

# SQL – LANGAGE D'INTERROGATION DE DONNÉES (LID)

Informatique Tronc Commun  
Bases de données  
E. CLERMONT

1

- Le langage SQL (Structured Query Language) est le langage normalisé de gestion de tout type de bases de données relationnelles, contrairement aux assistants fournis par certains logiciels (par exemple Access).

En fait, lorsqu'on réalise une requête en mode graphique (assistant graphique), le SGBD génère le code SQL associé.

- Le langage SQL comporte une subdivision regroupant les éléments nécessaires à la réalisation de **requêtes d'interrogation des données** :

## le Langage d'Interrogation des Données.

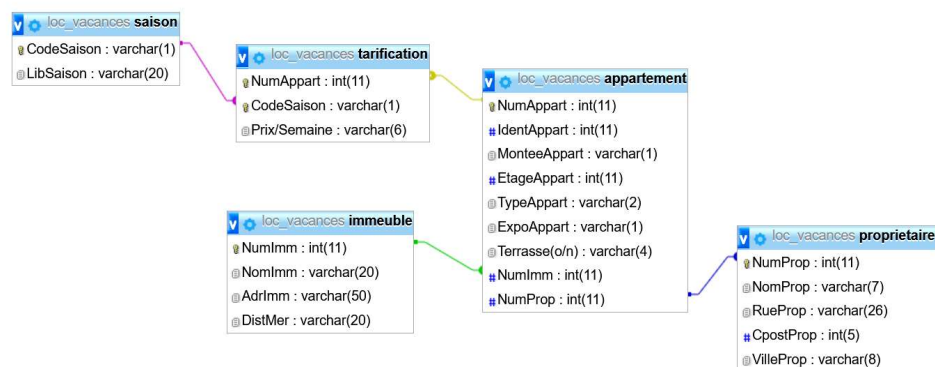
- Une requête d'interrogation est une **extraction** de données issues d'une combinaison de **sélections** et/ou **opérations** portant **sur une ou plusieurs tables** d'une base de données.
- Le **résultat de la requête** est lui-même une **table** dont l'existence est cependant éphémère (le temps de la requête).

2

- Pour illustrer les notions abordées, nous nous plaçons dans la société LOC VACANCES qui est une agence immobilière spécialisée dans la location d'appartements meublés.
- Pour la gestion de son parc locatif, elle a fait le choix de mettre en place une base de données évolutive sur laquelle s'appuieront progressivement tous les traitements que nécessitent une gestion immobilière de ce type (planning des locations, réservations d'appartements, publipostages des confirmations, gestion des acomptes, facturation, etc. ..).

3

## Représentation graphique du schéma relationnel associé



- Chaque attribut composant la clé primaire est précédé d'une clé
  - Chaque clé étrangère est représentée par un lien entre table
  - DistMer : distance de l'appartement jusqu'à la mer
- SAISON ( CodeSaison, LibSaison)
- PROPRIETAIRE (NumProp, NomProp, RueProp, CPostProp, VilleProp)
- IMMEUBLE ( NumImm, NomImm, AdriImm, DistMer)
- APPARTEMENT ( NumAppart, IdentAppart, MonteeAppart, EtageAppart, TypaAppart, ExpoAppart, Terrasse(o/n), NumImm#, NumProp#)
- TARIFICATION (NumAppart#, CodeSaison#, Prix/Semaine)

4

## SOMMAIRE

### 1-Interrogation portant sur une seule table

#### 1.1- Projection

#### 1.2- Restriction

#### 1.3- Tri

#### 1.4-Alias du Select

#### 1.5- Limit et Offset

#### 1.5- Calculs


### 2-Fonctions agrégats

### 3-Interrogation simple portant sur plusieurs tables

### 4-Regroupement

### 5-Les conditions sur regroupement

5

 Syntaxe générale d'une requête d'interrogation simple sur une table :

```
SELECT [DISTINCT] champ1 [, champ2 ...]  
FROM table  
[WHERE condition1 [ OR/AND condition2... ]  
[ORDER BY champ1 [ASC]/DESC] [, champ2 [ASC]/DESC]... ] ;
```

#### Remarques :

Les instructions notées entre [ ] sont facultatives.

Dans la clause **ORDER BY**, le classement par défaut est ascendant (**ASC**).

Les points de suspension indiquent une répétition possible n fois.

6

### 1.1- La projection

La **projection** est une opération relationnelle qui consiste, au travers d'une requête, à ne **retenir que certains champs** (colonnes) dans la table résultat(s).

Ch1	Ch2	Ch3	Ch4

L'ordre d'affichage dépend de l'ordre d'énumération dans la clause **SELECT**.

