GT'M ingénierie

Professionnalisation des métiers informatiques

Langage SQL MYSQL

Développeur/**Administrateur de Bases de Données**

Immeuble International – 2 rue Stephenson 78180 MONTIGNY LE BRETONNEUX
Tél : 01 61 08 48 20 – Fax : 01 61 08 48 25
RC Versailles 40928926100037 – Code APE 8559 A – N° d'agrément 11788133178
gtm@gtm-ingenierie.fr – www.gtm-ingenierie.fr

Sommaire

1. INTRODUCTION

A l'origine, SQL ("Structured Query Language") a été développé pour permettre la manipulation et la gestion des bases de données relationnelles. C'est le Dr E.F Codd qui a mis en place le premier modèle dans les années 1970, pour gérer une base nommée SYSTEM R.

La version initiale de SQL fût décrite pour la première fois en 1976 dans le journal de la recherche IBM.

SQL est plus qu'un langage d'interrogation (DML), il permet aussi des opérations de mise à jour comme 'INSERT', 'DELETE', 'UPDATE', ainsi que des opérations de définition de droits et de données (DDL).

SQL est utilisé aujourd'hui dans de nombreux SGBD relationnels tels que SQL SERVER (MICROSOFT), DB2 (IBM), ORACLE (ORACLE Corporation), MYSQL(ORACLE Corporation)...

1.1. AMELIORATIONS RECENTES DE SQL

Le moteur SQL est le fondement de toutes les applications de base de données Oracle. Ce langage évolue en permanence pour répondre aux exigences croissantes des applications de base de données, aux nouvelles architectures informatiques, aux interfaces de programmation d'applications (API) et aux protocoles réseau.

En plus des données traditionnelles structurées comme les données de type caractère, numérique ou date, SQL est capable de stocker, récupérer, et traiter des données plus complexes comme les types : OBJET, XMLType, LOB (Long Binary Object) sachant qu'un seul LOB peut atteindre de 8 à 128 téraoctets.

Pour faciliter l'accès et la manipulation des gros volumes, SQL intègre de plus en plus des fonctions analytiques. Ainsi, les améliorations du langage SQL continueront à fournir un soutien complet pour le développement d'applications polyvalentes, évolutives et performantes de base de données.

1.2. OUTILS SUPPORTS DE SQL

Oracle fournit un certain nombre de services pour faciliter votre processus de développement avec SQL:

- SQL * Plus est un outil permettant de lancer les requête d'une manière interactive ou via des scripts. Cet outil est installé à l'installation du serveur ou client de base de données Oracle. Il dispose d'une interface utilisateur de ligne de commande.

- Oracle SQL Developer est un outil graphique qui permet de gérer des objets de la base de données, éditer et déboguer des procédures PL / SQL, exécuter des instructions ou des scripts SQL, manipuler et exporter des données, et de créer et afficher des rapports. Oracle SQL Developer permet de se connecter à n'importe quel schéma du SGBDR Oracle afin d'effectuer des opérations sur les objets. Il permet aussi de se connecter à des schémas non-Oracle comme MySQL, Microsoft SQL Server et Microsoft Access et d'assurer la migration de données vers Oracle

- SQL Workshop est un composant d'Oracle HTML DB (environnement de développement d'applications Web). SQL Workshop vous permet de visualiser et de gérer des objets de base de données à partir d'un navigateur Web.

- Oracle JDeveloper est un environnement de développement intégré multi-plate-forme supportant le cycle de vie complet de développement pour Java, des services Web et SQL. Il fournit une interface graphique pour l'exécution et le réglage des instructions SQL et un composant de modélisation de base de données. JDeveloper permet d'éditer, compiler et déboguer les applications PL / SQL.

L'interface OCI (Oracle Call Interface) permet d'intégrer des instructions SQL dans les programmes C.

Les pré-compilateurs Oracle, tels que Pro * C / C++ et Pro * COBOL, permettent d'interpréter les instructions SQL intégrées et les traduire en instructions compréhensibles par les compilateurs des langages C / C ++ et COBOL.

1.3. CONVENTION LEXICALE

Dans une instruction SQL, les espaces, les retours chariot et les commentaires sont autorisés. Ainsi, la base de données Oracle évalue les deux requêtes suivantes de la même manière:

```
SELECT idarticle, designation, prixunit *0.5
FROM article;

SELECT idarticlecile, - - code article
      designation,
      prixunit*0.5
FROM  article;
```

La case sensitivité est négligeable dans les mots réservés, mots-clés, identificateurs et paramètres. Cependant, elle est importante avec les données de la base.

1.4. LES STANDARDS SQL

On peut penser que la place importante que SQL ait su prendre au fil des ans est très liée aux efforts de normalisation qui ont été entrepris sur ce langage.

Tout a commencé avec l'adoption par l'ANSI en 1986 d'une première norme sous la référence ANSI X3.135-1986, qui ne définissait pas clairement les comportements à adopter face aux problèmes d'intégrité.

C'est en 1989, qu'est né le SQL que l'on connaît aujourd'hui. Sous le nom de norme SQL89 (SQL 1), l'ANSI définissait un nouveau langage, intersection des langages SQL existants, qui pouvait gérer les problèmes d'intégrité référentielle, et qui permettait d'inclure SQL dans d'autres langages.

En 1992, apparaissait la norme SQL 92 (SQL 2) sur-ensemble de SQL 89 et qui est implémentée par les principaux SGBD tels que Oracle, DB2, MySQL, PostgreSQL, Access et SQL Server…Puis en 1999 la norme SQL 1999 (SQL 3) comportant le modèle Objet-Relationnel. Et depuis le travail de mise à jour, recherche et enrichissement du langage continue donnant naissances à SQL 2003, SQL 2006, SQL 2008 et récemment SQL 2011 ou ISO / IEC 9075:2011 (sous le titre général «Technologie de l'information - Langage de base de données SQL -»). C'est la septième révision de la norme ISO (1987) et ANSI (1986) pour le standard SQL . Il a été officiellement adopté en Décembre 2011.

1.5. LES APPORTS DE SQL

Le premier avantage de SQL est de ne pas être procédural. Ainsi au lieu de décrire comment on manipule les données, on décrit le résultat que l'on veut. C'est alors le moteur SQL qui travaille pour obtenir ce résultat.

Cette approche fait de SQL un langage destiné à une large population d'utilisateurs, qui peut aller aussi bien du néophyte en informatique, qu'au programmeur chevronné.

Enfin l'existence d'une norme respectée fait de SQL un langage privilégié pour la manipulation des bases de données relationnelles multiplateformes.

2. DEFINITION DES DONNEES

2.1. CREATION D'UNE TABLE

L'objet table est l'objet principal dans une base de données. L'instruction CREATE TABLE permet de créer une table. Une table est définie par un nom et un certain nombre de colonnes. Le nom de la table est unique au sein d'un schéma. Dans une table, chaque colonne porte un nom unique et correspond à un type de données. Optionnellement, une colonne peut être soumise à une ou plusieurs contraintes.

CREATE TABLE [IF NOT EXISTS] table_name
(create_definition,...);

« create_definition » est composée des définitions de toutes les colonnes (column_definition) de la table suivi des définition des contraintes.

create_definition se présente comme suit :

col_name column_definition
| [CONSTRAINT [symbol]] PRIMARY KEY [index_type] (index_col_name,...)
| [CONSTRAINT [symbol]] FOREIGN KEY
[index_name] (index_col_name,...) reference_definition
| [CONSTRAINT] UNIQUE [INDEX|KEY] [index_name] [index_type]
(index_col_name,...)
| CHECK (expr)

Tel que:

column_definition:
data_type [NOT NULL] [AUTO_INCREMENT] [UNIQUE [KEY] | [PRIMARY] KEY]

reference_definition:
REFERENCES table_name (index_col_name,...) [ON DELETE [CASCADE | SET NULL]]

Par défaut, la valeur « Null » est attribuée à toutes les colonnes de la table.

