PETIT MEMENTO SQL

Bloc SQL	• Un bloc SQL est composé de : SELECT .. FROM .. {WHERE …} {GROUP BY {HAVING}} {ORDER BY} dans cet ordre. • Les {} indiquent que le contenu est optionnel mais l'ordre est important • Dans le SELECT et dans le FROM les éléments sont séparés par des virgules (il n'y a pas de parenthèses). *Exemple : SELECT nom, prenom FROM vol, pilote* • Les parenthèses ne se mettent que s'il peut y avoir ambiguïté dans le WHERE par exemple avec des AND ou des OR qui se suivent. *Exemple : where age=15 AND nom='dupond' AND (ville = 'Paris OR ville ='Lyon')* • Le ; à la fin n'est pas obligatoire. Il est simplement utilisé par le système de gestion de données pour lui indiquer d'exécuter la requête (cf. TP). • Il ne peut pas y avoir de HAVING sans un GROUP BY
Lire une requête	• Il est important de bien lire la requête afin de déterminer en premier quelles sont les relations mises en jeu, ce qui est attendu dans le résultat puis les conditions. • Le fait de déterminer les relations permet d'écrire déjà une partie de la requête (le FROM) et surtout de ne pas oublier les jointures ! • Une bonne lecture de la requête permet d'en remplir la plus grande partie : le FROM et le SELECT. Après il suffit de se poser la question pour les conditions est-ce qu'elles doivent être dans le WHERE ou dans un HAVING ? Exemple : *« donner pour chaque acteur français (nom, prenom) le nombre de films qu'il a joué ». J'ai besoin d'ACTEUR pour connaître le nom et prénom. J'ai besoin de JOUER pour connaître le nombre de films. Je peux déjà écrire :* *SELECT nom, prenom* *FROM ACTEUR, JOUER* *WHERE idA=IdActeur* *Quelles sont les conditions ? nationalité = 'Français' -> je l'ajoute dans le WHERE* *SELECT nom, prenom* *FROM ACTEUR, JOUER* *WHERE idA=IdActeur*

	AND nationalité='France' *Là j'ai le nom et prenom de tous les acteurs de nationalité Française. Je veux dans le résultat le nombre de films qu'il a joué, je l'ajoute dans le SELECT.* *SELECT nom, prenom, count(*)* *FROM ACTEUR, JOUER* *WHERE idA=IdActeur* *AND nationalité='France'* Etant donné que je ne peux pas avoir une fonction d'agrégation avec un attribut dans un SELECT sans GROUP BY : *SELECT nom, prenom, count(*)* *FROM ACTEUR, JOUER* *WHERE idA=IdActeur* *AND nationalité='France* *GROUB BY idA, nom, prenom* *(ne pas oublier de mettre l'idA à cause des homonymes et TOUS les attributs qui sont dans le SELECT DOIVENT OBLIGATOIREMENT ETRE dans le GROUP BY)*
Faire un petit exemple	• Il est conseillé de toujours faire un petit exemple de tuples pour voir ce qui est mis en jeu. Cela permet de mieux comprendre si une condition s'applique dans un WHERE ou dans un HAVING, s'il peut y avoir plusieurs fois la même ligne dans le résultat (DISTINCT). • L'exemple n'a pas besoin d'être compliqué. Exemple : *les 4 lignes ci-dessous peuvent correspondre à 100 c'est un numéro de film ou d'acteur, la dernière colonne un nbspectateurs ou un salaire. L'idée est de voir un peu ce qui se passe derrière et d'aider à trouver comment résoudre la requête.* *100 1 30* *100 2 40* *101 4 50* *100 3 60*
Penser à faire une simulation	• Lorsque vous êtes dans un WHERE ou juste après le FROM, il faut imaginer que le système de gestion de base de données va analyser ligne par ligne comme dans une boucle FOR. Imaginez donc que vous avez une boucle FOR qui passe l'ensemble des enregistrement ligne par ligne. • Sur l'exemple : 100 1 30 100 2 40