Exemple1 : Requête R1.1\_1 « Liste des appartements (toutes les informations)»

REQUETE SQL	EXPLICATIONS
<b>SELECT *</b> <b>FROM APPARTEMENT</b>	<b>SELECT</b> : opérateur de projection Le caractère * permet d'afficher TOUS les champs de la table précisée dans la clause FROM. La clause FROM est suivie du nom des tables utiles à la réalisation de la requête.

Exemple2 : Requête R1.1\_2 «Liste des appartements (Type d'appartement et numéro d'appartement)».

REQUETE SQL	EXPLICATIONS
<b>SELECT TypeAppart,</b> <b>IdentAppart</b> <b>FROM APPARTEMENT</b>	<b>SELECT</b> : opérateur de projection Il est suivi de la liste des attributs à projeter séparés par une virgule

Exemple3 : Requête R1.1\_3 «Liste des différents types d'appartements en location chez LOC\_VACANCES».

REQUETE SQL	EXPLICATIONS
<b>SELECT DISTINCT</b> <b>TypeAppart</b> <b>FROM APPARTEMENT</b>	<b>SELECT</b> : opérateur de projection Il est suivi de la clause <b>DISTINCT</b> qui permet d'afficher uniquement les lignes de résultat ayant des valeurs distinctes. Si plusieurs appartements de type T2 sont à la location, le type T2 ne sera affiché qu'une seule fois dans le résultat si <b>DISTINCT</b> est présent.

### 1.2- La restriction

La **restriction** est une opération relationnelle qui vise, au travers d'une requête, à ne retenir que les tuples (*lignes*) de la table satisfaisant le(s) **critère(s) de restriction**.

Ch1	Ch2	Ch3	Ch4

Les restrictions portant sur des champs de type texte devront être mentionnées entre guillemets.

Exemple 1 : Requête R1.2\_1 « Appartements situés au rez-de-chaussée ».

REQUETE SQL	EXPLICATIONS
<b>SELECT *</b> <b>FROM APPARTEMENT</b> <b>WHERE EtageAppart = 0</b>	•WHERE est suivi de l'ensemble des restrictions (conditions) de la requête.

Requête R1.2\_1b « Appartements orientés au Nord ».

**SELECT \***  
**FROM APPARTEMENT**  
**WHERE ExpoAppart = 'N'**

Les **conditions** peuvent être construites à partir :

- d'*expressions* constituées de noms de champs ou de valeurs et éventuellement d'opérateurs arithmétiques (+, -, /, \*) et fonctions prédéfinies (somme, moyenne, ...)
- d'opérateurs de *comparaison* : >, <, >=, <=, <> (différent)
- d'opérateurs logiques : **AND, OR, NOT**
- d'opérateurs SQL : **BETWEEN... AND, IN, LIKE**

Il est également possible de tester si un champ n'est pas renseigné (valeur indéfinie) grâce à la valeur **NULL**.

Pour exprimer ce type de condition l'opérateur = n'est pas accepté on devra écrire :

**WHERE NomChamp IS NULL**

#### **Attention :**

Lorsque un champ contient la **valeur zéro** ou un **espace (qui correspond à une chaîne vide : " ")**, la valeur est définie.



**Exemple2 : Requête R1.2\_2** «Liste des appartements situés au 1<sup>er</sup> étage et exposés au Sud (codé 'S')».

REQUETE SQL

```
SELECT *
FROM APPARTEMENT
WHERE EtageAppart = 1
AND ExpoAppart = 'S'
```

**Exemple3 : Requête R1.2\_3** «Liste des appartements non exposés au nord (codé "N")».

REQUETE SQL version 1	REQUETE SQL version 2
<pre>SELECT * FROM APPARTEMENT WHERE NOT ExpoAppart = 'N' Ou WHERE ExpoAppart &lt;&gt; 'N'</pre>	<pre>SELECT * FROM APPARTEMENT WHERE ExpoAppart IN ('S', 'E', 'O')</pre>

11

**Exemple4 : Requête R1.2\_4** «Liste des noms de propriétaires habitant dans le département 64».

Code postal est de type alphanumérique (texte)	Code postal est de type alphanumérique
<pre>SELECT NomProp FROM PROPRIETAIRE WHERE CpostProp LIKE '64%'</pre>	<pre>SELECT NomProp FROM PROPRIETAIRE WHERE CpostProp &gt;= 64000 AND CpostProp &lt; 65000 ou SELECT NomProp FROM PROPRIETAIRE WHERE CpostProp BETWEEN 64000 AND 64999</pre>

Le "joker" % remplace de 0 à n caractères quelconques.

**Exemple** : Si l'on écrit après la commande **WHERE** attribut **LIKE** "F%", la condition portera sur toutes les valeurs de l'attribut dont la première lettre commence par F.

