LE LANGAGE SQL AVANCE

SOMMAIRE

LE LANGAGE SQL AVANCE

L'objectif de cette deuxième partie du cours d'Architecture Logicielle sur le langage SQL est d'approfondir l'étude du langage SQL dans ses 3 fonctions :
- Interrogation et modification d'une base de données relationnelle
- Définition et modification du schéma d'une base de données relationnelle
- Contrôle de sécurité et d'intégrité de la base.

NB : Les exemples donnés dans ce cours ne sont pas exhaustifs (Se reporter aux TP pour un complément d'information)

A. Langage de manipulation de données

A.1) Les sous-interrogations

Les sous interrogations permettent l'expression d'un prédicat à l'aide du résultat d'un « SELECT ».

La syntaxe générale est la suivante :

```
SELECT  Colonnes
FROM  Tables
WHERE  Colonne   Opérateur   (SELECT  Colonne
                              FROM  Table
                              WHERE  Condition) ;
```

A.1.1) Sous interrogations ramenant une ligne

Exemple : Considérons la table Commande suivante

CommandeClient
Numcommande
DateCommande
DateLivraisonsouhaitée

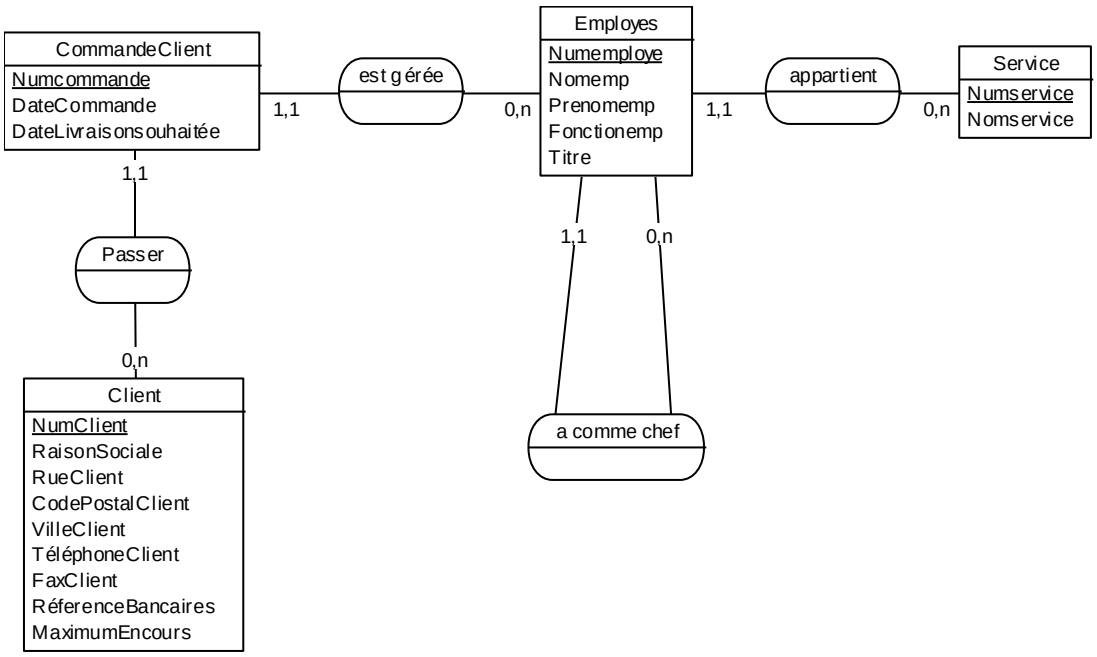
Donnez la requête SQL permettant d'afficher la liste des commandes passées après la commande numéro 9720018.

Décomposition de la sous – interrogation :
1. La sous interrogation est indépendante de l'interrogation principale
2. Le noyau évalue la sous – interrogation **puis** l'interrogation principale avec le résultat trouvé

A.1.2) Sous interrogations ramenant plusieurs lignes

Dans ce cas, il faut utiliser les opérateurs permettant de comparer une valeur à une liste de valeurs :
- l'opérateur **IN**
- un opérateur de comparaison (=, <>, >, >=, <, <=) suivi de **ANY** ou **ALL**

Exemple : Considérons le MCD suivant



[Pour rappel :
- Schéma relationnel de la table Employes :
- Ordres SQL de création de la table Employes :

]
- Donnez la liste des clients ayant passée une commande gérée par l'employé numéro 15

- Donnez le nom du service ayant le plus grand nombre d'employés :

A.1.3) Sous - interrogation multiple

La clause WHERE d'une requête principale peut contenir plusieurs sous – requêtes reliées par les connecteurs AND et OR.

