III-2 Le langage SQL

Le langage SQL (*Structured Query Language*) signifie langage d'interrogation structuré. Il a été créé au début des années 1970 par IBM. C'est une start-up nommée Relational Software qui produira la première version commercialisable en 1979. Cette start-up est depuis devenue Oracle Corp.

Le langage SQL se décompose en plusieurs sous-ensembles :

- Le **DDL** pour *Data Definition Language*, qui regroupe les ordres utilisés pour créer, modifier ou supprimer les structures de la base (tables, index, vues, etc.). Il s'agit principalement des ordres CREATE, ALTER, RENAME et DROP.
- Le **DML** pour *Data Manipulation Language*, qui regroupe les ordres utilisés pour manipuler les données contenues dans la base. Il s'agit principalement des ordres SELECT, INSERT, DELETE et UPDATE.
- Le **DCL** pour *Data Control Language*, qui regroupe les ordres utilisés pour gérer la sécurité des accès aux données. Il s'agit principalement des ordres GRANT et REVOKE.
- Le **TCL** pour *Transaction Control Language*, qui regroupe les ordres utilisés pour gérer la validation ou non des mises à jour effectuées sur la base. Il s'agit principalement des ordres COMMIT et ROLLBACK.

III-2-1 Le langage de définition des données

SQL fournit des instructions permettant de créer, supprimer, modifier, renommer des fichiers. Ces instructions sont les suivantes :

- CREATE
- DROP
- ALTER
- RENAME

1. La création de tables

La création d'une table se réalise avec l'ordre CREATE.

Syntaxe :

```
CREATE TABLE "nom de table"
("colonne 1" "type de données pour la colonne 1",
"colonne 2" "type de données pour la colonne 2",
... );
```

Création de la table Articles.

```
CREATE TABLE Articles
```

```
(NumArt INTEGER NOT NULL,
Désignation CHAR(60) NOT NULL,
Catégorie CHAR(30),
Prix INTEGER);
```

a. Définition de la clé primaire

Dans la table Articles, la clé primaire est NumArt. Voici comment une clé primaire est définie avec SQL :

```
CREATE TABLE Articles
(NumArt INTEGER NOT NULL,
Désignation CHAR(60) NOT NULL,
Catégorie CHAR(30),
Prix INTEGER
Constraint C1 PRIMARY KEY (NumArt));
```

b. Définition des clés étrangères

La table Achats contient deux clés étrangères :

- NumCli
- NumArt

Voici la requête de création de la table Achats :

```
CREATE TABLE Achats
(NumArt INTEGER NOT NULL,
NumArt INTEGER NOT NULL,
Date CHAR(10),
Qté INTEGER,
CONSTRAINT FK1 FOREIGN KEY (NumArt) REFERENCES Articles (NumArt)
CONSTRAINT FK2 FOREIGN KEY (NumCli) REFERENCES Clients (NumCli));
```

2. La suppression physique de tables

La suppression physique de tables se réalise avec l'ordre SQL DROP.

Par exemple :

Suppression de la table mailing.

```
DROP TABLE mailing;
```

3. Modification d'une structure de table

Il peut être nécessaire de modifier la structure d'une table par exemple pour ajouter un champ, redimensionner un champ ou supprimer un champ.

a. Ajouter un champ

Ajouter un champ Mail à la table Clients.

```
ALTER TABLE Clients
   ADD Mail CHAR(40);
```

b. Redimensionner un champ

Agrandir le champ Mail de la table Clients.

```
ALTER TABLE Clients
   MODIFY Mail CHAR(60);
```

c. Supprimer un champ

Supprimer le champ Mail de la table Clients.

```
ALTER TABLE Clients
   DROP COLUMN Mail;
```

d. Supprimer une clé sur une table existante

Il est possible de supprimer a posteriori des clés sur une table existante.

Supprimer la clé sur le champ NumArt de la table Articles.

```
ALTER TABLE Articles
   DROP CONSTRAINT C1;
```

Ajouter une clé primaire sur le champ NumArt de la table Articles.

```
ALTER TABLE Articles
   ADD CONSTRAINT C1 PRIMARY KEY (NumArt);
```

4. Renommer une table

Pour renommer une table, rien de plus simple.

Renommer la table Articles en Produits.

```
RENAME TABLE Articles TO Produits;
```

III-2-2 Le langage de manipulation des données

Nous allons aborder dans cette partie les ordres du langage de manipulation de données. Nous allons commencer par l'ordre SELECT.