Le "joker" \_ remplace un caractère quelconque et un seul.

**Exemple** : Si l'on écrit après la commande **WHERE** attribut **LIKE** "F\_\_", la condition portera sur toutes les valeurs de l'attribut dont la première lettre commence par F et ayant ensuite seulement 2 autres caractères.

### 1.3- Le tri : clause **ORDER BY**

Dans la clause **ORDER BY**, le sens de classement peut prendre 2 valeurs :

- **ASC** (ascendant ou croissant)
- ou **DESC** (descendant ou décroissant).

Le sens par défaut est le sens ascendant.

Exemple1 : **Requête R1.3\_1** «Liste des numéros (IdentAppart) de tous les appartements par exposition».

<b>SELECT</b> ExpoAppart, IdentAppart	<b>SELECT</b> ExpoAppart, IdentAppart
<b>FROM</b> APPARTEMENT	<b>FROM</b> APPARTEMENT
<b>ORDER BY</b> ExpoAppart	<b>ORDER BY</b> 1

Exemple2 : **Requête R1.3\_2** «Liste des appartements triés par exposition (décroissant) puis par niveau (du plus élevé au moins élevé)».

```
SELECT *  
FROM APPARTEMENT  
ORDER BY ExpoAppart DESC, EtageAppart DESC
```

13

### 1.4- **LIMIT** et **OFFSET**

La commande **LIMIT** permet de **spécifier le nombre maximum de résultats** que l'on souhaite obtenir,

**Exemple** : « Afficher les 10 premiers appartements »

```
SELECT *  
FROM APPARTEMENT  
LIMIT 10
```

**Exemple** : « Afficher les numéros des 3 appartements les plus chers en haute-saison »

```
SELECT NumAppart, `Prix/Semaine`  
FROM tarification  
WHERE CodeSaison ='H'  
ORDER BY `Prix/Semaine` DESC  
LIMIT 3
```

14

#### 1.4- LIMIT et OFFSET

La commande **Offset** s'utilise **en complément** de la commande **LIMIT**. Elle permet d'**effectuer un décalage** sur l'ensemble des résultats obtenus avec la commande **LIMIT** en décalant le nombre de résultats.

**Exemple : « Afficher 10 appartements à partir du 4<sup>ème</sup> »**

```
SELECT *  
FROM APPARTEMENT  
LIMIT 10  
OFFSET 3
```

15

#### 1.5- Les alias du SELECT (pour l'affichage)

Il est possible de renommer, temporairement pour une requête, l'entête d'une colonne (une expression ou un champ d'une table) dans la table résultat grâce à la clause **AS**.

Si le nom spécifié est composé de plusieurs mots, il faudra l'encadrer par des crochets sous Access (mais cela peut être différent avec d'autres SGBD).

L'alias défini dans le **SELECT** ne peut pas être utilisé dans les autres clauses de la requête (impossible d'y faire référence, il faudra réécrire l'expression ou le champ associé (cf. 1.5)).

Exemple : **Requête R1.4** «Prix de chaque appartement en haute saison».

REQUETE SQL	RESULTAT																
SELECT NumAppart, 'Prix/Semaine' AS 'Prix haute saison' FROM TARIFICATION WHERE CodeSaison= 'H'	<table><tr><th>NumAppart</th><th>Prix haute saison</th></tr><tr><td>1</td><td>487.84</td></tr><tr><td>2</td><td>655.53</td></tr><tr><td>3</td><td>823.22</td></tr><tr><td>4</td><td>823.22</td></tr><tr><td>5</td><td>609.80</td></tr><tr><td>6</td><td>487.84</td></tr><tr><td>7</td><td>487.84</td></tr></table>	NumAppart	Prix haute saison	1	487.84	2	655.53	3	823.22	4	823.22	5	609.80	6	487.84	7	487.84
NumAppart	Prix haute saison																
1	487.84																
2	655.53																
3	823.22																
4	823.22																
5	609.80																
6	487.84																
7	487.84																



### 1.5- Les alias du SELECT (pour les calculs)

Des expressions arithmétiques ou logiques (booléennes) peuvent être présentes dans le SELECT.

Elles sont traitées de la même manière que les champs issus de tables.

L'en-tête de colonne des champs « calculés » n'ayant pas de nom affiche le calcul effectué.

Ces calculs sont appelés « calculs en ligne » car ils sont répétés de manière indépendante sur chaque enregistrement.

Exemple1 : Requête R1.5\_1 «Prix de deux semaines de location pour chaque appartement».

