SQL : Structured Query Language

Introduction

SQL est un langage pour les BDR. Créé en 1970 par IBM.

Principales caractéristiques de SQL :

- **……** : SQL implémente le modèle relationnel.
- **……** : Du fait de cette normalisation, la plupart des éditeurs de SGBDR intègrent SQL à leurs produits (Oracle, Informix, Sybase, Ingres, MS SQL Server, DB2, etc.). Ainsi, les données, requêtes et applications sont assez facilement portables d'une base à une autre.
- **……..** : SQL est un langage de requêtes qui permet à l'utilisateur de demander un résultat sans se préoccuper des moyens techniques pour trouver ce résultat (assertionnel). C'est l'optimiseur du SGBD (composant du moteur) qui se charge de cette tâche.
- **……..** : SQL peut être utilisé à tous les niveaux dans la gestion d'une BDR :
  - Langage de Définition de Données LDD,
  - Langage de Manipulation de Données LMD,
  - Langage de Contrôle de Données LCD.

- Langage de définition de données LDD : permet la description de la structure de la base de données (tables, vues, attributs, index).
- Langage de manipulation de données LMD : permet la manipulation des tables et des vues avec les quatre commandes : SELECT, INSRET, DELETE, UPDATE.
- Langage de contrôle de données LCD : comprend les primitives de gestion des transactions : COMMIT, ROLLback et des privilèges d'accès aux données : GRANT et REVOKE.

SQL : Langage de Définition des Données (LDD)

INTRODUCTION

La définition de données dans SQL permet la définition des objets manipulés par le SGBD.

Les objets : table, vue, index

- Les commandes du LDD sont :
  a. **CREATE** : création des objets.
  b. **ALTER** : modification de la structure des objets.
  c. **DROP** : suppression des objets.

- **Création d'une table**
- **Vous devez avoir :**
  - **Le privilège CREATTE TABLE**
  - **Une zone de stockage**
- **Syntaxe de Création d'une table**

CREATE TABLE nom_table
(*colonne1 type* [DEFAULT expr][, …..]);

Vous devez spécifier :

- Un ***nom pour la table à créer***
- ***Pour chaque colonne de la table :***
  - Le nom de la colonne,
  - Son type
  - Sa taille
- L'option **DEFAULT** spécifie une valeur par défaut pour une colonne donnée dans un ordre insert

Définition des colonnes :

- La **taille** indique la valeur maximale de la longueur du champ.
- Les **types** de données les plus utilisés :

- **CHAR(n) :** chaîne de caractères de longueur fixe avec 1 <= n <= 2000. (Si on ne spécifie pas la longueur une colonne de ce type ne stocke qu'un seul caractère)

- **VARCHAR2(n) :** chaîne de caractères de longueur variable avec 1 <= n <= 4000.
- **NUMBER :** nombre entier ou décimal de 40 positions maximum de la forme :

Chiffre de 0 à 9 avec les signes « + » , « - » et le point décimal « . »

- **NUMBER(n) :** même chose que NUMBER avec une taille spécifique. La valeur maximale de n est 105.
- **NUMBER(n,m) :** nombre décimal pouvant avoir au maximum n chiffres dont m après la virgule.
- **DATE :** date dont le format standard est DD – MON - YYYY.

Exemple : Créer la table Produit ayant comme schéma :

Produit (Numprod, Desprod, Couleur, Poids, Qte_stk, Qte_seuil, Prix) Avec

- Numprod : de type numérique de taille 6,
- Desprod : de type caractère variable de taille 15,
- Couleur : de type caractère sur une position,
- Poids : de type numérique sur huit positions dont trois chiffre après la virgule,
- Qte_stk : de type numérique sur sept positions dont trois chiffres après la virgule,
- Qte_seuil : de type numérique sur sept positions dont trois chiffres après la virgule,
- Prix : de type numérique sur dix positions dont sept chiffres avant la virgule.

.....
.
.....
.
.....
.

.....
.
.....
.
.....
.
.....
.
.....
.
.....
.

