informatiques
IUT d'informatique de Rodez
mathieu.palosse@iut-rodez.fr

Plan

1) Jointures
2) Division
3) Fonctions d’agrégation
4) Grouper/filtrer les résultats
5) Trier les résultats
6) Expressions génériques

1) Jointures



1) Jointures

Synonymes :

- **INNER JOIN**, **JOIN**
- **FULL OUTER JOIN**, **FULL JOIN**
- **LEFT OUTER JOIN**, **LEFT JOIN**
- **RIGHT OUTER JOIN**, **RIGHT JOIN**

Recommandation : utiliser la notation la plus complète permet d'éviter les ambigüités

1) Jointures

CROSS JOIN : correspond au produit cartésien.

Exemple : la syntaxe

```
SELECT *
FROM A
CROSS JOIN B
```

est équivalente à

```
SELECT *
FROM A, B
```

2) Division

La division n'est pas nativement implémentée en SQL

Exemple : R=R1/R2

R1	A	B
	a	u
	a	x
	a	y
	b	y
	b	v
	c	u
	c	x
	c	t
	d	x
	d	y

R2	B
	u
	x

R	A
	a
	c

Toutes les valeurs dans A qui forment dans R1 un tuple avec chacune des valeurs de B dans R2.

2) Division

Syntaxe possible :

```
SELECT DISTINCT A FROM R1 X
WHERE NOT EXISTS (
(SELECT B FROM R2)
MINUS
(SELECT B FROM R1 WHERE X.A = R1.A)
);
```

3) Fonctions d'agrégation

Permettent des opérations statistiques :

- **`AVG()`** pour calculer la moyenne sur un ensemble d'enregistrement
- **`COUNT()`** pour compter le nombre d'enregistrements sur une table ou une colonne distincte
- **`MAX()`** pour récupérer la valeur maximum d'une colonne sur un ensemble de ligne. Cela s'applique à la fois pour des données numériques ou alphanumérique
- **`MIN()`** pour récupérer la valeur minimum de la même manière que **`MAX()`**
- **`SUM()`** pour calculer la somme sur un ensemble d'enregistrement

3) Fonctions d'agrégation

Syntaxe :

```
SELECT agg([DISTINCT] colonne) FROM table
```

où agg est une des fonctions d'agrégation

Exemple de syntaxe simple :

```
SELECT COUNT(*) FROM table
```

Permet de compter le nombre de lignes d'une table (* signifie « toutes colonnes »)

3) Fonctions d'agrégation

Remarque : le paramètre optionnel **DISTINCT** permet de ne prendre en compte que les valeurs différentes.

Exemple :

```
SELECT COUNT(DISTINCT Ville) FROM Client
```

Comptera le nombre de villes différentes dans lesquelles habitent les clients de la table Client.

4) Grouper/filtrer les résultats

Pour effectuer des traitements de groupes, on utilisera les fonctions d'agrégation avec la clause **GROUP BY**.

Exemple :

```
SELECT client, SUM(tarif)
FROM achat
GROUP BY client
```

Pour chaque *client* présent dans la table *achat*, on rassemble les différentes valeurs de *tarif* et on en effectue la *somme*.

4) Grouper/filtrer les résultats

Pour effectuer une restriction sur les résultats d'une commande **GROUP BY**, **WHERE** ne fonctionne pas. Il faut utiliser **HAVING**.

Exemple :

```
SELECT client, SUM(tarif) AS somme
FROM achat
GROUP BY client
HAVING somme> 1000
```

Pour chaque *client* présent dans la table *achat*, on rassemble les différentes valeurs de *tarif* et on en effectue la *somme*, et on ne restitue que ceux dont la *somme* est supérieure à 1000.

4) Grouper/filtrer les résultats

Dans le cas où plusieurs colonnes sont présentes dans la commande **GROUP BY**, leur ordre sera utilisé dans le tri final des résultats.

Exemple 1 :

```
SELECT client, ville, SUM(tarif) AS somme
FROM achat
GROUP BY client, ville
HAVING somme> 1000
```

Le tri est d'abord effectué selon le client, puis selon la ville.

client	ville	somme
DUPOND	Paris	1500
MARTIN	Arcachon	1200
VASQUEZ	Marseille	1100

4) Grouper/filtrer les résultats

Dans le cas où plusieurs colonnes sont présentes dans la commande **GROUP BY**, leur ordre sera utilisé dans le tri final des résultats.

Exemple 2 :

```
SELECT client, ville, SUM(tarif) AS somme
FROM achat
GROUP BY ville, client
HAVING somme> 1000
```

Le tri est d'abord effectué selon le client, puis selon la ville.

client	ville	somme
MARTIN	Arcachon	1200
VASQUEZ	Marseille	1100
DUPOND	Paris	1500

5) Trier les résultats

Les résultats d'une commande **GROUP BY** sont automatiquement triés selon les groupement, mais il peut être utile de réordonner autrement.

On utilise alors **ORDER BY colonne** [ASC/DESC] en fin de requête.