Pour voir plus la syntaxe complète de create table utiliser la commande : help create table.

2.1.1. TYPES DE DONNEES

Types numériques :

INT(INTEGER) : nombre entier négatif ou positif

NUMERIC et DECIMAL : nombre négatif ou positif. Les deux types sont équivalents et acceptent deux paramètres : la précision et l'échelle. La précision définit le nombre de chiffres significatifs stockés. Les 0 à gauche ne comptent pas. L'échelle définit le nombre de chiffres après la virgule.

DOUBLE : ne supporte pas les paramètres

Types alphanumériques :

CHAR (n): chaîne de caractères de longueur fixe 'n'.

VARCHAR (n): chaîne de caractères de longueur variable, maximum 'n' caractères.

TEXT : chaîne de caractères de longueur de plus de 255 caractères

ENUM : définit un certain nombre de valeurs autorisées, de type "chaîne de caractère". La colonne prend ainsi une de ces valeurs.

SET : définit un certain nombre de valeurs autorisées, de type "chaîne de caractère". La colonne est composée d'une ou de plusieurs de ces valeurs.

Type temporels:

DATE: stocke une date au format 'AAAA-MM-JJ'

TIME : stocke l'heure au format 'HH:MM:SS'

DATETIME : stocke une date et une heure au format 'AAAA-MM-JJ HH:MM:SS'

2.1.2. TYPES DE CONTRAINTES

- NOT NULL : interdit l'insertion d'une valeur nulle pour la colonne associée
- UNIQUE : interdit l'insertion d'une valeur déjà existante dans la colonne associée
- PRIMARY KEY : garantit NOT NULL et UNIQUE à la fois
- REFERENCES KEY : les valeurs de la colonne associée font référence à une colonne externe. Cette dernière est PRIMARY KEY ou unique
- CHECK : vérifie la validité d'une condition. La valeur saisie est acceptée si le test de validité retourne vrai

Exemple

```
Création de la table client
create table client
(idclient     integer auto_increment primary key,
raisonsociale VARCHAR(20) not null,
adresse       VARCHAR(15)
);
```

```
Création de la table commande
create table commande
(numcom        integer,
 datecom       date not null,
 idclient         integer,
 constraint PK_numcom primary key(numcom)
 ) ;

ALTER TABLE commande ADD CONSTRAINT FK_cde_idcl FOREIGN KEY
(idclient) REFERENCES client(idclient);
```

2.1.3. OPTIONS DE LA FOREIGN KEY

- **ON DELETE CASCADE**

Indique que la suppression d'une ligne à clé primaire parente implique les suppressions des lignes à FK correspondantes

- **ON DELETE SET NULL**

Indique que la suppression d'une ligne à clé primaire parente implique l'initialisation à NULL des lignes à FK correspondantes

```
 ALTER TABLE commande drop foreign key FK_cde_idcl;
 ALTER TABLE commande add foreign key FK_cde_idcl REFERENCES
client(idclient)on delete set null ;
```

2.2. DESCRIPTION D'UNE TABLE

La commande DESC (ou DESCRIBE) permet de décrire la structure d'une table existante.

Exemple

```
Description de la table client
        desc client ;
```

Les tables système INFORMATIONS_SCHEMA.TABLES et INFORMATIONS_SCHEMA. TABLE_CONSTRAINTS contiennent plus de détails sur les tables

Exemple

```
Use information_schema;
Show TABLES ;

 Interrogation de TABLES
    select TABLE_NAME, TABLE_TYPE
    from TABLES
    where TABLE_SCHEMA='GestCom';
```

```
Interrogation de TABLE_CONSTRAINTS
    Select TABLE_NAME,CONSTRAINT_NAME,CONSTRAINT_TYPE
    from TABLE_CONSTRAINTS where TABLE_SCHEMA='GestCom';
```

2.3. MODIFICATION D'UNE TABLE

Une table créée peut être modifiée par l'instruction ALTER TABLE comme suit :

2.3.1. Ajout d'une colonne avec ou sans contrainte

ALTER TABLE nom_table
ADD nom_colonne type_données type_contrainte ;

S'il s'agit d'ajouter plusieurs colonnes à la fois :

ALTER TABLE nom_table
ADD (nom_colonne1 type_données type_contrainte
, nom_colonne2 type_données type_contrainte
,…) ;

Exemple

```
Ajout de la colonne ville à la table client
        alter table client
        add ville VARCHAR(15) not null;

       create table article
         (
          idarticle integer ,
          designation VARCHAR(20)
          ) ;

Ajout de la colonne prix à la table article avec une
contrainte de validité

       alter table article add prixunit double CHECK(prixunit
       >=100);
```

2.3.2. Ajout d'une contrainte NOT NULL

ALTER TABLE nom_table
MODIFY COLUMN nom_colonne type_données type_contrainte ;

Exemple

```
Ajout de la contrainte NOT NULL
alter table client modify column adresse varchar(20) NOT NULL;
```

2.3.3. Agrandir la taille d'une colonne

ALTER TABLE nom_table
CHANGE COLUMN nom_colonne nouveau_nom type_données(nouvelle_taille) ;

Exemple

```
Augmentation de la taille de raisonsociale dans la table client
alter table client change column raisonsociale raisonsociale varchar(30) ;
```

2.3.4. Renommer/Deplacer une colonne

ALTER TABLE nom_table
CHANGE COLUMN nom_colonne nouveau_nom type_données AFTER nom_colonne1;

Exemple

```
Renommer la colonne prixunit  en prix et la déplacer après la colonne Désignation

alter table article change column prixunit prix double after designation;

desc article ;
```

2.3.5. Suppression d'une (de plusieurs) colonne(s)

ALTER TABLE nom_table
DROP COLUMN nom_colonne ;

Exemple

```
Supprimer la colonne client.adresse

alter table client drop column adresse;

desc client ;
```

2.3.6. Ajout de la contrainte PRIMARY KEY

ALTER TABLE nom_table
ADD primary key (colonne_clé) ;

Exemple

```
Ajout de la contrainte nommée Pk_idarticle à la table article
         alter table article
         add primary key (idarticle);
```

2.3.7. Ajout de la contrainte REFERENCES (Foreign key)

ALTER TABLE nom_table
ADD [constraint nom_contrainte] Foreign key (nom_colonne_FK)
REFERENCES nom_table_contenant_PK (nom_colonne_PK) ;

Exemple

```
Création de la table representant
     create table representant
     (idrep integer(11) primary key,
      nomrep VARCHAR(15),
      idclient  integer(5)) ;

Ajout d'une contrainte de référence sur idclient, liant ainsi
la table representant à table client
   alter table  representant
   add constraint fk_idcl foreign key(idclient)
   references client(idclient) ;
```

2.3.8. Suppression d'une contrainte Foreign Key

ALTER TABLE nom_table
DROP Foreign Key nom_contrainte ;

2.3.9. Suppression d'une contrainte Primary Key

ALTER TABLE nom_table
DROP Primary Key ;

Exemple

```
Suppression de la contrainte de foreign key dans commande :

alter table commande drop foreign key `FK_cde_idcl`;
```

2.4. SUPPRESSION D'UNE TABLE

La commande DROP TABLE permet de supprimer une table.

DROP TABLE [IF EXISTS] nom_table [, nom_table] ...;

Toutes les lignes de la table et la définition elle-même seront supprimées.

Exemple

```
Suppression des tables ligne_com et commande
     drop table if exists commande;
```

Si des contraintes d'intégrités existent entre les tables, la suppression échoue. il faut lancer la commande suivante : set FOREIGN_KEY_CHECKS=0;

2.5. RENOMER UNE TABLE

La commande RENAME TABLE permet de renommer une table.

RENAME table nom_table TO nouveau_nom;

Exemple

```
Renommer la table article en table Produit
    Rename table article to produit;
```

2.6. COPIER UNE TABLE

La commande CREATE AS SELECT permet de créer une table et lui injecter le résultat de la requête select.