Définition des contraintes d'intégrité

- Une contrainte d'intégrité est une
  - Propriété sémantique que doivent respecter les données afin d'assurer la cohérence de la BD.
  - Règle qui permet de contrôler au mieux la valeur d'une colonne
- **Les contraintes définies par SQL sont** :
- **NOT NULL** : **Non nullité des valeurs d'un attribut** : L'utilisateur est obligé de saisir la valeur de l'attribut.
- **UNIQUE : Unicité de la valeur d'un attribut ou d'un groupe d'attributs** : la valeur doit être unique.

■ **Exemple**

On suppose que deux produits différents ne peuvent pas avoir la même désignation et que le champ couleur est obligatoire. La commande de création da la table produit devient :

.....
.

.....
.
.....
.

- **Clé primaire (un attribut ou un groupe)** : indique que l'attribut est une clé primaire. Elle peut être définie comme contrainte de table ou comme contrainte de colonne.

■ Clé primaire comme contrainte de table selon la syntaxe :

CONSTRAINT nom_contrainte PRIMARY KEY(att1, att2,…, attn) ;

- Clé primaire comme contrainte de colonne : en ajoutant devant la colonne clé primaire : **Primary Key**.
  - **Remarque :** Dans le cas de clé primaire multiple, la clé primaire doit être créée comme contrainte de table (Syntaxe 1).

■ **Exemple 1 : Clé primaire comme contrainte de table**

pour Produit (Numprod, Desprod, Couleur, Poids, Qte_stk, Qte_seuil, Prix)

.....
.
.....
.
.....
.
.....
.
.....
.
.....
.
.....
.

.....
.
.....
.

- **Exemple 2 : Clé primaire comme contrainte de colonne**

.....
.
.....
.
.....
.
.....
.
.....
.
.....
.

Exemple 3 : Clé primaire multiple

LigneCommande (NumCde, NumProd, QteCde)

```
CREATE TABLE LigneCommande
        (Num_cde number(8),
         NumProd number(6),
         QteCde number(7,3),
```

.....
.
…….

Remarque

Dans le cas de clé primaire multiple, la clé primaire doit être créée comme contrainte de table.

1) **Clé étrangère (intégrité référentielle) :** lorsque la clé primaire figure dans une autre table en tant qu'un attribut. La clé étrangère peut être définie comme contrainte de table ou comme contrainte de colonne.

- Clé étrangère comme contrainte de table selon la syntaxe :

CONSTRAINT nom_contrainte FOREIGN KEY(nom_att) references nom_table(nom_att) [ON DELETE CASCADE]

- Clé étrangère comme contrainte de colonne : en ajoutant devant la colonne clé étrangère : **references nom_table(nom_att) [ON DELETE CASCADE]**
- **Remarque** :

Il est impossible de créer une clé étrangère si la clé primaire associée n'existe pas.

ON DELETE CASCADE : demande la suppression des lignes dépendantes dans la table en cours de définition dés que la ligne contenant la clé primaire correspondante dans la table maitre est supprimée.

Définition des contraintes

- Types de CI Oracle :
  - NULL / NOT NULL : niveau colonne
  - 1. UNIQUE (colonne1 [, colonne2] …)
  - 2. PRIMARY KEY (colonne1 [, colonne2] …)
  - 3. FOREIGN KEY (colonne1 [, colonne2] …) REFERENCES nomTablePere (colonne1 [, colonne2] …) [ON DELETE {CASCADE | SET NULL }]
  - 4. CHECK {condition}

Les 4 dernières CI sont définissables au niveau colonne ou au niveau table.

- Si on considère le schéma suivant :

MAGASIN(NumMag, Adresse, Surface)

PRODUIT(NumProd, DesProd, Couleur, Poids, Qte_Stk, CodMag #)

- La commande pour la création de la table Magasin étant :
- La commande pour la création de la table Produit peut être écrite de deux façons:

Solution 1: clé étrangère comme contrainte de table

CREATE TABLE Produit

(Numprod number(6) primary key,

Solution 2: clé étrangère comme contrainte de colonne

CREATE TABLE Produit

(Numprod number(6) primary key,

o Modification de la structure d'une table

Trois possibilités de modification de la structure de table :

…… des colonnes, ……. la structure d'une colonne ou ……. des colonnes existantes.