Exemple 1 :

```
SELECT client, ville, SUM(tarif) AS somme
FROM achat
GROUP BY client, ville
HAVING somme> 1000
ORDER BY somme
```

client	ville	somme
VASQUEZ	Marseille	1100
MARTIN	Arcachon	1200
DUPOND	Paris	1500

Pour chaque *client* présent dans la table *achat*, on rassemble les différentes valeurs de *tarif* et on en effectue la *somme*, et on ne restitue que ceux dont la *somme* est supérieure à 1000, ordonné par *somme* croissante (implicitement ASC)

5) Trier les résultats

Les résultats d'une commande `GROUP BY` sont automatiquement triés selon les groupement, mais il peut être utile de réordonner autrement.

On utilise alors **`ORDER BY colonne [ASC/DESC]`** en fin de requête.

Exemple 2 :

```
SELECT client, ville, SUM(tarif) AS somme
FROM achat
GROUP BY client, ville
HAVING somme> 1000
ORDER BY somme DESC
```

client	ville	somme
DUPOND	Paris	1500
MARTIN	Arcachon	1200
VASQUEZ	Marseille	1100

Pour chaque *client* présent dans la table *achat*, on rassemble les différentes valeurs de *tarif* et on en effectue la *somme*, et on ne restitue que ceux dont la *somme* est supérieure à 1000, ordonné par *somme* décroissante (`DESC` explicitement)

6) Expressions génériques : LIKE

L'opérateur **LIKE** permet de rechercher un modèle particulier.

Syntaxe :

```
SELECT *
FROM table
WHERE colonne LIKE modele
```

Les deux caractères spéciaux (*wildcards*) supportés par l'opérateur **LIKE** de Oracle sont % et _ :

- % : représente zéro, un ou plusieurs caractères quelconques.
- _ : représente un seul caractère quelconque.

Remarque : l'implémentation des caractères spéciaux dépend du SGBD et varie de l'un à l'autre.

Lien vers la doc Oracle :
https://docs.oracle.com/cd/F49540_01/DOC/inter.815/a67843/cqspcl.htm#20687

6) Expressions génériques : LIKE

Exemples :

1)

```
SELECT * FROM table WHERE colonne LIKE 'a%'
```

Requête SQL cherchant les enregistrements débutant par la lettre « a ».

2)

```
SELECT * FROM table WHERE colonne LIKE '%a'
```

Requête SQL cherchant les enregistrements terminant par la lettre « a ».

6) Expressions génériques : LIKE

Exemples :

3)

```
SELECT * FROM table WHERE colonne LIKE '%a%'
```

Requête SQL cherchant les enregistrements qui possèdent la lettre « a ».

4)

```
SELECT * FROM table WHERE colonne LIKE '_a%'
```

Requête SQL cherchant les enregistrements qui possèdent la lettre « a » en deuxième caractère.

6) Expressions génériques : LIKE

Exemples :

5)

```
SELECT * FROM table WHERE colonne LIKE 'a%z'
```

Requête SQL cherchant les enregistrements qui débutent par la lettre « a » et se terminent par la lettre « z »

6) Expressions génériques : REGEXP_LIKE

Pour une utilisation plus avancée, Oracle implémente les expressions régulières étendues POSIX ERE à l'aide de la fonction **REGEXP_LIKE()**

(https://docs.oracle.com/cd/B12037_01/server.101/b10759/conditions018.htm)

Syntaxe :

```
SELECT colonne
FROM table
WHERE REGEXP_LIKE(colonne, modele [,option] )
```

modele est une chaine de caractère (entre ' '), contenant des caractères spéciaux POSIX

6) Expressions génériques : REGEXP_LIKE

Caractères spéciaux pour écrire un modèle POSIX :

(https://docs.oracle.com/cd/B12037_01/server.101/b10759/ap_posix001.htm#i690819)

`^` : caractère de début de ligne (à placer en premier)

`$` : caractère de fin de ligne (à placer en dernier)

`.` : un caractère quelconque (sauf NULL)

`*` : chercher 0 ou plusieurs occurrences d'un motif (à placer après le motif)

`+` : chercher 1 ou plusieurs occurrences d'un motif (à placer après le motif)

`?` : chercher 0 ou 1 occurrence d'un motif (à placer après le motif)

`|` : opérateur "OU" pour définir des motifs alternatifs

`[]` : liste de caractères pouvant intervenir dans l'expression. Si "^" est present au début, les caractères ne doivent pas intervenir.

`()` : groupement d'expression, traité comme une unique sous-expression

`{m}` : Motif répété exactement m fois (à placer après le motif)

`{m,}` : Motif répété au moins m fois (à placer après le motif)

`{m,n}` : Motif répété entre m et n fois (à placer après le motif)

6) Expressions génériques : REGEXP_LIKE

Classes de caractères :

[:class:] : spécifie une classe de caractères. Correspond à n'importe quel caractère appartenant à la classe.