	101 4 50 100 3 60 Cela veut dire que l'on regarde d'abord la ligne : 100 1 30 Où on peut par exemple chercher une condition sur la valeur d'un attribut (salaire=30 par exemple) Puis la ligne 100 2 40 • Vous pouvez appliquer le même principe pour un GROUP BY. Lorsqu'il y a un GROUP BY il faut imaginer une boucle FOR qui parcours partition par partition.
Les jointures	• Lorsqu'il y a n relations il y a n-1 jointures. Elles s'écrivent tout de suite dans le WHERE pour ne pas les oublier. Exemple : *SELECT * FROM R1, R2 WHERE R1.id=R2.id.* Autrement il s'agit du produit cartésien ! La notation JOIN peut bien sûr être utilisée.

Les fonctions d'agrégations	• Il existe différentes fonctions d'agrégation : COUNT, SUM, MIN, MAX, AVG, …
	• Elles ne peuvent se situer que dans :
	  ○ un SELECT. Exemple : *SELECT COUNT (*)*
	  ○ à gauche d'un opérateur dans le HAVING. Exemple : *HAVING COUNT (*) =* où '=' est l'opérateur
	• Ne jamais utiliser une fonction d'agrégation dans un WHERE à gauche d'un opérateur. Exemple : *WHERE COUNT(*) =* est faux !
	• Je ne peux pas avoir un attribut dans un select avec une fonction d'agrégation si je n'ai pas de GROUP BY. Exemple : *SELECT nom, COUNT(*) FROM R* est faux ! il faut *SELECT nom, COUNT(*) FROM R GROUP BY idR, nom*

Contenu du GROUP BY	• Tous les attributs qui sont dans le SELECT sont dans le GROUP BY. *Exemple : SELECT nom, prenom FROM R GROUP BY nom, prenom* • L'inverse n'est pas vrai : je peux avoir des attributs qui sont dans le GROUP *BY et qui ne sont pas dans le SELECT. Exemple : SELECT COUNT(*) FROM R Group by idR ;* • Attention aux homonymes : il peut y avoir des homonymes et si la clé de la relation n'est pas dans le SELECT il faut penser à partitionner en fonction de la clé. Exemple : *SELECT nom, prenom FROM R GROUP BY idR, nom, prenom*. • Remarque 1 : si la clé est dans le SELECT de toute façon elle va se retrouver dans le GROUP BY. Exemple : *SELECT idR, nom, prenom FROM R GROUP BY idR, nom, prenom*. • Remarque 2 : il est conseillé de toujours mettre la clé en premier avant les attributs pour partitionner en fonction de la clé et être sûr de ne pas l'oublier. Exemple : *GROUP BY idR, nom, prenom* plutôt que *GROUP BY nom, prenom, idR*.