- **Ajout de nouvelles colonnes à une table**

Syntaxe :

ALTER TABLE nom_table
ADD (col1 type [(taille)] [null / not null] ,
col2 type [(taille)] [null / not null] ,
. . .
coln type [(taille)] [null / not null]) ;

Exemple :

Supposons qu'on veut ajouter une colonne type_clt à la table client :

ALTER TABLE CLIENT
ADD type_clt char(3) ;

- **Modification de la structure d'une colonne existante**

Syntaxe :

ALTER TABLE nom_table
MODIFY (col1 type [(taille)] [null / not null] ,
col2 type [(taille)] [null / not null] ,
. . .
coln type [(taille)] [null / not null]) ;

Remarque :

Pour modifier le nom d'une colonne :

ALTER TABLE nom_table
RENAME ancien_nom_col TO nouveau_nom_col ***;***

- Insertion à travers la copie des valeurs des colonnes d'une autre table

INSERT INTO nom_table [(les champs de la table)] Requête;

Exemple : INSERT INTO client_Sfax (num_clt, nom_clt, tel_clt) select * from client Where ville =‘Sfax';

1) **UPDATE** : **Modification de données**

UPDATE nom_table
SET col1 = val1 , … , coln = valn
WHERE condition ;

Remarque :

- Il n'est pas possible de mettre à jour plus qu'une table à la fois.
- La modification des données n’est pas autorisée que si les contraintes sont toujours valides.
- Les valeurs peuvent être des constantes, des expressions, des résultats de sous-requêtes ou NULL (pour supprimer la valeur initiale du champ).
- Si la clause Where n’apparaît pas dans la commande, il s’agit de mettre à jour tous les enregistrements de la table avec la même valeur.

Exemples

- Modifier la désignation du produit numéro 80 en Imprimante

;

- Majorer de 5% les prix des produits dont le prix est supérieur à 10.

- Modifier les quantités de tous les produits avec la valeur 10.

2) **Suppression de données**

DELETE FROM nom_table
[WHERE condition];

Exemple :

- Supprimer tous les Produits de couleurs Blanche
  ……. Where Couleur = 'B';
- Supprimer toutes les lignes de la table Produit
  DELETE FROM Produit;

III. Modification de la structure d'une table

Trois possibilités de modification de la structure de table :

…… des colonnes, …… la structure d'une colonne ou …… des colonnes existantes.

- **Ajout de nouvelles colonnes à une table**

ALTER TABLE nom_table
ADD (col1 type [(taille)] [null / not null] ,
col2 type [(taille)] [null / not null] ,
. . .
coln type [(taille)] [null / not null]) ;

Exemple: Supposons qu'on veut ajouter une colonne type_clt à la table client

- **Modification de la structure d'une colonne existante**

ALTER TABLE nom_table
MODIFY (col1 type [(taille)] [null / not null] ,
col2 type [(taille)] [null / not null] ,
. . .
coln type [(taille)] [null / not null]) ;

Remarque : Pour modifier le nom d'une colonne :

ALTER TABLE nom_table
RENAME ancien_nom_col TO nouveau_nom_col ***;***

- **Suppression de colonnes existantes**

ALTER TABLE nom_table
DROP (col1 , col2 ,…, coln) ;

Exemple : Supposons qu'on veut supprimer le champ ville de la table Magasin :

ALTER TABLE Magasin …….. ;

3) **Ajout d'une contrainte**

ALTER TABLE nom_table
ADD Constraint Def_de_contrainte ;

Exemple : Ajouter à la relation Magasin la contrainte suivante : la surface doit être comprise entre 10 et 100 m^2

ALTER TABLE Magasin .....

- **Suppression d'une contrainte clé primaire** :

ALTER TABLE nom_table DROP PRIMARY KEY [CASCADE] ;

Remarque : L'option cascade est ajoutée pour pouvoir supprimer une clé primaire référencée.