CREATE TABLE nom_table
AS
SELECT …;

Exemple

```
Création de la table client75
```

```
create table client75
as
select *
from client
where upper(ville)='PARIS';
```

3. LANGAGE DE MANIPULATION DE DONNEES

L'unité manipulée est la ligne.

Il existe trois commandes SQL permettant d'effectuer les trois types de mise à jour d'une base de données.

- INSERT : Ajout de lignes,
- UPDATE : Modification de lignes,
- DELETE : Suppression de lignes.

3.1. AJOUT DE LIGNES

La commande INSERT permet d'ajouter une ligne à une table, en précisant la valeur affectée à chaque champ.

3.1.1. INSERTION D'UNE LIGNE

INSERT INTO nom_ table [(nom_colonne1, nom_colonne2, ..., nom_colonne n)]
VALUES (valeur1, valeur2, ..., valeur n) ;

La liste des colonnes est optionnelle. Par défaut, la liste inclus toutes les colonnes de la table dans l'ordre de leur création.
Les colonnes qui ne sont pas spécifiées dans la liste prennent la valeur NULL (sauf la primary key).

Exemple

```
Insertion d'une ligne dans la table Client
  insert  into client (raisonsociale,ville)
   values ('client1','ville1') ;
```

3.1.2. INSERTION DE PLUSIEURS LIGNE

INSERT INTO nom_ table [(nom_colonne1, nom_colonne2, ..., nom_colonne n)]
VALUES (valeur1, valeur2, ..., valeur n) ,
(valeur10, valeur20, ..., valeur n0) ,
(valeur100, valeur200, ..., valeur n00) ,
...;

Exemple

```
Insertion de plusieurs lignes dans la table Client
  insert into client(raisonsociale, ville)
  values ('SEPHORA', 'TOULOUSE'),
         ('SEPHORA', 'PARIS'),
         ('SEPHORA', 'BORDEAUX');
```

Il est possible de charger des lignes provenant d'une autre table

INSERT INTO nom_table [(nom_colonne1, nom_colonne2, ..., nom_colonne n)]
SELECT… ;

La commande SELECT est du type sous interrogation.

3.2. MODIFICATION DE LIGNES

Cette instruction autorise la modification d'une (ou de plusieurs) colonne(s), dans une ou plusieurs lignes d'une table.

UPDATE nom_table
SET nom_colonne1= expr 1,
nom_colonne2= expr 2,
…
[WHERE condition(s)] ;

Les valeurs des colonnes sont modifiées dans toutes les lignes satisfaisant aux conditions.

En l'absence de clause WHERE toutes les lignes de la table seront modifiées.

Exemple

```
Augmenter de 5 % les prix des articles
    update article
    set prixunit = prixunit * 1.05;
```

3.3. SUPPRESSION DE LIGNES

L'instruction DELETE permet d'effacer une ou plusieurs lignes d'une table.

DELETE FROM nom_table
[WHERE condition(s)];

En l'absence de clause WHERE, toutes les lignes de la table spécifiée seront effacées.
En cas contraire, toutes les lignes répondant au prédicat seront supprimées.

Exemple

```
Supprimer tous les articles dont le prix est < 100
    delete from article
    where prixunit < 100;
```

3.4. VALIDATION OU ANNULATION DE TRANSACTION

Une transaction est définie par une suite d'instructions de type INSERT, UPDATE et DELETE sur les données.

Pour garantir le partage cohérent des données, il est nécessaire de valider les mises à jour des données.

L'ordre '**COMMIT** ' permet de valider les mises à jour depuis le dernier COMMIT ou ROLLBACK.

L'ordre '**ROLLBACK**' permet d'annuler les mises à jour depuis le dernier COMMIT ou ROLLBACK.

3.5. SELECTION DE DONNEES

L'interrogation d'une base de données constitue l'opération la plus généralement utilisée en langage SQL et s'effectue grâce à l'instruction SELECT

SELECT [DISTINCT] nom_colonne1, nom_colonne2,…
FROM nom_table
[WHERE condition(s)] ;

Sa forme la plus élémentaire est :

SELECT *
FROM nom_table ;

Où "*" signifie que toute les colonnes de la table associée doivent être projetées.

On peut limiter le nombre de lignes retournées par le select en utilisant l'option **Limit**.

Exemple

```
Sélection de toutes les informations sur les représentants
   select *
   from representant;
```

```
Affichage de 3 clients avec un idclient compris entre 3 et 10
   select * from client
   where idclient between 3 and 10
   limit 3;
```

La sélection peut être limitée à un choix de colonnes, en indiquant, à la place de l'astérisque, une liste de colonnes.

Exemple

```
Sélection de quelques informations sur les représentants
   select idrep, nomrep, ville
   from representant;
```

La clause DISTINCT ajoutée à l'instruction SELECT permet d'éliminer les lignes répétitives.

Exemple

```
Sélection des différentes villes des représentants sans redondance
    select distinct ville
    from representant;
```

3.5.1. SELECTION DE LIGNES

La clause WHERE, associée à l'opérateur SELECT, permet de spécifier quelles sont les lignes à sélectionner.

Cette clause est suivie d'un prédicat (expression logique ayant soit la valeur vrai, soit la valeur faux) qui sera évalué pour chaque ligne de la table.

Les lignes pour lesquelles le prédicat est vrai sont ainsi sélectionnées.

3.5.2. PREDICAT SIMPLE

C'est le résultat de la comparaison de deux expressions au moyen d'un opérateur de comparaison :

- Opérateurs arithmétiques

= : égal,
!= ou <>: différent,
> : supérieur,
< : inférieur,
<= : inférieur ou égal,
>= : supérieur ou égal,

Exemple

```
Sélection des informations sur la commande numéro 2
  select *
  from commande
  where numcom=2;
Sélection des clients qui ne sont pas de PARIS
  select raisonsociale
  from client
  where  upper(ville) !='PARIS';--upper  réécrit  la  ville  en
--majiscule

Sélection des articles dont les prix sont compris entre 200et
1000
  select designation, prixunit
  from article
  where prixunit>=200 and prixunit<=1000;
```

- **BETWEEN < expr 1 > AND < expr 2 >** : vrai si l'expression à tester est comprise entre les deux bornes (incluses) déterminées par < expr 1 > et < expr 2 >.

Exemple

```
Sélection des articles dont les prix sont compris entre 200et
1000
  select designation, prixunit
  from article
  where prixunit between 200 and 1000;
```

- **IN < expr 1, expr 2, ... >** : vrai si l'expression à tester appartient à la liste d'expressions cité.

Exemple

```
Sélection des représentants dont idrep = 1 ou 3 ou 5
   select idrep, nomrep
   from representant
   where idrep in (1,3,5);
ou bien
   select idrep, nomrep
   from representant
   where idrep =1
   or idrep=3
   or idrep=5;
```

- **LIKE < chaîne >** : vrai si l'expression à tester (type chaîne) contient < chaîne >.
  Les caractères spéciaux '_' et '%' peuvent apparaître dans < chaîne > :
  '_' (tiret bas) : utilisé avec LIKE, remplace exactement un caractère,

'%' : utilisé avec LIKE, remplace une séquence quelconque de caractères.

Exemple

```
Sélection des clients dont le nom commence par D
   select raisonsociale
   from client
   where upper(raisonsociale) like 'D%' ;

Sélection des clients dont le nom comprend un A en deuxième
position
   select raisonsociale
   from client
   where upper(raisonsociale) like '_A%' ;
```

3.5.3. PREDICAT COMPOSE - OPERATEURS LOGIQUES

Les opérateurs logiques AND et OR sont utilisés pour combiner plusieurs prédicats.