A.1.4) Sous - interrogation synchronisée

La sous – interrogation peut faire référence à une colonne de l'interrogation principale. Ceci nécessite la réévaluation de la sous - interrogation pour chaque ligne principale. Dans certains cas, il est obligatoire d'utiliser les alias pour renommer une des tables.

Exemple : On reprend le MCD précédent
Donnez la liste des employés ne travaillant pas dans le même service que **leur** chef.

La sous – interrogation est réévaluée pour chaque nouveau A.Numchef. Il y a une dépendance entre la sous – interrogation et l'interrogation principale.

A.1.5) Sous – interrogation testant l'existence

Une des formes particulière de la sous requête synchronisée est celle testant l'existance de lignes de valeurs répondant à telle ou telle condition.

Le mot clé **EXISTS** est placé à cet effet devant une sous-requête.
Cet opérateur renvoie le résultat Vrai si la sous – requête renvoie au moins une ligne de valeurs, faux sinon.

L'interrogation principale s'exécute si Vrai.

L'opérateur EXISTS peut être nié par NOT : … **NOT EXISTS**…

Exemple :
Donnez la liste des commandes pour lesquelles les références de l'employé ont été mal saisies ou sont Null.
Rmq : Cette liste revient à tester la présence d'une clé étrangère dans la table Commandes sans clé primaire associé dans la table des Employés.

A.2) La division

L'opérateur relationnel de la division n'existe pas en SQL. En effet, la division algébrique concerne la prise en compte du quantificateur universel : « Quelque soit » qui n'existe pas en SQL. En SQL, seul existe le quantificateur existentiel : « Il existe ».

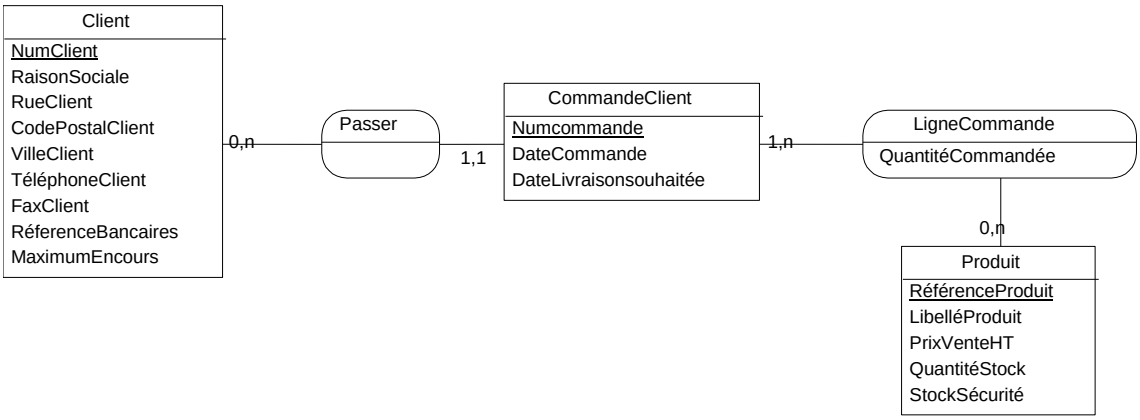
Rappel sur la division en algèbre relationnelle :
La division permet de chercher dans une relation les sous-tuples qui sont complétés par tous ceux d'une autre relation. Elle permet ainsi d'élaborer la réponse à des questions de la forme « quel que soit x, trouver y » de manière simple.

A.2.1) Double forme négative

Une première approche est d'utiliser la double forme négative pour exprimer la division.

Quel que soit x, P(x) est Vrai ⇔ Il n'existe pas x tel que P(x) est Faux.

Exemple : on considère le MCD suivant.



Sélectionner les clients (Numclient) ayant commandés **tous** les produits de l'entreprise.

A.2.2) Utilisation de la fonction de comptage COUNT

Une autre façon de traduire la division est d'utiliser la fonction de comptage COUNT.