Exemple : Supprimer la contrainte clé primaire de la table magasin

ALTER TABLE …….. DROP PRIMARY KEY CASCADE ;

- **Suppression d'une contrainte autre que la clé primaire :**

ALTER TABLE nom_table DROP CONSTRAINT nom_contrainte ;

Exemple : Supprimer la contrainte spécifiant les couleurs possibles pour les produits **ALTER TABLE produit DROP CONSTRAINT Ck4_Produit ;**

Remarque :

Pour retrouver les différentes contraintes avec leur propriétés, on peut utiliser la commande suivante :

Select * from user_constraints
[where table_name = 'NOMTABLE'];

Il est à remarquer que pour cette commande, le nom de la table doit être écrit en **majuscule**.

2) **Suppression d'une table**

DROP TABLE nom_table ;

Exemple : Supposons qu'on veut supprimer la table client_tunis :

DROP TABLE client_tunis ;

3) Renommage et création de synonymes de tables

RENAME ancien_nom TO nouveau_nom ;

Il est également possible de donner à une même table plusieurs noms différents appelés synonymes :

CREATE SYNONYM nom_synonyme FOR nom_table ;

4) **Pour supprimer un synonyme** : ***DROP SYNONYM nom_synonym ;***

SQL : Langage de Manipulation des Données (LMD)

Les commandes du LMD sont :

1) ……. : ajoute des lignes à une table.
2) …….: mettre à jour les colonnes d'une table.
3) …….: suppression d'un ou de plusieurs enregistrements.
4) …….: définition de la liste des colonnes que l'on peut obtenir.

a. INSERT : ajouter des lignes à une table.

Les valeurs à ajouter doivent vérifier les contraintes définies au moment de la définition des données. Tout enregistrement ne vérifiant pas les contraintes sera rejeté.

Il existe deux formes d'insertions de données :

- Insertion des valeurs pour la totalité des valeurs de la table

INSERT INTO nom_table [(les champs de la table)] VALUES (liste des valeurs) ;

Les valeurs des colonnes de type caractère ou chaîne de caractère doivent apparaître entre deux cotes (' ').

Il est possible d'insérer un enregistrement en connaissant seulement les valeurs de quelques champs. Pour cela , il faut

- Soit indiquer la liste des champs à insérer et leurs valeurs respectives
- Soit mettre NULL dans la liste de valeurs pour les champs vides.

Les champs ayant été créés avec la contrainte Not Null devront, obligatoirement, avoir des valeurs.

- Insertion à travers la copie des valeurs des colonnes d'une autre table

INSERT INTO nom_table [(les champs de la table)] Requête;

Exemple : INSERT INTO client_Sfax (num_clt, nom_clt, tel_clt) select * from client Where ville ='Sfax';

b. UPDATE : Modification de données

UPDATE nom_table
SET col1 = val1 , … , coln = valn
WHERE condition ;

Remarque :

- Il n'est pas possible de mettre à jour plus qu'une table à la fois.
- La modification des données n'est pas autorisée que si les contraintes sont toujours valides.
- Les valeurs peuvent être des constantes, des expressions, des résultats de sous-requêtes ou NULL (pour supprimer la valeur initiale du champ).
- Si la clause Where n'apparaît pas dans la commande, il s'agit de mettre à jour tous les enregistrements de la table avec la même valeur.

Exemples

- Modifier la désignation du produit numéro 80 en Imprimante
- Majorer de 5% les prix des produits dont le prix est supérieur à 10.
- Modifier les quantités de tous les produits avec la valeur 10.

c. *DELETE* Suppression de données

DELETE *FROM nom_table*
[WHERE condition];

Exemples :

- Supprimer tous les Produits de couleurs Blanche

- Supprimer toutes les lignes de la table Produit

DELETE FROM Produit;

d. **SELECT Consultation de données**

SELECT col1, col2, … ,coln
FROM nom_table
[WHERE condition];

- **SELECT** : définir la liste des colonnes que l'on peut obtenir.
- **FROM** : noms des tables nécessaires pour obtenir le résultat souhaité.
- **WHERE** : la condition que doit vérifier un n-uplet donné pour qu'il fasse partie du résultat.

Remarques

- SELECT * pour avoir toutes les colonnes de la table.
- pour avoir toutes les lignes de la table, on ne met pas la clause WHERE.