L'opérateur NOT est utilisé pour la négation. Not IN, NOT BETWEEN et NOT LIKE sont les négations de IN, BETWEEN et LIKE

Exemple

```
Sélection des articles dont la désignation commence par E et
le prix entre 200 et 300
      select designation, prixunit
      from article
      where upper(designation) like 'D%'
      and prixunit between 200 and 300;


Sélection des clients dont idclient est plus petit que 3 et
qui sont de PARIS ou de LYON
      select raisonsociale, idclient, ville
      from client
      where idclient <3
      and ( upper(ville) in ('PARIS','LYON'));


Sélection des représentants dont idrep est différent de 1 et
de 3 et de 5 et la ville est PARIS ou LYON

     select idrep, nomrep
     from  representant
     where idrep not in (1,3,5)
     and upper(ville) in ('PARIS','LYON');
```

```
Ou bien
     select idrep, nomrep
     from  representant
     where idrep not in (1,3,5)
     and (upper(ville)='PARIS' or upper(ville)='LYON');
```

L'opérateur 'AND' est prioritaire par rapport à l'opérateur 'OR'.
Des parenthèses peuvent être utilisées pour modifier ces priorités ou plus simplement pour clarifier la requête.

3.6. OPERATIONS ENSEMBLISTES

- Opération d'union (UNION):

SELECT liste_colonnes FROM nom_table1
UNION
SELECT liste_colonnes FROM nom_table 2;

On peut aussi utiliser NOT IN et NOT EXISTS

SELECT nom_colonne1, nom_colonne2,...
FROM nom_table1
WHERE nom_colonne1 NOT IN (SELECT nom_colonne1 FROM nom_table2);

SELECT nom_colonne1, nom_colonne2,…
FROM nom_table1
WHERE NOT EXISTS (SELECT *
FROM nom_table2
WHERE nom_table1.nom_colonne1=nom_table2.nom_colonne1) ;

Exemple

```
Sélection de l'ensemble des données : raisonsociale de Client
et nom de representant
      select concat('NOM DU REPRESENTANT: ', nomrep)  Resultat
      from representant
      union
      select concat('RAISONSOCIALE DU CLIENT: '
      ,raisonsociale)
      from client;

Sélection de l'ensemble des données : idclient de Client pour
les clients qui n'ont pas commandé
      select  idclient
      from client
      where idclient not in(select idclient from commande);
```

```
Sélection de l'ensemble des données : idclient de Client pour
les clients qui ont commandé

      select  idclient
      from client
      where idclient in(select idclient from commande);
```

3.7. OPERATEUR DE CONCATENATION

L'opérateur **Concat** permet de concaténer des chaines de caractères

Exemple

```
Sélection des articles dont la désignation commence par E et
le prix entre 200 et 300
      select concat('Designation de l''article: ',designation,
             '   ','prix unitaire: ',prixunit)  Détails
      from article
      where upper(designation) like 'D%'
      and prixunit between 100 and 300;
```

3.8. LES CONSTANTES

Les constantes apparaissant dans l'ordre SELECT en tant qu'expressions peuvent être de trois types :

- numérique,
- date,
- caractères.

A chacun d'eux correspond un format particulier :

- **Constante numérique** :
  nombre pouvant contenir un signe, un point décimal et un exposant puissance de 10.

- **Constante alphanumérique** : chaîne de caractères entre simples quottes.
  Une apostrophe peut être incluse dans la chaîne en la doublant.

- **Constante date** : chaîne de caractères entre simples quottes au format jour-mois-année.

Exemple

```
Sélection des informations sur deux commandes
   select *
   from commande
   where datecom in ('2017-02-12','2016-12-12');
```

3.9. LA DATE SYSTEME

La variable prédéfinie SYSDATE charge la date système

Exemple

```
Sélection de la date du système
   select  sysdate();

Sélection de l'heure
   select date_format(sysdate(), '%T') heure_systeme;
```

3.10. PARAMETRAGE DU FORMAT DE LA DATE

La fonction **DATE_FORMAT(*date, format*)** permet le formatage d'une date dans le format résultant d'une combinaison du tableau suivant :

%%	Le caractère '%'
%d	Jour du mois en numérique
%m	Mois de l'année en numérique
%Y	Année sur 4 chiffres (YYYY)
%H	Heures, sur 24 heures
%i	Minutes
%s	Secondes
%T	Heure complète (hh:mm:ss)

3.11. EXPRESSIONS ET FONCTIONS

Une expression est une combinaison de variables (contenu d'une colonne), de constantes et de fonctions.
Les expressions peuvent figurer :

- en attribut projeté dans la clause SELECT
- dans une clause WHERE
- dans une clause ORDER BY

Exemple

```
Sélection des articles et leurs prix avant et après une
réduction de 50%
    select designation, prixunit
    ,prixunit*0.5 prix_reduit
    from article
    order by prix_reduit asc ;
```

```
Sélection du prix d'une  commande de 100 téléviseurs
      select prixunit*100 Total
      from article
      where upper(designation)='TV LED-3D' ;
```

3.12. LA VALEUR NULL

Une valeur Null est une valeur non définie, différente donc de la valeur 0. On peut comparer à cette valeur en utilisant les opérateurs IS Null ou IS Not Null.

Exemple

```
Sélection des clients dont l'adresse est non renseignée
   select idclient,raisonsociale,adresse
   from client
   where adresse is Null;
```

3.13. LA COMMANDE CASE

Comme dans un langage de programmation, dans SQL la commande

CASE...
WHEN...
THEN...
ELSE…
END

permet d'utiliser le traitement conditionnel

IF…THEN…ELSE…END IF.

La commande est utilisée pour:

- Comparer une colonne à une liste de valeurs possible

Exemple

```
Sélection des noms et des secteurs des représentants commerciaux
        select idrep,nomrep
        , (case idrep
         when 1 then 'Cosmetique'
         when 20 then 'Pharmaceutique'
         else
        'Informatique'
```

```
         end) as SECTEUR
        from representant;
```

- Élaborer une série de conditions booléennes pour déterminer un résultat

Exemple

```
Sélection des articles par tranche de prix
        select designation, prixunit
        , (case
         when prixunit<300 then 'TRANCHE1'
         when prixunit between 300 and 1000 then 'TRANCHE2'
         else
        'TRANCHE3'
         end) as TRANCHE
        from article
        order by TRANCHE;
```

La commande peut être utilisée dans n'importe quelle instruction ou clause, telle que SELECT, UPDATE, DELETE, WHERE, ORDER BY ou HAVING.

Exemple

```
Mise à jour des prix des articles :
Prixunit <300              réduction de 50%
Prixunit entre 300 et 1000 réduction de 30%
Prixunit >1000             réduction de 10%
        update article
        set prixunit=
        (case when prixunit <300 then prixunit*0.5
         when prixunit between 300 and 1000 then prixunit*0.7
         else prixunit*0.9
         end);
```

3.14. EVALUATION D'UNE EXPRESSION ARITHMETIQUE

Une expression arithmétique peut comporter plusieurs opérateurs, le résultat de l'expression peut varier selon l'ordre dans lequel sont effectuées les opérations.

Les opérateurs de multiplication et de division sont prioritaires par rapport aux opérateurs d'addition et de soustraction.

Des parenthèses peuvent être utilisées pour forcer l'évaluation de l'expression dans un ordre différent de celui découlant de la priorité des opérateurs ou simplement pour clarifier l'écriture.

3.15. EXPRESSION DE TYPE DATE

Grâce aux opérateurs arithmétiques '+' et '-', il est possible de construire les expressions :

- **date +/- interval** nombre [**day | months | year**] : résultat de type date obtenu en ajoutant (ou en retranchant) le nombre de jours,mois ou année précisé à la date.

Exemple

```
Insertion d'une commande dans la table commande
  alter table commande
  add dateliv date ;

  insert into commande (numcom,idclient,datecom,dateliv)
  values (10, 6, sysdate(), sysdate()+ interval 7 day) ;

  insert into commande (numcom,idclient,datecom,dateliv)
  values (20, 6, sysdate() - interval 3 month, sysdate() -
  interval 6 day);

  select numcom,datecom,dateliv + interval 2 day  livraison
  from commande;
```

3.16. TRI DU RESULTAT

Les lignes constituant le résultat d'une instruction SELECT sont obtenues dans un ordre indéterminé dépendant des mécanismes internes du SGBDR utilisé.