REQUETE SQL	RESULTAT		
SELECT NumAppart,CodeSaison, `Prix/Semaine`*2 AS Total FROM TARIFICATION	NumAppart	CodeSaison	Total
	1	B	365.88
	1	H	975.68
	1	M	1000.00
	2	B	548.82

Exemple2 : Requête R1.5\_2 «Appartements dont le prix de location pour deux semaines en haute saison est inférieur à 950 €».

REQUETE SQL	RESULTAT								
<pre>SELECT NumAppart, `Prix/Semaine`*2 AS `(Px hte *2)&lt;950€` FROM TARIFICATION WHERE CodeSaison= 'H' AND (`Prix/Semaine`*2 ) &lt;950</pre>	<table> <tr> <th>NumAppart</th><th>(Px hte *2)&lt;950€</th></tr> <tr> <td>12</td><td>800.50</td></tr> <tr> <td>26</td><td>900.00</td></tr> <tr> <td>34</td><td>914.70</td></tr> </table>	NumAppart	(Px hte *2)<950€	12	800.50	26	900.00	34	914.70
NumAppart	(Px hte *2)<950€								
12	800.50								
26	900.00								
34	914.70								

Ici, un alias a été utilisé afin de donner un nom explicite à la colonne résultat. Cependant, cet alias ne peut pas être utilisé dans le critère de restriction. Il n'est valable que dans la clause SELECT.

Des fonctions prédéfinies peuvent également être utilisées :

**YEAR(champ)**

Renvoie l'année d'une valeur stockée dans un champ de type date

**ROUND(champ, nombre décimales)**

Renvoie la valeur arrondie d'une valeur stockée dans un champ numérique en conservant un nombre spécifié de décimales. Si le second paramètre n'est pas mentionné, la valeur par défaut est 0. Ceci signifie que la fonction renverra la valeur entière supérieure.

19

**Exemple3 : Requête R1.5\_3** «Prix de location de deux semaines pour chaque appartement».

REQUETE SQL	RESULTAT		
	NumAppart	CodeSaison	Px arrondi
<b>SELECT</b> NumAppart,CodeSaison, ROUND('Prix/Semaine'*2) AS 'Px arrondi' <b>FROM</b> tarification ;	1	B	364
	1	H	974
	1	M	1000
	2	B	548

20

### 1.6 Les opérations ensemblistes

Les opérations ensemblistes en SQL, sont celles définies dans l'algèbre relationnelle. Elles sont réalisées grâce aux opérateurs :

- **UNION**

- **INTERSECT** (ne fait pas partie de la norme SQL => pas implémenté dans tous les SGBD)

- **EXCEPT** (ne fait pas partie de la norme SQL => pas implémenté dans tous les SGBD)

Syntaxe :  
**SELECT ...**  
**{UNION | INTERSECT | EXCEPT}**  
**SELECT ...**

Dans une requête utilisant des opérateurs ensemblistes :

- Tous les **SELECT** doivent avoir le même nombre de colonnes sélectionnées, et leur types doivent être un à un identiques.
- Les doublons sont éliminés (**DISTINCT** implicite).
- Les noms de colonnes sont ceux du 1<sup>er</sup> **SELECT**.
- La largeur des colonnes est la plus grande parmi tous les **SELECT**.
- On ne peut trouver qu'un seul **ORDER BY**. S'il est présent, il doit être mis dans le dernier **SELECT** et il ne peut faire référence qu'aux numéros des colonnes et non pas à leurs noms (car les noms peuvent être différents dans chacune des interrogations)

(21)

### 1.6 Les opérations ensemblistes

#### L'opérateur **UNION**

- Cet opérateur permet d'effectuer une **UNION** des tuples sélectionnés par deux clauses **SELECT** (les deux tables sur lesquelles on travaille devant avoir le même schéma).

**SELECT ... FROM ... WHERE ...**

**UNION**

**SELECT ... FROM ... WHERE ...**

- Par défaut les doublons sont automatiquement éliminés. Pour conserver les doublons, il est possible d'utiliser une clause **UNION ALL**

- Exemple : Requête R1.6\_1 « Liste des appartements qui sont soit au sud soit au 1<sup>er</sup> étage ».

**SELECT \* FROM appartement**

**WHERE EtageAppart = 1**

**UNION**

**SELECT \* FROM appartement**

**WHERE ExpoAppart ='S'**

(22)

### 1.6 Les opérations ensemblistes

#### L'opérateur **INTERSECT**

Cet opérateur permet d'effectuer une INTERSECTION des tuples sélectionnés par deux clauses *SELECT* (les deux tables sur lesquelles on travaille devant avoir le même schéma).

```
SELECT ...FROM ... WHERE ...  
INTERSECT  
SELECT ... FROM ... WHERE ...
```