Exemple

On considère la table

Produit (Numprod, Desprod, Couleur, Poids, Qte_stk, Qte_seuil, Prix)

1-Afficher les numéros et désignations de tous les produits

2-Afficher les numéros et désignations de produits existants en stock avec une quantité > 20

3-Afficher les produits existants en stock avec une quantité > 20

4-Afficher les couleurs des différents produits.

Notion d'alias

On peut modifier ces noms de colonnes, à l'affichage uniquement, en ajoutant des ALIAS. Si l'alias est composé de plusieurs mots il faut qu'il apparaisse entre cotes " ". La commande devient :

SELECT col1 [alias 1], col2 [alias 2], … ,coln [alias n]
FROM nom_table
[WHERE condition];

- Exemple : Afficher les numéros et désignations de tous les produits. Les titres des colonnes à afficher seront respectivement Numéro et Nom Produit

Select numprod Numéro, desprod "Nom produit" From Produit;

La condition est composée généralement de trois termes :

- un nom de colonne,
- un opérateur de comparaison,
- et une constante, une colonne, une liste de valeurs, une expression ou une requête.
- Opérateurs classiques de comparaison (= , <> , < , > , >= , <=)
- Opérateurs spécifiques dont principalement :
  - ***IS NULL***
  - ***IN*** *(liste de valeurs)*
  - ***BETWEEN V1 and V2***
  - ***LIKE*** *chaîne générique* :

    La chaîne générique est une chaîne de caractères qui contient l'un des caractères suivants :
    - ✓ % : remplace une autre chaîne de caractères qui peut être même une chaîne vide.
    - ✓ - : remplace un seul caractère.
- Si la condition est composée de sous conditions, on fait recours aux opérateurs logiques **AND**, **OR** et la négation **NOT** (NOT In, NOT Like, NOT Between, Is NOT LIKE)

Exemples :

1) Afficher les numéros de produits dont la couleur n'a pas été saisie.

2) Afficher les produits de couleur Rouge, Bleu ou Gris,

3) Afficher les numéros de produits dont le prix est compris entre 100 et 200,

4) Afficher les produits dont la désignation commence par 'o‘.

e. Afficher les numéros et désignations des produits dont les noms commencent par 'o' ou par 's'.

f. Afficher les désignations des produits contenant 'r' en deuxième position de la désignation et existant en stock avec une quantité > 20.

Remarque

Il existe une forme de création de tables accompagnée d'une insertion. Cette forme permet de créer une table et d'y insérer des données à partir d'une ou plusieurs tables ou vues.

Syntaxe : ***CREATE TABLE nom_table***
[(col1 type [(taille)] [null / not null] ,
col2 type [(taille)] [null / not null] ,
. . .
coln type [(taille)] [null / not null])]
AS requête ;

Dans les clauses SELECT et WHERE, on peut utiliser des expressions arithmétiques et des fonctions :

Les expressions arithmétiques

- ABS(n) : permet de calculer la valeur absolue de n.
- CEIL(n) : permet d'avoir le plus petit entier supérieur ou égal à n.
- Ceil(128.3) retourne 129 Ceil(128.8) retourne 129
- FLOOR(n) : permet d'avoir la partie entière de n.
- FLOOR(128.3) retourne 128 FLOOR(128.8) retourne 128
- MOD(m,n) : permet d'avoir le reste de la division entière de m par n.
- ROUND(m,n) : arrondit la valeur n à m décimal.
  Round(128.3) retourne 128 Round(128.8) retourne 129
- POWER(m,n) : permet d'avoir m puissance n
- SIGN(n) : donne –1 si n <0, donne 0 si n=0 et donne 1 si n>1.
- SQRT(n) : permet d'avoir √x.