On peut demander, en fin d'instruction SELECT que le résultat soit ordonné de manière ascendante ou descendante suivant une ou plusieurs colonnes.

Les critères de tri sont indiqués dans la clause ORDER BY dont la syntaxe est :

SELECT liste_colonnes
FROM nom_table
WHERE condition(s)
ORDER BY nom_colonne 1 [ASC|DESC], nom_colonne 2[ASC|DESC],… ;

Le tri se fait tout d'abord selon la première colonne citée, puis les lignes ayant une même valeur pour celle-ci sont classées selon la deuxième colonne, etc.

Par colonne, le tri peut être ascendant ASC (valeur par défaut) ou descendant DESC.

Exemple

Sélection des articles classés dans l'ordre décroissant du stock

```
select designation, qtestock
from   article
where  qtestock is not null
order  by qtestock desc;
```

4. JOINTURES

La jointure est l'opération permettant d'obtenir des informations provenant de plusieurs tables.

4.1. EQUIJOINTURE

Une jointure est formulée en spécifiant simplement plusieurs tables dans la clause FROM de la commande SELECT. Les conditions de jonction sont des égalités.

En SQL de base :

```
SELECT    liste_colonnes
FROM      nom_table1, nom_table2…
WHERE  condition(s)_de_jointure;
```

4.2. JOINTURE INTERNE

Avec la clause INNER JOIN (jointure interne) à partir du SQL2, supporté aujourd'hui par tous les SGBDR

```
SELECT    *
FROM      nom_table1
INNER JOIN nom_table2
ON condition_jointure_table1_table2
INNER JOIN nom_table3
ON condition_jointure_table2_table3…
```

Le mot clé INNER est facultatif

Dans la première syntaxe, s'il y a omission de condition de jointure, le résultat sera le produit cartésien (Cross Join) entre les tables nom_table1, nom_table2.

La deuxième syntaxe ne retourne pas de résultat s'il y a omission de condition après ON. Pour obtenir un produit cartésien la syntaxe est la suivante :

```
SELECT *
FROM NOM_TABLE1
CROSS JOIN  NOM_TABLE2;
```

Exemple

TABLE REPRESENTANT

IDREP	NOM	...
1	REP1	

TABLE CLIENT

IDREP	RS
1	CLI1
1	CLI2

REPRESENTANT.IDREP = CLIENT.IDREP

```
Le produit cartésien s'obtient comme suit:
    select nomrep, raisonsociale
    from representant , client;
Ou bien
    select *
    from representant
    cross join client;

Sélection de la liste des représentants et leurs clients
    select nomrep, raisonsociale
    from representant , client
    where representant.idrep=client.idrep\G
Ou bien
    select nomrep, raisonsociale
    from representant
    JOIN client ON representant.idrep=client.idrep\G
```

4.3. LA CLAUSE USING

La clause **USING** peut être utilisée dans une équijointure pour préciser la colonne commune aux deux tables

Exemple

```
Sélection de la liste des représentants et leurs clients

    select nomrep, raisonsociale
    from representant join client
    using (idrep);
```

4.4. ALIAS DE TABLE

Un alias de table peut être donné à un nom de table, dans la clause FROM d'une commande SELECT.

Ceci permet, entre autres choses, d'éviter la répétition d'un nom de longueur importante.

Exemple

```
Sélection de la liste des représentants et leurs clients

    select nomrep, r.ville, raisonsociale, c.ville
    from representant r join client c
    where r.idrep=c.idrep;
```

4.5. JOINTURE NATURELLE

L'équijointure naturelle (NATURAL JOIN) joint deux tables sur l'ensemble des colonnes qu'elles ont en commun. Les colonnes communes ne sont affichées qu'une seule fois.

SELECT *
FROM nom_table1 **NATURAL JOIN** nom_table2**;**

Exemple

```
Sélection de la liste des représentants et leurs clients
    select *
    from client  natural join commande\G

La colonne commune idclient s'affiche qu'une seule fois.
```

4.6. JOINTURE EXTERNE

La jointure externe est une équijointure entre deux tables favorisant en affichage les colonnes de l'une des deux tables par rapport à l'autre.

4.6.1. JOINTURE EXTERNE A GAUCHE

Avec la clause LEFT OUTER JOIN (jointure externe à gauche) à partir du SQL2

SELECT *
FROM nom_table1 T1
LEFT OUTER JOIN nom_table2 T2
ON T1.colonne_de_jointure=T2.colonne_de_jointure ;

En plus du résultat de
SELECT *
FROM nom_table1 T1
INNER JOIN nom_table2 T2
ON T1.colonne_de_jointure=T2.colonne_de_jointure ; on obtient les

données de T1 qui n'ont pas de correspondance dans T2.

Le mot clé OUTER est facultatif

4.6.2. JOINTURE EXTERNE A DROITE

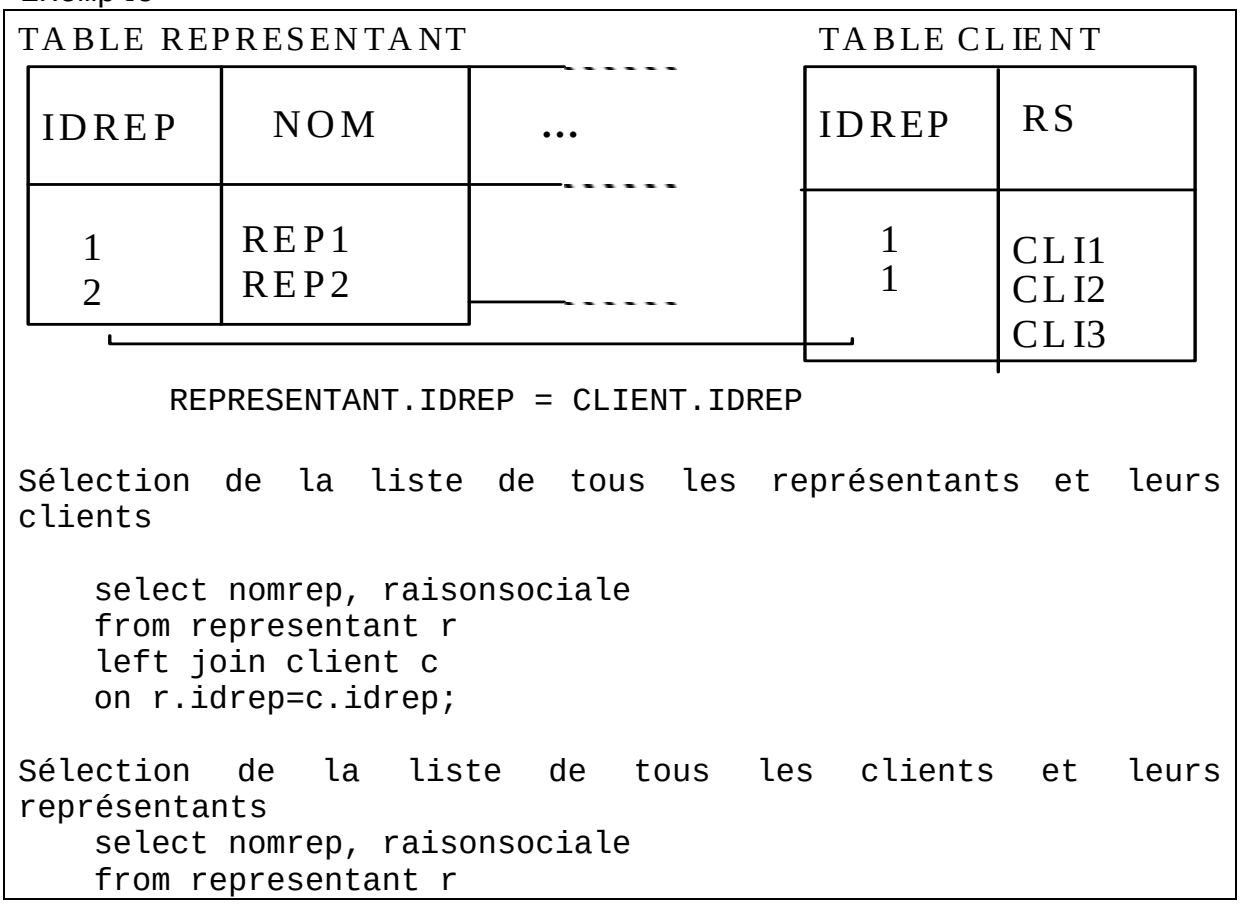
La clause RIGHT OUTER JOIN (jointure externe à droite) à partir du SQL2 est l'opposé de LEFT OUTER JOIN