- **Exemple** : Requête R1.6\_2 «Liste des appartements qui sont soit à la fois orientés au sud et au rez-de -chaussée ».

```
SELECT * FROM APPARTEMENT  
WHERE EtageAppart = 0  
  
INTERSECT  
  
SELECT * FROM APPARTEMENT  
WHERE ExpoAppart ='S'
```

```
SELECT * FROM APPARTEMENT  
WHERE EtageAppart = 0  
AND ExpoAppart ='S'  
Ou  
SELECT * FROM APPARTEMENT  
WHERE EtageAppart = 0  
AND NumAppart IN (  
    SELECT NumAppart  
    FROM APPARTEMENT  
    WHERE ExpoAppart ='S')
```

(23)

### 1.6 Les opérations ensemblistes

#### L'opérateur **EXCEPT**

et opérateur permet d'effectuer une DIFFERENCE entre les tuples sélectionnés par deux clauses *SELECT*, c'est-à-dire sélectionner les tuples de la première table n'appartenant pas à la seconde (les deux tables devant avoir le même schéma).

```
SELECT ... FROM ... WHERE ...  
EXCEPT  
SELECT ... FROM ... WHERE ...
```

- **Exemple** : Requête R1.6\_2 «Liste de tous appartements qui ne sont pas orientés au nord ».

```
SELECT *  
  
FROM APPARTEMENT  
  
EXCEPT  
  
SELECT *  
  
FROM APPARTEMENT  
  
WHERE ExpoAppart='N'
```

```
SELECT * FROM APPARTEMENT  
WHERE ExpoAppart <>'N'  
Ou  
SELECT * FROM APPARTEMENT  
WHERE NumAppart NOT IN (  
    SELECT NumAppart  
    FROM APPARTEMENT  
    WHERE ExpoAppart ='N')
```

(24)

## 1.6 Les opérations ensemblistes

### Le produit cartésien

permet de retourner chaque ligne d'une table avec toutes les lignes d'une autre table.

```
SELECT ...  
FROM table1, table2 [...]  
WHERE ...
```

#### ▪ Exemple :

```
SELECT *  
FROM saison, appartement
```

```
SELECT ...  
FROM table1 CROSS JOIN table2  
[...]  
WHERE ...
```

```
SELECT *  
FROM saison CROSS JOIN  
appartement
```

CodeSaison	LibSaison	NumAppart	IdentAppart	MonteeAppart	EtageAppart	TypeAppart	ExpoAppart	Terrasse(o/n)	NumImm	NumProp
B	Basse Saison	1	101	G	1	T1	S	oui	1	1
H	Haute Saison	1	101	G	1	T1	S	oui	1	1
M	Moyenne Saison	1	101	G	1	T1	S	oui	1	1
B	Basse Saison	2	102	G	1	T2	S	oui	1	4
H	Haute Saison	2	102	G	1	T2	S	oui	1	4
M	Moyenne Saison	2	102	G	1	T2	S	oui	1	4
B	Basse Saison	3	103	G	1	T6	S	oui	1	4
H	Haute Saison	3	103	G	1	T6	S	oui	1	4
M	Moyenne Saison	3	103	G	1	T6	S	oui	1	4

A manier avec précaution



## SOMMAIRE

1-Interrogation portant sur une seule table

2-Fonctions agrégats

3-Interrogation simple portant sur plusieurs tables

4-Regroupement

5-Les conditions sur regroupement

Elles permettent d'effectuer des calculs verticaux (en colonne) pour l'ensemble ou un sous-ensemble des valeurs d'une colonne.

(Le calcul porte sur un champ unique, mais concerne plusieurs enregistrements).

Les fonctions principales sont :

Fonctions	Symboles
SUM	permet d'effectuer la somme des valeurs d'une colonne numérique
AVG	permet d'effectuer la moyenne des valeurs d'une colonne numérique
MAX	permet de rechercher la valeur maximale d'une colonne numérique
MIN	permet de rechercher la valeur minimale d'une colonne numérique
COUNT	permet de compter le nombre de valeurs d'une colonne numérique

Exemple1 : Requête R2\_1 «Nombre d'appartements en location».

```
SELECT COUNT(*) AS 'Nombre d appartements'
FROM APPARTEMENT
```

ou

```
SELECT COUNT(NoAppart) AS 'Nombre d appartements'
FROM APPARTEMENT
```

Exemple2 : Requête R2\_2 «Nombre d'appartements de type T2 en location».

```
SELECT COUNT(*) as 'Nombre de T2'
FROM APPARTEMENT
WHERE TypeAppart = 'T2'
```

Pour compter le nombre de valeurs distinctes prises par une colonne, il faut indiquer l'argument DISTINCT suivi de l'argument considéré.