1ère étape : On compte le nombre de produits de l'entreprise ;

2ème étape : Ensuite on compte le nombre de produits différents commandés par client.

3ème étape : On compare en final les deux nombres en imbriquant les deux requêtes. En cas d'égalité, le client a commandé tous les produits de l'entreprise.

A.3) Les jointures avancées

A.3.1) Généralités :

Syntaxe de jointure **SQL Server**:

SELECT nom_table.nom_colonne, nom_table.nom_colonne[,nom_table.nom_colonne…]
FROM {nom_table, nom_table}
WHERE nom_table.nom_colonne **opérateur jointure** nom_table.nom_colonne

Liste des opérateurs de jointures

Symbole	*Signification*
=	Egal
>	Supérieur à
<	Inférieur
>=	Supérieur ou égal
<=	Inférieur ou égal
<>	Différent de
=*	Jointure externe droite
*=	Jointure externe gauche

Types de jointures
Il existe trois types généraux de jointure :
- les jointures internes qui comprennent les jointures naturelles et les équijointures
- les jointures croisées (jointures non restreintes)
- les jointures externes qui comprennent les jointures gauche, droite et externes intégrales

Les jointures internes ont été abordées dans la première partie du cours SQL.

A.3.2) Jointures croisées et non restreintes
Une jointure croisée ou non restreinte produit un ensemble de résultats qui comprend toutes les combinaisons de toutes les lignes entre les tables de la jointure. C'est en fait le produit cartésien des tables jointes. Par exemple si une table présente 8 lignes, et une autre 9, le résultat comprend 72 lignes.

Exemple : Produit cartésien des champs nom_éditeur et titre des tables titres et éditeurs. (La table éditeurs comprend 9 lignes, la table titres 18. L'ensemble comprendra donc 162 lignes)

Remarque : rarement utilisé

A.3.3) La jointure externe

Dans le cas d'une jointure interne ou croisée , lorsqu'une ligne d'une table ne satisfait pas à la condition de jointure, cette ligne n'apparaît pas dans le résultat final de la requête.

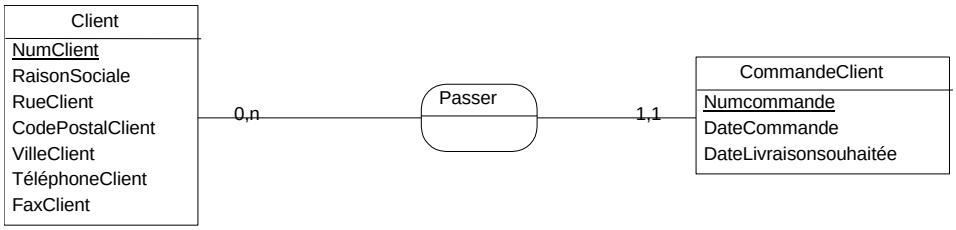
Il peut cependant être souhaitable de conserver les lignes d'une table qui ne répondent pas à la condition de jointure. On parle alors de semi-jointure ou de jointure externe.

On distingue 3 types de jointures externes :
Jointure externe gauche ou semi-jointure gauche : inclut toutes les lignes issues de la première table nommée
Jointure externe droite ou semi-jointure droite : inclut toutes les lignes issues de la deuxième table nommée
Jointure externe intégrale : inclut les lignes sans correspondance des tables droites et gauche

- jointures externes gauche et droite

Les jointures externes droite et gauche peuvent être créées en employant les opérateurs de jointure externe gauche « *= » et jointure externe droite « =* » dans la clause WHERE d'une instruction SELECT.

Exemple : on reprend l'exemple de la jointure entre une table client et une table CommandeClient



Si l'on considère la semi-jointure gauche, elle consiste dans l'exemple à ajouter au résultat de la jointure normale, l'ensemble des clients qui n'ont pas passé de commandes.

```
SELECT *
FROM Client, CommandeClient
WHERE Client.Numclient  *=  CommandeClient.Numclient
```

Si l'on considère la semi-jointure droite, il s'agirait d'afficher ses commandes possédant un numéro de client inexistant dans la table client.

```
SELECT *
FROM Client, CommandeClient
WHERE Client.Numclient   =*   CommandeClient.Numclient
```

- Jointures externes intégrales ou jointure généralisée (FULL OUTER JOIN)

Les jointures externes intégrales sont utilisées lorsque deux tables sont jointes et que chacune contient plusieurs lignes qui ne correspondent à aucune ligne dans l'autre table. Elle consiste à faire appraître les lignes des deux tables qui ne satisfont pas à la condition de jointure.