**SELECT *
FROM nom_table1 T1
RIGHT OUTER JOIN nom_table2 T2
ON T1.colonne_de_jointure=T2.colonne_de_jointure ;**

En plus du résultat de
SELECT *
FROM nom_table1 T1
INNER JOIN nom_table2 T2
ON T1.colonne_de_jointure=T2.colonne_de_jointure ; on obtient les données de T2 qui n'ont pas de correspondance dans T1.

Le mot clé OUTER est facultatif

Exemple




```
Sélection de la liste de tous les représentants et leurs
clients

    select nomrep, raisonsociale
    from representant r
    left join client c
    on r.idrep=c.idrep;

Sélection de la liste de tous les clients et leurs
représentants
    select nomrep, raisonsociale
    from representant r
```

```
    right join client c
    on r.idrep=c.idrep;
```

4.7. JOINTURE D'UNE TABLE A ELLE-MEME

La jointure SELF JOIN peut être utile de rassembler des informations venant d'une ligne d'une table avec des informations venant d'une (plusieurs) ligne (s) de la même table.

Exemple

```
Sélection des articles qui coûtent autant que 'TELEVISEUR'
    select a1.designation, a1.prixunit, a2.designation
    from article a1 join article a2
    where a1.prixunit = a2.prixunit
    and a1.designation = 'TV LED-3D' ;

Afin d'éviter la comparaison de 'TV LED-3D' à lui-même on
écrit

    select a1.designation, a1.prixunit, a2.designation
    from   article a1 join article a2
    where  a1.prixunit = a2.prixunit
    and    a1.designation = 'TV LED-3D'
    and    a2.designation <> 'TV LED-3D';
```

4.8. AUTRES TYPES DE JOINTURES

L'égalité est le critère de jointure le plus naturel donc le plus fréquemment utilisé.

Cependant, d'autres types de comparaison peuvent être utilisés comme critères de jointure (<, >, <=, >=, < >).

Exemple

```
Sélection des articles qui coûtent plus que 'Disque dur 2To'
    select     a1.designation, a1.prixunit, a2.designation
    from       article a1 join article a2
    where      a1.designation = 'Disque dur 2To'
    and        a1.prixunit < a2.prixunit;
```

5. FONCTIONS D'AGREGAT

Il est possible d'effectuer des calculs sur l'ensemble des valeurs d'une colonne, à l'aide des fonctions de groupe.

Les valeurs nulles sont ignorées par les fonctions de groupe.

5.1. CALCUL DE LA MOYENNE AVG()

La fonction AVG() appliquée à une colonne numérique retourne la moyenne des valeurs de celle-ci.

Exemple

```
Sélection de la moyenne des prix des articles
    select avg(prixunit) moyenne_prix
    from article;
```

Un SELECT de groupe peut être utilisé aussi dans une sous-requête.

Exemple

```
Sélection des articles dont le prix est supérieure à la
moyenne
    select designation, prixunit
    from article
    where prixunit > (select   avg(prixunit)
    from article);
```

5.2. CALCUL DU MINIMUM MIN()

La fonction MIN() appliquée à une colonne numérique retourne la valeur la plus petite de celle-ci.

Exemple

```
Sélection du prix de l'article le moins cher
    select min(prixunit) prix_minimum
    from article;
```

5.3. CALCUL DU MAXIMUM MAX()

La fonction MAX() appliquée à une colonne numérique retourne la valeur la plus grande de celle-ci.

Exemple

```
Sélection du prix de l'article le plus élevé
```

```
    select max(prixunit) prix_maximum
    from article;

calcul de la différence entre le prix le plus élevé et le prix
le plus bas
    select max(prixunit),min(prixunit),
    max(prixunit)-min(prixunit) diff_prix
    from article ;
```

5.4. COMPTAGE DE VALEURS COUNT()

La fonction COUNT() appliquée à une colonne retourne le nombre de valeurs de celle-ci.

COUNT (*) donne le nombre de lignes satisfaisant au prédicat de la clause WHERE.

COUNT (nom_colonne) donne le nombre de lignes ayant une valeur non nulle dans la colonne en question.

Le nombre de valeurs différentes d'une colonne (ensemble au sens mathématique) est déterminé par l'expression COUNT (DINSTINCT nom_colonne).

Exemple

```
Sélection du nombre de lignes dans la table commande
    select count(*) nbre_commande
    from commande;

Sélection du nombre de commande pour le client 1
    select count(numcom) nbre_commande
    from commande
    where idclient=1;

Sélection du nombre des différentes villes des clients
    select count(distinct ville)
    from client;
```

5.5. CALCUL DE LA SOMME SUM( )

La fonction SUM() appliquée à une colonne numérique retourne la somme des valeurs de celle-ci.

Exemple

```
Sélection de la somme des quantités commandées de l'article 2
    select idarticle, sum(qtecom) quantité_total
    from ligne_com
```

```
    where idarticle=2 ;

Calcul du montant que rapport cet article
    select a.idarticle, sum(qtecom)*prixunit montant_cde
    from ligne_com l
    join article a
    on a.idarticle=l.idarticle
    where a.idarticle=2;
```

5.6. LA CLAUSE GROUP BY

Il est possible de subdiviser une table en groupes, chacun étant l'ensemble des lignes ayant une valeur commune.

Ces groupes sont définis par la clause GROUP BY:

SELECT liste_colonnes
FROM liste_tables
WHERE condition(s)
GROUP BY expr 1 [, expr 2 , ...];

Une commande SELECT comportant une fonction de groupe en résultat, donnera une ligne résultat pour chaque groupe.

Exemple

```
Calcul du nombre de commandes par client
    select idclient, count(numcom)
    from  commande
    group by idclient ;
```

Dans la liste des colonnes citées dans un SELECT de groupe, ne peuvent figurer que des caractéristiques de groupe :

- Soit des fonctions de groupe,
- Soit des expressions figurant dans la clause GROUP BY.

Exemple

```
    select raisonsociale,cl.idclient, count(numcom)
    from client cl
    join commande co
    on cl.idclient=co.idclient
    group by cl.idclient;
```

5.7. LA CLAUSE HAVING

De la même façon qu'il est possible de sélectionner certaines lignes, grâce à la clause WHERE, il est possible dans un SELECT de groupe, de sélectionner certains groupes seulement par la clause HAVING.

Le prédicat associé à HAVING suit les mêmes règles syntaxiques qu'un prédicat figurant dans une clause WHERE.

Il ne peut cependant porter que sur des caractéristiques de groupe : fonctions ou expressions figurant dans la clause GROUP BY.

Exemple

```
Sélection des clients qui ont passés plus de 2 commandes
    select idclient, count(numcom)
    from commande
    group by idclient
    having count(numcom)>2;

Sélection de la ville où habite le plus grand nombre de
clients
    select upper(ville), count(*)
    from client
    group by upper(ville)
    having count(*)>= all (select count(*) from client
    group by upper(ville));
```

6. SOUS-REQUETES IMBRIQUEES

6.1. SOUS-REQUETES A UNE VALEUR

Un critère de recherche (sélection) employé dans une clause WHERE peut être lui-même le résultat d'un SELECT.