Exemple3 : Requête R2\_3 «Nombre de types différents d'appartements».

```
SELECT COUNT(DISTINCT TypeAppart) AS 'Nombre de types d appartements'
FROM APPARTEMENT
```

**Exemple4 :** Requête R2\_4 «Prix minimum, moyen et maximum des locations en haute-saison». (CodeSaison : 'H ' dans la table Saison)

```
SELECT MIN(`Prix/Semaine`) as Minimum,  
        AVG(`Prix/Semaine`) as Moyen,  
        MAX(`Prix/Semaine`) as Maximum  
FROM tarification  
WHERE CodeSaison = 'H'
```

29

## SOMMAIRE

1-Interrogation portant sur une seule table

2-Fonctions agrégats

**3-Interrogation simple portant sur plusieurs tables**

**3.1- Equi-jointure**

3.2- Alias du FROM

3.3- Jointure réflexive

3.4- Requête imbriquée


4-Regroupement

5-Les conditions sur regroupement

30

Dès lors que l'on souhaite obtenir des informations provenant de plusieurs tables, il va être nécessaire d'établir les liens existants (champs en commun) entre ces tables.

=> jointures

 Syntaxe générale d'une requête d'interrogation portant sur plusieurs tables:

```
SELECT champ1 [, champ2 ...]  
FROM table1  
JOIN table2 [,...]  
ON ....  
[WHERE condition1 [ OR/AND condition2...]] ]  
[ORDER BY champ1 [sens] [, champ2 [sens]...] ] ;
```

*Remarques : les instructions notées entre [ ] sont facultatives.*

*Les points de suspension indique une répétition possible n fois.*

31

### 3.1- Equi-jointure

Une jointure naturelle ou équi-jointure **rapproche** deux tables grâce à **deux champs** ayant un **même sens** : il s'agit généralement de lier la clé primaire d'une table avec la clé étrangère qui y fait référence dans une autre table.

Cette opération associe, à chaque tuple de la première table, tous ceux de la seconde qui satisfont le **critère de jointure**.

Cette jointure ne sera possible que si les deux champs appartiennent au *même domaine de valeurs* (type de données et taille).

32



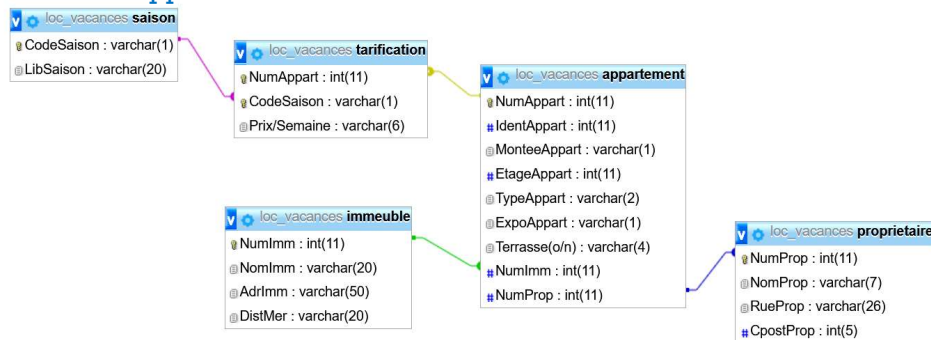
**Exemple1 : Requête R3.1\_1** «Liste des appartements (IdentAppart, TypeAppart, Terrasse(o/n),ExpoAppart) triés par nom d'immeuble ».

```
SELECT NomImm, IdentAppart, TypeAppart, `Terrasse(o/n)`, ExpoAppart
FROM APPARTEMENT
JOIN IMMEUBLE
ON IMMEUBLE.NumImm = APPARTEMENT.NumImm
ORDER BY NomImm
```

Conjonction d'égalité

**Remarque :** Lorsqu'un attribut est présent dans 2 tables avec le même nom, il doit être précédé du nom de sa table d'origine.

Ex : Appartement.NumImm ou Immeuble.NumImm



**Exemple2 : Requête R3.1\_2** «Liste des appartements (IdentAppart) de type T2 triés par nom d'immeuble».

```
SELECT NomImm, IdentAppart
FROM APPARTEMENT
JOIN IMMEUBLE
ON IMMEUBLE.NumImm = APPARTEMENT.NumImm
WHERE TypeAppart='T2'
ORDER BY NomImm
```

Exemple3 : Requête R3.1\_3 «Liste des appartements (IdentAppart) de type T2 ou plus, triés par nom d'immeuble (NumImm et Nom car des homonymes sont possibles dans les noms d'immeubles)».

```
SELECT APPARTEMENT.NumImm, NomImm, IdentAppart
FROM APPARTEMENT
JOIN IMMEUBLE
ON IMMEUBLE.NumImm=APPARTEMENT.NumImm
WHERE TypeAppart<>'T1'
ORDER BY APPARTEMENT.NumImm, NomImm
```

