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).

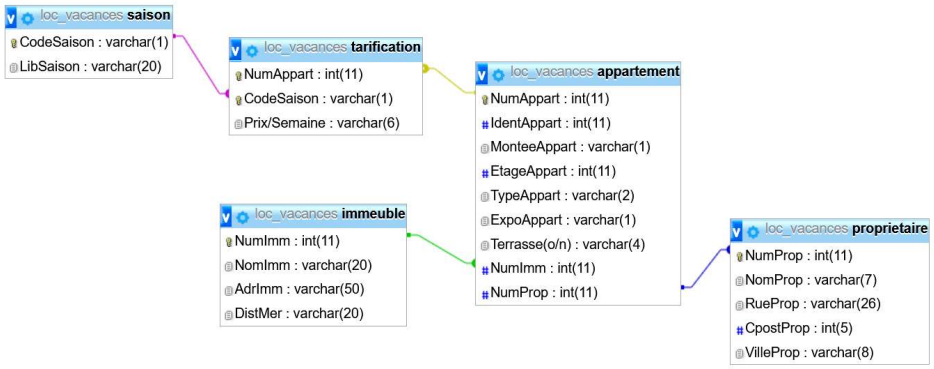
2

- 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 representée 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, AdrImm, DistMer)
APPARTEMENT(NumAppart, IdentAppart,MonteeAppart,EtageAppart,TypaAppart,ExpoAppart,Terrassa(o/n),NumImm#, NumProp#)
TARIFICATION(NumAppart#, CodeSaison#, Prix/Semaine)

4

SOMMAIRE

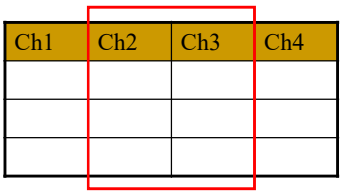
📖 Syntaxe générale d'une requête d'interrogation simple sur une table :

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 indique une répétition possible n fois.

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).



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
SELECT * FROM APPARTEMENT	SELECT : 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)».

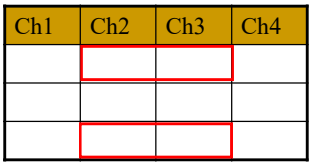
REQUETE SQL	EXPLICATIONS
SELECT TypeAppart, IdentAppart FROM APPARTEMENT	SELECT : opérateur de projection Il est suivi de la liste des attributs à projeter séparés par une virgule

Exemple3 : **Requête R1.1_3** «Liste des différents types d'appartements en location chez LOC_VACANCES».

REQUETE SQL	EXPLICATIONS
SELECT DISTINCT TypeAppart FROM APPARTEMENT	SELECT : opérateur de projection Il est suivi de la clause DISTINCT 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 DISTINCT 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.**



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

Exemple1 : Requête R1.2_1 « Appartements situés au rez-de-chaussée ».

REQUETE SQL	EXPLICATIONS
SELECT * FROM APPARTEMENT WHERE EtageAppart = 0	•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 1er 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
SELECT * FROM APPARTEMENT WHERE NOT ExpoAppart = 'N' Ou WHERE ExpoAppart <> 'N'	SELECT * FROM APPARTEMENT WHERE ExpoAppart IN ('S', 'E', 'O')

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
SELECT NomProp FROM PROPRIETAIRE WHERE CpostProp LIKE '64%'	SELECT NomProp FROM PROPRIETAIRE WHERE CpostProp>=64000 AND CpostProp < 65000 ou SELECT NomProp FROM PROPRIETAIRE WHERE CpostProp BETWEEN 64000 AND 64999

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».

```
SELECT ExpoAppart, IdentAppart
FROM APPARTEMENT
ORDER BY ExpoAppart
```

```
SELECT ExpoAppart, IdentAppart
FROM APPARTEMENT
ORDER BY 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 4ème»

```
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'en-tête 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

```
SELECT NumAppart, `Prix/Semaine` AS 'Prix haute saison'
FROM TARIFICATION
WHERE CodeSaison= 'H'
```

RESULTAT

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
SELECT NumAppart, `Prix/Semaine`*2 AS `(Px hte *2)<950€` FROM TARIFICATION WHERE CodeSaison= 'H' AND (`Prix/Semaine`*2) <950	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

```
SELECT NumAppart,CodeSaison,
    ROUND(`Prix/Semaine`*2) AS 'Px arrondi'
FROM tarification ;
```

RESULTAT

NumAppart	CodeSaison	Px arrondi
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 1er 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 1er 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 …

SELECT …
FROM table1 CROSS JOIN table2 […]
WHERE …

- Exemple :

SELECT *
FROM saison, appartement

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

25

SOMMAIRE

26

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

27

Exemple1 : Requête R2_1 «Nombre d'appartements en location».

```
SELECT COUNT(*)  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
```

28

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