Exemple

```
Sélection des clients ayant la même ville que CLI1
    select raisonsociale
    from client
    where upper(ville)=(select upper(ville)
                    from client
                    where raisonsociale ='CLI1');
```

6.1.1. SOUS-REQUETES SYNCHRONISEE AVEC LA REQUETE PRINCIPALE

Dans les exemples précédents, la sous-interrogation (requête imbriquée) pouvait être évaluée dans un premier temps puis son résultat utilisé pour l'exécution des niveaux supérieurs.

Cependant, une requête imbriquée peut faire référence à une colonne de la table utilisée dans une interrogation supérieure.

Le traitement est ainsi plus complexe puisqu'il faut évaluer la sous-interrogation pour chaque ligne de l'interrogation principale.

6.1.2. SOUS-INTERROGATIONS MULTIPLES

Une commande SELECT peut comporter plusieurs niveaux d'imbrication ou plusieurs sous-interrogations au même niveau combinés par les opérateurs logiques 'AND' et 'OR'.

Exemple

```
Sélection des clients des articles les plus commandés et ayant
un prix supérieure ou égale à la moyenne

select
c.raisonsociale,a.designation,a.prixunit,l.qtecom,l.numcom
from  client c join commande co
on co.idclient = c.idclient
join ligne_com l on l.numcom=co.numcom
```

```
join article a  on a.idarticle = l.idarticle
where l.qtecom = ( select max(l2.qtecom)
                    from ligne_com l2 join article a2
                   on a2.idarticle=l2.idarticle
                   where l2.numcom=l.numcom
                   and a2.prixunit >= (select avg(a3.prixunit)
                                       from article a3)
                                       group by l2.numcom);
```

6.2. SOUS-REQUETES RAMENANT PLUSIEURS LIGNES

Le résultat de la sous-requête doit être cohérent avec la manière dont il est traité au niveau supérieur.

Une sous requête peut ramener plusieurs lignes à condition que l'opérateur de comparaison admette à sa droite un ensemble de valeurs.

Les opérateurs permettant de comparer une valeur à un ensemble de valeurs sont :

- IN
- L'usage simultané d'un opérateur de comparaison (= !=, <, <=, ...) et d'un des mots-clefs :
- ANY : La comparaison sera vraie si elle est vraie pour au moins un élément de l'ensemble,
- ALL : La comparaison sera vraie si elle est vraie pour tous les éléments de l'ensemble.

Exemple

```
Sélection des articles qui n'ont pas été commandés
   Select designation
   from article
   where idarticle not in (select distinct idarticle
                           from ligne_com) ;
Sélection des articles dont la quantité en stock est 30 fois
supérieure à toute quantité commandée
    select a.idarticle,a.designation,a.qtestock,
    ifnull(l.qtecom,0)
    from article a
    left join ligne_com l
    on l.idarticle = a.idarticle
    where a.qtestock/30 > all(select l2.qtecom
                         from ligne_com l2
                         where l2.idarticle =  a.idarticle);

Sélection des articles dont la quantité en stock est 30 fois
supérieure à au moins une quantité commandée
    select a.idarticle,a.designation,a.qtestock,
    ifnull(l.qtecom,0)
    from article a
    left join ligne_com l
    on l.idarticle = a.idarticle
```

```
    where a.qtestock/30 > any(select l2.qtecom
                           from ligne_com l2
                           where l2.idarticle = a.idarticle);
```

Il est à noter que l'expression IN est équivalent à l'expression '= ANY', de même, NOT IN correspond à '!=ALL'.

7. AUTRES OBJETS SQL

Rappelons que l'objet principal est l'objet table.

7.1. LA VUE

Le concept de vue dans le modèle relationnel correspond à la notion de sous-schéma.

Une vue peut être considérée comme une table temporaire dérivée des tables du schéma par l'application d'une requête.

La raison d'être des vues est double :

- Augmenter l'indépendance logique,
- Préserver la sécurité des données.

7.1.1. CREATION D'UNE VUE

L'ordre CREATE VIEW permet de créer une vue en spécifiant la requête constituant la définition de la vue :

CREATE [OR REPLACE] VIEW nom_vue [(nom_colonne1, …)]
AS SELECT … ;

La liste des noms de colonnes est optionnelle, par défaut les colonnes de la vue auront les mêmes noms que les colonnes résultat du SELECT.

Exemple

```
Création d'une vue sur les noms des représentants parisiens et
les noms des clients correspondants
    create or replace view rep_cli as
    select nomrep, raisonsociale
    from representant r
    join client c
    on r.idrep=c.idrep
    where upper(r.ville)= 'PARIS';
```

7.1.2. INTERROGATION A TRAVERS UNE VUE

Une vue est utilisée dans une commande SELECT de la même manière qu'une table.

Exemple

```
Interrogation de la vue REP_CLI
   select *
   from rep_cli ;
```

7.1.3. MISE A JOUR A TRAVERS UNE VUE

On peut effectuer des opérations d'insertion (INSERT) et de modification (UPDATE) des tables sur lesquelles portent les vues, par leur intermédiaire, sous deux conditions :

- Le SELECT définissant la vue ne comporte pas de jointure,
- Les colonnes résultats du SELECT ne sont pas des expressions.

7.1.4. SUPPRESSION D'UNE VUE

Une vue est détruite par la commande

DROP VIEW nom_vue ;

Exemple

```
Destruction de la vue REP_CLIPILOTE
     drop view rep_cli;
```

7.2. L'INDEX

Lors de la définition d'une table, aucune colonne n'est particularisée comme étant une clé. La création d'un index sur une colonne (un groupe de colonnes) permet :

- La spécification d'une colonne (ou groupe de colonnes) clé primaire : intégrité d'unicité de clé
- L'accélération des procédures d'accès à une table sur les colonnes les plus fréquemment référencés.

7.2.1. CREATION D'UN INDEX

Un index est créé par la commande :

CREATE INDEX nom_index
ON nom_table (nom_colonne1,[,nom_colonne2, …]) ;

Un index est tenu à jour automatiquement par le système en fonction des modifications effectuées sur la table support.

Plusieurs index indépendants peuvent être créés sur la même table.

Les requêtes SQL sont transparentes à la présence (ou absence) d'un index. C'est le système, en fonction des colonnes spécifiés dans l'interrogation, qui choisit ses chemins d'accès.

Exemple

```
Création d'un index sur la ville de client
```

```
   create index ndx_ville
    on client (ville);
```

7.2.2. SUPPRESSION D'UN INDEX

Un index est supprimé par la commande :

DROP INDEX nom_index on table_name ;

Exemple

```
Destruction de l'index sur la ville de client
      drop index ndx_ville on client;
```

8. CONTROLE DES ACCES A LA BASE

Il est permis à plusieurs utilisateurs de travailler sur la même base.
Les commandes SQL 'GRANT' et 'REVOKE' définissent les droits des utilisateurs sur les objets de la base.

Lors de la connexion au système, tout usager doit fournir son identificateur et un mot de passe. Ces informations vont déterminer ses droits d'accès.

Une table appartient à une seule base de données. Une base de données est appelée « schema ».

Un utilisateur peut être autorisé à consulter des tables d'une ou plusieurs schémas.