**Remarque** : On pourrait utiliser aussi TypeAppart IN ("T1", "T2", ...) ou (TypeAppart="T1" OR TypeAppart="T2"..... )

35

Exemple4 : Requête R3.1\_4 «Nom et adresse des immeubles d'Arcachon ou d'Anglet dans lesquels M. Dupouy a des appartements».

```
SELECT NomImm, AdrImm
FROM ( PROPRIETAIRE
JOIN APPARTEMENT
ON PROPRIETAIRE.NumProp=APPARTEMENT.NumProp)
JOIN IMMEUBLE
ON IMMEUBLE.NumImm=APPARTEMENT.NumImm
WHERE NomProp ='Dupouy'
AND AdrImm LIKE '%Arcachon%' OR AdrImm LIKE '%Anglet%'
```

36

### 3.2 - Alias du FROM (pour le traitement)

Il est possible de renommer, **temporairement pour une requête**, une ou plusieurs table(s) utilisées grâce à la clause **AS**.

La table renommée (associée à un alias) **ne pourra plus être désignée par son vrai nom dans la requête** concernée par le renommage.

Il faudra encadrer le nom spécifié par des `...` ou des [...] (selon le SGBD utilisé) si celui-ci est composé de plusieurs mots.

37

**Exemple : Requête R3.2** «Nom des propriétaires, des appartements de l'immeuble "Arlas" (numéro des appartements, numéro et nom d'immeuble) triés par nom de propriétaire».

<i>Requête sans alias</i>	<i>Requête avec alias</i>
<b>SELECT</b> NomProp, NumAppart, Immeuble.NumImm, NomImm <b>FROM</b> ( PROPRIETAIRE <b>JOIN</b> APPARTEMENT <b>ON</b> PROPRIETAIRE.NumProp= APPARTEMENT.NumProp) <b>JOIN</b> IMMEUBLE <b>ON</b> IMMEUBLE.NumImm= APPARTEMENT.NumImm <b>WHERE</b> NomImm = 'Arlas' <b>ORDER BY</b> NomProp	<b>SELECT</b> NomProp, NumAppart, I.NumImm, NomImm <b>FROM</b> ( PROPRIETAIRE <b>AS</b> P <b>JOIN</b> APPARTEMENT <b>AS</b> A <b>ON</b> P.NumProp= A.NumProp) <b>JOIN</b> IMMEUBLE <b>AS</b> I <b>ON</b> I.NumImm= A.NumImm <b>WHERE</b> NomImm = 'Arlas' <b>ORDER BY</b> NomProp

Remarques : La projection de APPARTEMENT.NumImm donnera le même résultat que celle de IMMEUBLE.NumImm. Ces champs, même s'ils renvoient les mêmes résultats ne sont pas identiques pour l'interpréteur SQL. Lorsque l'on utilise des attributs homonymes dans une requête, il convient donc de les préfixer avec le nom de leur table d'origine (ou son alias).

### 3.3 - Jointure réflexive (d'une table avec elle-même)

Il est nécessaire de faire une jointure réflexive si l'on souhaite rassembler des informations venant d'une ligne d'une table avec des informations venant d'une autre ligne de la même table.

Partons d'un schéma relationnel un peu plus complet de la base de données de LOC-VACANCES gérant également les locataires des appartements (uniquement pour des locations supérieures à trois semaines).

**PERSONNE** (NumPers, NomPers, RuePers, CpostPers, VillePers)

Clé primaire : NumPers

**APPARTEMENT** (NumAppart, IdentAppart, MontéeAppart, EtageAppart, TypeAppart, ExpoAppart, Terrasse(O/N), NumProp#, NumImm#, NumLocataire#)

Clé primaire : NumAppart

Clé étrangère : NumImm **de la relation APPARTEMENT en référence à NumImm de la relation IMMEUBLE**

Clé étrangère : NumProp **de la relation APPARTEMENT en référence à NumPers de la relation PERSONNE**

Clé étrangère : NumLocataire **de la relation APPARTEMENT en référence à NumPers de la relation PERSONNE**

39

Exemple : Requête R3.3 «Nom du propriétaire et du locataire de chaque appartement (NumAppart)».

**SELECT** NumAppart, PROP.NomPers, LOC.NomPers

**FROM** PERSONNE PROP

**JOIN** APPARTEMENT

**ON** PROP.NumPers = APPARTEMENT.NumProp

**JOIN** PERSONNE LOC

**ON** LOC.NumPers = APPARTEMENT.NumLocataire;

Remarques :

Le nom d'alias peut être composé de plusieurs lettres.

Le mot clé « AS » est facultatif.