Exemples

Afficher la désignation et les prix arrondis en Dinars de tous les produits

SELECT desprod, ROUND(prix) "Prix en D" FROM produit;

- Modifier la table produit de manière à majorer les prix en dinars.
  Update produit Set prix = ceil(prix);
- Supprimer tous les produits dont le prix appartient à [100,101]
  Delete from Produit Where floor(prix) = 100;

Les fonctions s'appliquant aux chaînes de caractère

- RTRIM (ch) : supprime l'espace à la fin de la chaîne ≠ LTRIM.
- RPAD (ch,n) : ajoute n espaces l'espace à la fin de la chaîne ≠ LPAD.
- INITCAP(ch) : met en majuscule la première lettre de chaque mot de la chaîne.
- INSTR(ch1,ch2,n,m) : donne la position de la m ième occurrence de ch2 dans ch1 à partir du caractère à la position n.
- LENGTH(ch) : renvoie la longueur d'une chaîne.
- LOWER(ch) : transforme la chaîne ch en minuscule ≠ UPPER.
- SUBSTR(ch,m,n) : permet d'extraire la sous-chaîne de ch qui commence à partir du caractère à la position m et de longueur n.
- TRANSLATE(ch,ch1,ch2) : permet de transformer dans la chaîne ch toutes les occurrences de ch1 par ch2.
- Replace(ch, ch1[,ch2]) : remplace une chaîne par une autre dans une colonne. Si on ne met pas ch2, ch1 va être remplacée par un vide.
- ch1||ch2 : concatène les deux chaînes.

Exemples

- Afficher les numéros des produits correspondants à des souris.
- Supprimer les produits dont la désignation est composée au maximum de 6 caractères.
- Afficher les désignations des produits avec la première lettre en majuscule
- Afficher les numéros et couleurs des produits dont la désignation contient "er" à partir de la 6ème position.

- Afficher toutes les désignations des produits en remplaçant toute "a" par "A"

  ……;
- Ajouter le champ Propriete de type caractère variable sur 4 positions formé des trois premières caractères de la désignation concaténés à la couleur.

  Alter table ……..
  Add ……. varchar(4);
  Update ……..
  Set propriete= substr(ltrim(……;

Les expressions s'appliquant à des dates

- ADD_MONTHS(d,n) : permet d'ajouter n mois à la date d sachant que n est un entier.
- GREATEST(d1,d2) : permet d'avoir la date la plus récente parmi d1 et d2 ≠ LEAST.
- MONTHS_BETWEEN(d1,d2) : permet d'avoir le nombre de mois qui se trouvent entre la date d1 et la date d2.
- LAST_DAY(d) : permet d'avoir la date du dernier jour de la date d.
- LAST_DAY('02/01/02') donne : 31/01/02
- SYSDATE : donne la date et l'heure système.

Exemples

- Soit la table Employé (Matricule, Nom, Prenom, Date_nais, Date_emb)

On suppose qu'un employé est à la retraite à l'âge de 60 ans, afficher les noms et prénoms des employés ainsi que leurs dates de retraite

Select nom, prenom, add_months(dateNais, 720) "retraite" From Employe;

- En supposant qu'un employé est à la retraite après 30 ans de service.

Afficher les noms et prénoms des employés ainsi que les dates prévus pour leurs retraites

Select nom, prenom, add_months(dateEmb, 360) "retraite" From Employe;

- Afficher le matricule et l'âge en mois de l'employé lors de son embauche

Select matricule, months_between(dateEmb, dateNais) "Age en Mois" From Employe;

- Afficher le matricule et l'âge en années de l'employé lors de son embauche

Select matricule, …….. "Age" From Employe;

- Afficher le matricule et le nombre de jours de travail pendant le premier mois d'embauche de chaque employé.

Select matricule, …….. "Nombre de Jours" From Employe;

Exemple 09/10/2006 → last_day(09/10/2006) → 30/10/2006
30/10/2006 - 09/10/2006 + 1 = 22 J

Les fonctions de conversion

- TO_CHAR(valeur-date, format-date) / TO_CHAR(nombre[,format]) : convertit une date ou une valeur numérique à une chaîne de caractères.
- TO_DATE(valeur-chaîne, format-date) : convertit une chaîne de caractères représentant une date à une date.
- TO_NUMBER(ch[,format]) : convertit une chaîne de caractères représentant un nombre en nombre.