8.1. ATTRIBUTION DE DROIT

L'instruction SQL 'GRANT' permet au d'attribuer à un utilisateur des droits sur table, une vue ou sur tout un schéma

GRANT autorisation ON niveau_priv
TO nom_utilisateur [,nom_utilisateur,…] ;

Les autorisations possibles sur une table ou vue sont :

- SELECT : droit de lecture,
- INSERT : droit d'insertion,
- UPDATE [(nom_colonne1, …)] : droit de modification pouvant être restreint à certaines colonnes,
- DELETE : droit de suppression,
- ALTER : droit de modification de la structure,
- INDEX : droit de création d'index,
- ALL : ensemble des droits.
-

Et au niveau_priv on trouve :

- *.* : tous les composants de tous les schémas
- db_name.* : tous les composants du schema db_name
- db_name.table_name : sur la table table_name du schema db_name

Exemple

```
Création d'un utilisateur 'toto'@'localhost'
        create user 'toto'@'localhost'
        identified by 'P@ssw0rd';

Accorder l'autorisation de lecture sur la table client à toto
        grant select on GestCom.client to 'toto'@'localhost';
```

```
Accorder toutes les autorisations sur le schéma db1 à toto
       grant all on world.* to 'toto'@'localhost';
```

Pour le système, le nom complet d'une table (d'une vue) est préfixé du nom du schéma propriétaire.

Cela évite toute ambiguïté si des tables de schémas différents portent le même nom..

8.2. RETRAIT DE DROIT

Un utilisateur ayant donné une autorisation peut la reprendre par la commande REVOKE

REVOKE autorisation ON niveau_priv
FROM nom_utilisateur[,nom_utilisateur,…] ;

Exemple

```
Retrait de la permission de sélection sur la table
GestCom.client
revoke select on GestCom.client from 'toto'@'localhost';
```

9. FONCTIONS PREDEFINIES

9.1. FONCTIONS SUR CHAINES

- **LENGTH (ch : CHAR) : NUMBER**

Retour : nombre de caractères de la chaîne *ch*

Ex: LENGTH ('anti-constitutionnel') = 20

- **LOWER (ch : CHAR) : CHAR**

Retour : chaîne *ch* en minuscules

Ex: LOWER ('PAS VU') = 'pas vu'

- **UPPER (ch : CHAR) : CHAR**

Retour : chaîne *ch* en majuscules

Ex: UPPER ('minus') = 'MINUS '

- **LPAD (ch1: CHAR, lg: NUMBER [,ch2 : CHAR]) : CHAR**

Retour : chaîne *ch1* complétée à gauche sur *lg* caractères par la chaîne *ch2* (blanc par défaut)

Ex: LPAD ('PAS PRIS', 15, '*=+') = *=+*=+*PAS PRIS

- **REPLACE (ch1 : CHAR, ch2 : CHAR [,ch3 : CHAR]) : CHAR**

Retour : chaîne *ch1*, chaque occurrence de *ch2* ayant été remplacée par *ch3*. Si *ch3* n'est pas spécifiée, enlève toutes les occurrences de *ch2* dans *ch1*.

Ex: REPLACE ('SA SOUPE','S','L') = 'LA LOUPE'
REPLACE ('EPRISE','E') = 'PRIS'

- **RPAD (ch1 : CHAR, lg : NUMBER[,ch2 : CHAR]) : CHAR**

Retour : chaîne *ch1* complétée à droite sur *lg* caractères par la chaîne *ch2* (blanc par défaut)

Ex: RPAD ('TATARA',12,'TA') = 'TATARATATATA'

- **SUBSTR (ch: CHAR ,n : NUMBER[, lg : NUMBER]) : CHAR**

Retour : extrait de la chaîne *ch* à partir de la position *n*, sur *lg* caractères (ou jusqu'à la fin par défaut)

Ex: SUBSTR ('FORMATION', 4) = 'MATION'
SUBSTR ('FORMATION', 1,3) = 'FOR'

9.2. FONCTIONS NUMERIQUES

- **ABS (n : NUMBER) : NUMBER**

Retour : valeur absolue de n

Ex: ABS (-1) = 1

- **CEIL (n : NUMBER) : NUMBER**

Retour : plus petit entier immédiatement ≥ *n*

Ex: CEIL (123.123) = 124

- **FLOOR (n : NUMBER) : NUMBER**

Retour : plus grand entier immédiatement ≤ *n*

Ex: FLOOR(123.123) = 123

- **MOD (m : NUMBER, n : NUMBER) : NUMBER**

Retour : reste de la division *m* | *n*

Autre syntaxe : m MOD n.

Ex: MOD(10,3) = 1

reste := 10 MOD 3;

- **POWER (n : NUMBER, m : NUMBER) : NUMBER**

Retour : *n* puissance *m*

Ex: POWER(3,4) = 81

- **ROUND (n : NUMBER[,m : NUMBER]) : NUMBER**

Retour : *n* arrondi à la décimale *m*. Si *m* n'est pas spécifié, *n* est arrondi à 0 décimales. Si *m* < 0, *n* est arrondi à gauche du point décimal.

Ex.: ROUND (129.607,1) = 129.7
ROUND (129.607) = 130
ROUND (129.607,-2 = 100

- **SIGN (n : NUMBER) : NUMBER**

Retour : -1 si *n* < 0, 0 si *n* = 0, 1 si *n* > 0

Ex: SIGN(-15) = -1

- **SQRT (n : NUMBER) : NUMBER**

Retour : racine carré de *n* ou NULL si *n* < 0

Ex: SQRT(2307) = 48.0312

- **TRUNC (n : NUMBER[,m : NUMBER]) : NUMBER**

Retour : *n* tronqué à la décimale *m*. Si *m* n'est pas spécifié, *n* est tronqué à 0 décimales. Si *m* < 0, *n* est tronqué à gauche du point décimal.

Ex: ROUND (129.607,1) = 129.6
ROUND (129.607) = 129
ROUND (129.607,-2 = 100

9.3. FONCTIONS SUR DATE

- **DATEDIFF (d : DATE, d : DATE) : NUMBER**

Retour : nombre jours entre les deux dates

Ex: DATEDIFF('2016-08-08','1996-08-08')= 7305

- **DATE_ADD(date, INTERVAL n *uniteDeTemps*)** tel que *uniteDeTemps* prend HOUR,DAY,MONTH ,YEAR…

Retour : la date plus n jours

Ex: DATE_ADD(CURDATE(), INTERVAL 3 DAY)= 2016-07-23

- **DATE_SUB(date, INTERVAL n *uniteDeTemps*)**

Retour : la date plus n jours

Ex: DATE_SUB(CURDATE(), INTERVAL 3 DAY)= 2016-07-17

- **DATE_FORMAT(date, format)**

Retour : la date sous le format indiqué

Ex: DATE_FORMAT(CURDATE(), "%d-%m")= 20-07

- **NOW()**

Retour : la date et l'heure actuelles

Ex: NOW()= 2016-07-20 01:30:33

- **SYSDATE()**

Retour : retourne la date et l'heure courante

Ex: SYSDATE()=2016-07-20 01:32:55

- **CURTIME()**

Retour : retourne l'heure actuelle

Ex: CURTIME()=01:36:15

- **CURDATE()**

Retour : retourne à la colonne de type DATETIME la date actuelle à l'heure 00 :00 :00

Ex: select datecom from commande where numcom=100 ;
2016-07-20 00:00:00

-

9.4. FONCTIONS PARTICULIERES

- **IFNULL (a1: CHAR, a2: CHAR): CHAR**
- **IFNULL (a1: NUMBER, a2: NUMBER): NUMBER**
- **IFNULL (a1: DATE, a2: DATE): DATE**

Retour : si *a1* n'est pas NULL, retourne *a1*, sinon *a2*.

Ex: NVL (a, 'TOTO') retourne 'TOTO', si *a1* = NULL,
Sinon retourne *a2*.

10. DICTIONNAIRE DES DONNEES

Tout SGBD relationnel contient un dictionnaire de données intégré, ensemble de tables (et vues) dans lesquelles sont stockées les descriptions des objets des bases.

Ces tables sont tenues à jour automatiquement par le système et peuvent être consultées par le langage SQL.

Exemple

Lister les tables créées dans la base GestCom.

```
use INFORMATION_SCHEMA –- le dictionnaire
show TABLES;
Desc INFORMATION_SCHEMA.TABLES ;
select TABLE_NAME
            from TABLES
            where TABLE_SCHEMA='GestCom';
```