	• **Faut-il toujours la clé primaire dans le GROUP BY ?**. Exemple : *« donner par année, le nombre de films ». La réponse est : SELECT annee, count(*) FROM FILM GROUP BY annee ; ici il n'y a pas de idF dans le GROUBY pourquoi ? si on met l'idF dans le GROUP BY on va créer autant de ligne qu'il existe de film et après pour chaque ligne on va partitionner par année. Ce n'est pas ce qui est demandé. On veut simplement partitionner par rapport aux années.* Conclusion : il faut toujours bien regarder ce qui est attendu et ce qui se passe et ne pas automatiquement mettre la clé. Dans l'exemple, il n'y a pas de problème d'homonymie.
Contenu du HAVING	• Attention dans un HAVING il faut obligatoirement une fonction d'agrégation à gauche d'un opérateur. Exemple : *HAVING COUNT(*) <* …..où '<' est un opérateur. HAVING avec un attribut est faux. Exemple : *HAVING nbspectateurs <* …est faux ! la condition doit être réalisée dans le WHERE.
GROUP BY VS. DISTINCT	• Généralement on utilise DISTINCT pour supprimer les doublons dans le résultat. • Il est parfois préférable d'utiliser un GROUP BY plutôt qu'un DISTINCT notamment lorsqu'il y a une clé primaire en jeu. Exemple : *Considérons les tuples suivants* *100 1 30 DUPOND* *100 2 20 DUPOND* *101 1 10 DURAND* *100 3 70 DUPOND* *102 1 50 DUPOND* Où la première colonne est par exemple la clé primaire d'une personne (DURAND a le numéro 101). Si je cherche les enregistrements dont la valeur du troisième attribut est supérieure à 40, je vais trouver les deux derniers : 100 et 102. Si dans mon SELECT, j'ai : SELECT nom, je vais avoir : *DUPOND* *DUPOND* Si je transforme mon SELECT en : SELECT DISTINCT nom Je vais obtenir simplement : *DUPOND* *Pour avoir les deux, je suis obligé de faire un GROUP BY :* *SELECT nom FROM R GROUP BY idR, nom* • Attention cela ne veut pas dire qu'il faut toujours utiliser un GROUP BY. Exemple : *SELECT DISTINCT nationalité FROM acteur retourne bien les différentes nationalités des acteurs.*
Représentation du MAX	• Il y a des requêtes qui nécessitent de comparer un résultat cumulé (e.g. COUNT) et de voir si la valeur retournée est bien la plus grande (resp. la plus petite) dans un HAVING. Pour faire un max (resp. un min):

	HAVING fonction() **>= ALL** (select fonction () from R **GROUP BY** idR) où fonction est une fonction d'agrégation. Pour le min : il faut mettre <=. • Attention HAVING fonction() **= MAX** (select fonction () From R **GROUP BY** idR) est faux et ne fonctionne que sur MySQL.
Requêtes imbriquées	• Les requêtes imbriquées sont souvent utilisées pour faire des comparaisons avec « soi-même ». Le principe est que la sous requête donne un résultat s'il est possible de faire une jointure avec la requête principale. • Considérons notre exemple : 100 1 30 100 2 40 101 4 50 100 3 60 • Je veux connaître, pour une valeur de la première colonne, quelle est la plus grande valeur de la dernière colonne. C'est à dire que pour 100 je veux retourner 60 et pour 101 je veux 50. • La requête s'écrit de la manière suivante : SELECT id FROM R R1 WHERE val_col3= (SELECT MAX(R2.val_col3) FROM R R2 WHERE R1.id=R2.id) La première partie de la requête : « SELECT id FROM R R1 WHERE val_col3 = » permet de parcourir, comme une boucle FOR, chaque enregistrement de la relation. Donc on commence ici avec la ligne 100 1 30. R1.Id=100 et R1.val_col3 = 30. On va donc comparer 30 avec la sous requête. La sous requête : SELECT MAX(R2.val_col3) FROM R R2 WHERE R1.id=R2.id va rechercher un MAX mais par contre elle va aussi rechercher dans R2 tous les identifiants qui sont similaires à R1.id (WHERE R1.id=R2.id). Dans notre cas nous avons R1.id=100, donc on va regarder dans R2 tous les enregistrements ayant 100, on trouve donc : 100 1 30 100 2 40 100 3 60 On prend le max de la dernière colonne et on a : 60. Donc compare le R1.val_col3=30 de la première partie avec le 60 et on voit que ce n'est pas égal donc la plus grande valeur pour 100 n'est pas la première ligne. Idem pour la ligne 2. Imaginons qu'on arrive à la 4ième ligne, on a donc R1.id=100, R1.val_col3=60 et en regardant la sous requête on a (WHERE R1.id=R2.id), les mêmes enregistrements que précédemment avec le max vaut 60. Donc on voit bien que 60=60 il s'agit pour le R1.id de la plus grande valeur.

	Pour la troisième ligne (R1.id=101, R1.val_col3=50), la condition de la sous requête (WHERE R1.id=R2.id) donne une seule ligne : 101 4 50 Donc le max est 50 et on voit bien qu'il s'agit de la plus grande valeur pour R1.id=101.