40

### 3.4 - Requête imbriquée

Le langage SQL permet de faire appel, dans la clause WHERE, au résultat d'une autre requête d'interrogation. L'opérateur permettant d'exprimer le critère reliant la sous-requête à la requête principale sera différent selon que le résultat de la sous-requête comporte une ou plusieurs lignes.

#### 3.4.1 Sous-requête à résultat unique

Exemple : Requête R3.4.1 «Appartements les moins chers en basse saison».

```
SELECT NumAppart
FROM TARIFICATION
WHERE CodeSaison = 'B'
      AND `Prix/Semaine` = ( SELECT MIN(`Prix/Semaine`)
                           FROM TARIFICATION
                           WHERE CodeSaison = 'B' )
```

Remarques :

- la requête imbriquée ne renvoyant qu'un seul résultat (le plus petit prix trouvé) on peut utiliser le signe « = » pour comparer les Prix/semaine.

Dans le cas contraire, il faudra remplacer le signe « = » par l'opérateur « IN » (voir exemple suivant).

#### 3.4.2 Sous-requête à résultats multiples

Exemple : Requête R3.4.2 «Locataires qui sont aussi propriétaires (d'appartements différents)».

```
SELECT NumLocataire
FROM APPARTEMENT
WHERE NumLocataire IN ( SELECT DISTINCT NumProp
                        FROM APPARTEMENT)
```

## SOMMAIRE

1-Interrogation portant sur une seule table

2-Fonctions agrégats

3-Interrogation simple portant sur plusieurs tables

**4-Regroupement**

5-Les conditions sur regroupement

43

Le regroupement permet de répartir les enregistrements d'une table en sous-ensembles dans le but en général de réaliser des calculs statistiques.

Il s'exprime en SQL par la clause **GROUP BY** suivie du nom des colonnes servant de critères de regroupement.

Une opération effectuée dans l'instruction **SELECT** avec une clause de regroupement pourra délivrer **plusieurs lignes** dans le résultat de la requête.

Exemple 1 : Requête R4\_1 «Nombre d'appartements de chaque immeuble (NumImm)».

```
SELECT NumImm , COUNT(NumAppart) AS Nbre_Appart  
FROM APPARTEMENT  
GROUP BY NumImm
```

44

Exemple2 : Requête R4\_2 «nombre d'appartements de chaque propriétaire dans chaque immeuble (NumProp, NumImm)».

```
SELECT NumProp, NumImm, COUNT(NumAppart) AS Nbre_Appart  
FROM APPARTEMENT  
GROUP BY NumProp, NumImm
```

Remarque : Tous les attributs de la clause **SELECT** (sauf ceux utilisés par les fonctions agrégats) devront être placés dans la clause **GROUP BY**. (L'inverse n'est pas vrai !)

45

## SOMMAIRE

- 1-Interrogation portant sur une seule table
- 2-Fonctions agrégats
- 3-Interrogation simple portant sur plusieurs tables
- 4-Regroupement
- 5-Les conditions sur regroupement

46

Lorsqu'une condition de restriction porte sur un ensemble de lignes obtenu par regroupement, celle-ci est exprimée avec la **clause HAVING**.

Cette clause **suit la clause GROUP BY**.

Elle ne doit pas être confondue avec la clause WHERE qui exprime une restriction portant sur chaque ligne d'une table.

Exemple1 : Requête R5\_1 «Numéro des immeubles contenant plus de deux appartements (avec le nombre d'appartements de chacun).

```
SELECT NumImm, COUNT(NumAppart) AS NbAppart
FROM APPARTEMENT
GROUP BY NumImm
HAVING COUNT(NumAppart)>2
```

Remarque : (rappel sur les alias). Il n'est pas possible de faire référence à l'alias NbAppart dans la requête (même dans le **HAVING**).



Exemple2 : Requête R5\_2 «Nombre d'appartements de type T2 par immeuble (NumImm, NomImm), uniquement si le nombre est compris entre un et trois».

```
SELECT APPARTEMENT.NumImm, NomImm, COUNT(NumAppart)
FROM APPARTEMENT
JOIN IMMEUBLE
ON IMMEUBLE.NumImm=APPARTEMENT.NumImm
WHERE TypeAppart='T2'
GROUP BY APPARTEMENT.NumImm, NomImm
HAVING COUNT(NumAppart) BETWEEN 1 AND 3
```

**Ou**

```
SELECT APPARTEMENT.NumImm, NomImm, COUNT(NumAppart)
FROM APPARTEMENT
JOIN IMMEUBLE
ON IMMEUBLE.NumImm=APPARTEMENT.NumImm
WHERE TypeAppart='T2'
GROUP BY APPARTEMENT.NumImm, NomImm
HAVING COUNT(NumAppart) <= 3
```