Les expressions agrégats (ou fonctions de groupe)

ORACLE dispose d'une fonction appelée fonctions agrégats qui s'appliquent à un ensemble de données :

- AVG : permet d'avoir la moyenne arithmétique d'un ensemble donné.
- COUNT : permet d'avoir le nombre d'occurrences des enregistrements.
- MAX : permet d'avoir la valeur maximale dans une colonne.
- MIN : permet d'avoir la valeur minimale dans une colonne.
- SUM : permet d'avoir la somme des éléments.
- STDDEV : permet d'avoir l'écart type.
- VARIANCE : permet d'avoir la variance.

Chacune de ces fonctions a comme argument un nom de colonne ou une expression arithmétique. Elles ignorent les valeurs nulles et par défaut prennent les valeurs multiples pour des valeurs différentes.
Pour ne prendre que les valeurs distinctes, il faut ajouter l'opérateur DISTINCT. Et si l'on veut insister pour prendre toutes les valeurs qui existent, on doit précéder la colonne ou l'expression par l'opérateur ALL.

- La fonction COUNT peut prendre comme argument le caractère * pour connaître le nombre de lignes sélectionnées.
- Remarque

On ne peut pas mettre une fonction de groupe après la clause WHERE parce qu'elle s'agit d'une valeur inconnue.

Exemples

- Donner le nombre de produits de couleurs rouge
  SELECT Count(*) "Nombre" FROM Produit Where Upper(Couleur)='R';
- Afficher la quantité totale en stock des produits de couleur Rouge
  SELECT sum(qte_stk) "quantité" FROM Produit
  Where Upper (Couleur) = 'R';
- Afficher la quantité moyenne en stock des produits de couleur Rouge
  SELECT avg(qte_stk) "quantité" FROM Produit
  Where upper (Couleur) = 'R';
- Afficher les quantités minimales et maximale des produits en stock
  SELECT max(qte_stk) "maximum" , min(qte_stk) "minimum"
  FROM Produit;

Tri des résultats

Pour obtenir un résultat trié, il suffit d'ajouter à la requête SQL la clause :

…… expression [asc / desc]

Remarque :

Si on ne spécifie pas asc ou desc par défaut le tri est croissant (asc).

Exemple :

Donner la liste des produits ordonnés par ordre croissant de leurs prix.

SELECT * FROM produit ORDER BY prix;

Donner la liste des produits ordonnés par ordre croissant de leurs prix et décroissant de leurs désignations.

SELECT * FROM produit ORDER BY prix, desprod desc;

La jointure simple

Elle permet de …….. issues à partir de deux ou plusieurs tables en vue de retrouver des données associées.

Les colonnes utilisées pour faire la jointure doivent être de ……. et de …….. et sont appelés ……..

Pour effectuer une jointure, il faut spécifier :

Les noms des tables, dans la clause FROM sont séparés par des virgules.

La condition de jointure est dans la clause WHERE.

Les noms de colonnes doivent être préfixé par les noms de tables pour éviter toute ambiguïté.

La forme générale d'une requête de jointure est :

SELECT col1 , col2 , …. , coln
FROM table1 [variable 1] ,… . , tablen [variable n]
WHERE condition ;

Exemple : Magasin (Nummag, adresse, Surface)
Produit (Numprod, Desprod, Couleur, Poids, Qte_stk, Qte_seuil; Prix, Nummag#)

Afficher les numéros des produits ainsi que leur adresse de stockage

Afficher les adresses des magasins contenant le produit numéro 100

Afficher les surfaces des magasins des produits dont le poids est entre 100 et 500

L'autojointure

Elle consiste à faire une jointure d'une table avec elle-même dans le but de construire un n-uplet contenant des attributs en provenance de deux lignes, inter-reliées, cherchées dans la même table. Pour cela il faut créer un SYNONYME de la table ou utiliser des variables ALIAS.

Exemple

Trouver la désignation et le prix unitaire des produits dont le prix est supérieur à celui du produit 10

Solution avec synonyme

CREATE …….. prod FOR produit;
Select prod.desprod, prod.prix From ……, ……. Where …….numprod = 10 and ……prix > ……...prix;

Solution avec ALIAS

Select p1.desprod, p2.prix From produit p1, produit p2 Where p2.numprod=10 and p1.prix > p2.prix;