Exemples de classes de caractères :

[:alnum:] : tout caractère alphanumerique
[:alpha:] : tout caractère alphabetique
[:digit:] : tout caractère chiffre
[:lower:] : tout caractère minuscule
[:upper:] : tout caractère majuscule
[:blank:] : tout caractère espace ou tabulation.
[:space:] : tout caractère non imprimable (espaces, tab, retour à la ligne)
[:punct:] : tout caractère ponctuation

6) Expressions génériques : REGEXP_LIKE

Options :

'c' : sensible à la case

'i' : non sensible à la case.

Si 'c' et 'i' ne sont pas présents, 'c' est activé par défaut.

'n' : autorise le point ".", correspondant à tout caractère non NULL, de correspondre au caractère "newline". Sans ce parameter, cette option est désactivée par défaut.

'm' : traite la chaine d'entrée comme ayant plusieurs lignes. Oracle interprète ^ et $ comme le début et la fin, respectivement de n'importe quelle ligne de la chaine, plutôt que de les interpreter comme le début et la fin de la totalité de la chaine. Sans ce paramètre, Oracle traite la chaine source comme une seule ligne.

6) Expressions génériques : REGEXP_LIKE

Exemples :

1) Requête SQL cherchant les enregistrements débutant par la lettre « a » :

```
SELECT *
FROM table
WHERE REGEXP_LIKE(colonne, '^a')
```

6) Expressions génériques : REGEXP_LIKE

Exemples :

2) Requête SQL cherchant les enregistrements terminant par la lettre « a » :

```
SELECT *
FROM table
WHERE REGEXP_LIKE(colonne, 'a$')
```

6) Expressions génériques : REGEXP_LIKE

Exemples :

3) Requête SQL cherchant les enregistrements qui possèdent la lettre « a » :

```
SELECT *
FROM table
WHERE REGEXP_LIKE(colonne, 'a')
```

6) Expressions génériques : REGEXP_LIKE

Exemples :

4) Requête SQL cherchant les enregistrements qui possèdent la lettre « a » en deuxième caractère :

```
SELECT *
FROM table
WHERE REGEXP_LIKE(colonne, '^.a')
```

6) Expressions génériques : REGEXP_LIKE

Exemples :

5) Requête SQL cherchant les enregistrements qui débutent par la lettre « a » et se terminent par la lettre « z » :

```
SELECT *
FROM table
WHERE REGEXP_LIKE(colonne, '^a.*z$')
```

6) Expressions génériques : REGEXP_LIKE

Exemples :

6) Requête SQL cherchant les enregistrements qui possèdent la lettre « a » , minuscule ou majuscule :

```
SELECT *
FROM table
WHERE REGEXP_LIKE(colonne, 'a', 'i')
```

6) Expressions génériques : REGEXP_LIKE

Exemples :

7) Requête SQL cherchant les enregistrements qui possèdent un (au moins) caractère espace :

```
SELECT *
FROM table
WHERE REGEXP_LIKE(colonne, '[[:blank:]]')
```

6) Expressions génériques : REGEXP_LIKE

Exemples :

8) Requête SQL cherchant les enregistrements qui possèdent un (au moins) caractère ponctuation :

```
SELECT *
FROM table
WHERE REGEXP_LIKE(colonne, '[[:punct:]]')
```

6) Expressions génériques : REGEXP_LIKE

Exemples :

9) Requête SQL cherchant les enregistrements qui commencent par une majuscule :

```
SELECT *
FROM table
WHERE REGEXP_LIKE(colonne, '^[[:upper:]]')
```

6) Expressions génériques : REGEXP_LIKE

Exemples :

10) Requête SQL cherchant les enregistrements qui commencent par soit « A », soit « B », soit « C » :

```
SELECT *
FROM table
WHERE REGEXP_LIKE(colonne, '^[ABC]')
```

6) Expressions génériques : REGEXP_LIKE

Exemples :

11) Requête SQL cherchant les enregistrements qui ne commencent ni par « A », ni par « B », ni par « C » :

```
SELECT *
FROM table
WHERE REGEXP_LIKE(colonne, '^[^ABC]')
```

6) Expressions génériques : REGEXP_LIKE

Exemples :

12) Requête SQL cherchant les enregistrements qui contiennent le motif « aa » :

```
SELECT *
FROM table
WHERE REGEXP_LIKE(colonne, 'a{2}')
```

6) Expressions génériques : REGEXP_LIKE

Exemples :

13) Requête SQL cherchant les enregistrements qui contiennent 5 caractères exactement :

```
SELECT *
FROM table
WHERE REGEXP_LIKE(colonne, '^.{5}$')
```

6) Expressions génériques : REGEXP_LIKE

Exemples :

14) Requête SQL cherchant les enregistrements qui contiennent de 4 à 6 caractères :

```
SELECT *
FROM table
WHERE REGEXP_LIKE(colonne, '^.{4,6}$')
```

Annexe : REGEX POSIX

Pour plus d'aide sur l'utilisation des REGEX POSIX :

https://www.regular-expressions.info/tutorial.html