La réalisation de la jointure externe intégrale peut se faire en utilisant l'union de la semi- jointure droite et de la semi-jointure gauche.

```
SELECT *
FROM table1, table2
WHERE colonne1 * =  colonne2
UNION
SELECT *
FROM table1, table2
WHERE colonne1  =*   colonne2
```

Rmq : Suivant la base de données utilisée, d'autres syntaxes sont possibles pour les jointures externes.

A.3.4) Inéquijointure

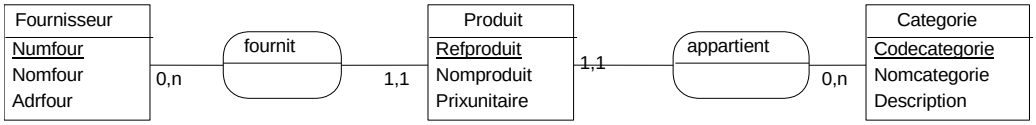
La mise en relation des colonnes communes à deux tables ne s'établit pas forcément par l'intermédaire d'une opération d'égalité. On parle alors d'inéquijointure. Les opérateurs possibles sont : >, <, <=, >=, < >.

A.3.5) L'autojointure ou jointure d'une table avec elle-même

Une autojointure relie les lignes d'une table avec d'autres lignes de la même table.
Pour joindre une table à elle même, vous devez lui affecter deux **alias** (abréviation de table) afin que la table puisse être référencée comme deux tables distinctes. Les alias sont affectés dans la clause FROM en spécifiant l'alias après le nom de table.

L'utilisation d'un alias permet de renommer une des tables et évite les problèmes d'ambiguïté pour les noms de colonnes qui doivent être préfixées par le synonyme des différentes tables.

Exemples : On considère le MCD suivant :



Exemple 1 : Liste des fournisseurs ayant un prix supérieur au produit de référence « AG153 ».

Exemple 2 : Afficher les couples de produits ayant le même code catégorie. (Il faut éliminer les couples de produits réflexifs et symétriques)

B . Langage de définition de données

B.1) Les étapes SQL

A partir d'un modèle relationnel en 3FN (troisième forme normale) et **après une éventuelle optimisation** de ce schéma, vous pouvez créer votre base de données à l'aide de votre SGBD relationnel incorporant généralement le langage SQL.

1. Création de(s) la base de données
2. Créer les tables et les index
3. Définir les autorisation d'accès à la base de données
4. Saisir les informations de la base de données
5. Mettre la base de données à la disposition des utilisateurs
6. Vérifier la protection des informations

B.2) Les vues

B.2. 1) Principe de la vue

Une vue constitue une manière différente de consulter des données issues d'une ou de plusieurs tables de la base de données.

L'utilisateur qui interroge une vue à l'illusion d'accéder à une (ou plusieurs) tables contenant réellement des données, alors qu'il exécute une instruction SQL faisant référence à une ou plusieurs tables réelles.

B. 2.2) Définition

Les vues sont « des fenêtres dynamiques» sur la base de données. Une vue est une «table virtuelle » construite à partir d'une ou plusieurs tables.

Les objectifs d'une vue sont:
- faciliter l'accès aux données,
- masquer des données ou limiter l'accès aux informations sensibles et confidentielles,
- contrôler l'intégrité des données,
- accéder aux données de façon dynamique (en fonction de l'environnement)

B.2.3) Utilité des vues

- Elles confèrent aux applications un certain degré d'indépendance face aux restructurations opérées au sein d'une base de données
- Elles accroissent la sécurité des données (protection des colonnes).
- Elles facilitent l'accès aux données et optimisent le partage des données entre les utilisateurs.
- Elles simplifient la manipulation des données:
- les requêtes les plus fréquemment utilisées sont définies comme des vues
- les données dont la consultation est superflue ou inutile n'apparaissent pas dans la vue

Dans la gestion des vues vous pouvez
- prévoir des vues différentes par utilisateurs,
- utiliser des colonnes de différentes tables,
- interroger une vue comme s'il s'agissait d'une table de la base,
- permettre aux utilisateurs d'effectuer des mises à jour, des créations et suppressions sur une base de données.

B.2.4) Création des vues

Syntaxe:
CREATE VIEW NomVue
[(aliascolonne1, aliascolonne2,...)]
AS SELECT
[WITH CHECK OPTION]

- Nom Vue: différent du nom des tables existantes
- Aliascolonnel : permet d'identifier les colonnes de la vue
- Instruction SELECT : ordre SELECT qui permet de créer la vue
- WITH CHECK OPTION : permet d'assurer la cohérence des informations modifiées

Remarques:
Les données insérées à travers une vue sont toujours stockées dans une table.
La mise à jour de données à travers la vue se fait en utilisant la condition de la clause WHERE.
Cette clause est indispensable en cas de mise à jour à partir de la vue et permet de contrôler en cas de modification (UPDATE) ou d'insertion (INSERT) que les nouvelles lignes vérifient la condition de La clause WHERE.

Exemple I:

- Créer la vue CommandesEmployesVendeur. Cette vue permet de visualiser toutes les informations des tables CommandesClient et Employes. Seuls seront retenus les enregistrement pour les quels l'employé concerné occupe la fonction de « Vendeur».

(cf MCD page 3)
CREATE VIEW CommandesEmployesVendeur AS

*SELECT * FROM CommandeClient, Employés*

WHERE CommandeClient. NumEmployé = Employés.NumEmployé

AND Fonctionemp LIKE "Vendeur"

- Interrogation à partir de la vue: Afficher les informations fournies par la vue

*SELECT * FROM CommandesEmployesVendeur*

Remarques:
Lors de l'interrogation à partir de la vue, l'ordre SQL de définition de la vue sera exécuté à chaque appel de la vue. On peut aller jusqu'à définir une vue par le biais d'une requête exploitant des vues et des tables ou mêmes uniquement des vues.

Il est parfois obligatoire de nommer explicitement les colonnes d'une vue c'est le cas lorsqu'une colonne de la vue contient le résultat d'une fonction agrégative ou d'une expression mathématique.

Exemple 2 :

- Créer la vue NBCommandesEmployes. Cette vue doit contenir les informations suivantes:
  - le numéro de l'employé,
  - son nom,
  - son prénom
  - le nombre de commandes qu'il gère (NbCommandes)

CREATE VIEW NBCommandesEmployes AS (Numéro, Nom, Prénom, Nbcommande)
AS
SELECT Numemploye, Nomemp, Prenomemp, COUNT(Numcommande)

FROM CommandeClient, Employés

WHERE CommandeClient. NumEmployé = Employés.NumEmployé

GROUP BY Numemploye, Nomemp, Prenomemp

- Interrogation à partir de la vue : Afficher la liste (numéro employé, nombre de commandes sous sa gestion)

SELECT numero, Nbcommande

FROM NBCommandesEmployes

B.2.5) Les mise à jour à travers une vue

Les opérations d'INSERT et UPDATE sont possibles mais doivent respecter les contraintes suivantes:
- le SELECT définissant la vue ne comporte pas de jointure.
- Toutes les colonnes de la tables définies en « Not nulI» se trouvent dans la vue
- Les colonnes résultats sont des colonnes réelles et non des résultats d'expression
- Les clauses GROUP BY et HAVING sont interdites dans le SELECT
- La clause WHERE ne peut contenir de sous requêtes corrélées.

NOTA BENE
- ne pas oublier CHECK OPTION pour contrôler l'intégrité des données
- la mise à jour à travers une vue est une opération délicate qu'il faut utiliser avec beaucoup de prudence.

B. 2.6) Création de vues dynamiques

Intérêt : construire des vues en fonction de données variables.

Exemple:
Construire une vue qui restitue les données relatives aux personnes travaillant dans le même département que l'utilisateur connecté

```
CREATE VIEW CommandesUtilisateurs AS
SELECT *
FROM CommandeClient WHERE Numemploye  =
        (SELECT Numemploye
        FROM Employes
        WHERE Nomemploye =USER);
```

B. 2.7) Suppression d' une vue

Syntaxe: DROP VIEW NomVue

Remarque: Vous ne pouvez supprimer que les vues que vous avez créées.