Pour les exemples, nous allons utiliser les trois fichiers tirés de ces modèles relationnels :

Clients(**NumCli**, Nom, Prénom, Adresse, Cp, Ville, Téléphone)

Achats(**#NumCli, #NumArt**, Date, Qté)

Articles(**NumArt**, Désignation, Catégorie, Prix)

Clients	NumCli	Nom	Prénom	Adresse	CP	Ville	Téléphone
	1	Auguy	Jean	1 rue droite	30000	Nîmes	0485957575
	2	Baptiste	Jean-Luc	7 rue courbe	12000	Rodez	0565428775
	3	Baptiste	Amandine	Avenue Foch	12000	Rodez	
	4	Collard	Marie-Claire	Rue d'Espagne	66000	Perpignan	
	5	Durand	Raymond	Rue des oliviers	30000	Nîmes	0475145425

Achats	NumCli	NumArt	Date	Qté
	1	1	30/01/2018	1
	1	5	30/01/2018	4
	4	3	29/01/2018	1
	4	2	30/01/2018	2
	5	2	01/02/2018	2
Articles	**NumArt**	**Désignation**	**Catégorie**	**Prix**
	1	Charlie Winston	Cd	12
	2	Caméra Café	Dvd	19
	3	WebCam	Informatique	24

	4	Graveur	Informatique	38
	5	Clé Usb 16G	Informatique	18

1. Sélection des données

La commande SELECT permet de réaliser une lecture d'informations selon certains critères. La commande SELECT retourne les résultats dans un tableau. Cette commande permet de sélectionner une ou plusieurs colonnes d'une ou plusieurs tables.

Syntaxe :

```
SELECT [ALL / DISTINCT] nom_attribut1 [, nom_attribut2, ......]
FROM nom_table1 [, nom_table2, ....]
WHERE <condition de recherche>;
```

L'option **ALL** est l'option par défaut qui permet de sélectionner l'ensemble des lignes satisfaisant à la condition de recherche.

L'option **DISTINCT** permet de ne conserver que des lignes distinctes, en éliminant les doublons.

La **liste des attributs** indique la liste des colonnes choisies, séparées par des virgules. Pour sélectionner l'ensemble des colonnes d'une table, il est possible d'utiliser l'option *.

La **liste des tables** indique l'ensemble des tables (séparées par des virgules) sur lesquelles portent les opérations.

La **condition de recherche** permet d'exprimer des critères de recherche complexes à l'aide d'opérateurs logiques et/ou de comparateurs arithmétiques.

a-La projection

La projection est une opération sur une relation RELATION1 consistant à composer une relation RELATION2 en enlevant à la relation initiale tous les attributs non mentionnés en opérande (aussi bien au niveau du schéma que des tuples) et en éliminant les tuples en double qui ne sont conservés qu'une seule fois.

Un tuple représente une ligne dans un fichier.

Voyons ensemble quelques exemples :

Afficher le contenu de la table Clients.

```
SELECT *
FROM Clients;
```

Résultat renvoyé par la requête :

NumCli	Nom	Prénom	Adresse	CP	Ville	Téléphone
1	Auguy	Jean	1 rue droite	30000	Nîmes	0485957575
2	Baptiste	Jean-Luc	7 rue courbe	12000	Rodez	0565428775
3	Baptiste	Amandine	Avenue Foch	12000	Rodez	
4	Collard	Marie-Claire	Rue d'Espagne	66000	Perpignan	
5	Durand	Raymond	Rue des oliviers	30000	Nîmes	0475145425

Afficher les noms et prénoms des clients.

```
SELECT Nom, Prénom
FROM Clients;
```

Résultat retourné par la requête :

Nom	Prénom
Auguy	Jean
Baptiste	Jean-Luc
Baptiste	Amandine
Collard	Marie-Claire
Durand	Raymond

Utilisation du mot-clé DISTINCT.

Activons cette requête :

```
SELECT Nom
FROM Clients;
```

Comme le mot-clé ALL est actif par défaut, le résultat retourné par la requête sera le suivant :

Nom
Auguy
Baptiste
Baptiste
Collard
Durand

Nous nous rendons compte que le nom Baptiste apparaît deux fois, ce qui dans notre cas n'est d'aucun intérêt.

Voici la requête corrigeant ce problème :

```
SELECT DISTINCT Nom
FROM Clients;
```

Et le résultat retourné :

Nom
Auguy
Baptiste
Collard
Durand

Il est possible de modifier l'affichage d'un nom de colonne en utilisant le mot-clé AS. Par exemple, nous souhaitons modifier l'affichage du Nom par Nom des clients.

Voici la requête :

```
SELECT DISTINCT Nom AS Nom des clients
FROM Clients;
```

et le résultat retourné :

Nom des clients
Auguy
Baptiste
Collard
Durand

La commande DISTINCT évite donc les redondances dans les données retournées par une requête.

b. La restriction

La restriction est une opération sur une relation RELATION1 produisant une relation RELATION2 de même schéma mais comportant les seuls tuples qui vérifient la condition précisée en opérande.

Essayons les requêtes suivantes :

Sélectionner les clients habitant Rodez.

```
SELECT *
FROM Clients
WHERE Ville='Rodez';
```

Lister les articles dont le prix est supérieur à 15 euros.

```
SELECT *
FROM Articles
WHERE Prix>15;
```

Les apostrophes ne servent que pour différencier les valeurs alphanumériques des valeurs numériques.

Dans la clause WHERE les opérateurs suivants peuvent être utilisés :

>	Supérieur
>=	Supérieur ou égal
<	Inférieur

<=	Inférieur ou égal
=	Égal
<>	Différent
AND	Et
OR	Ou
NOT	Pas
IS NULL	Valeur indéterminée
ALL	Tous
ANY	Au moins un
EXISTS	Existence

Lister les articles dont le prix est compris entre 14 et 30 euros.

```
SELECT *
FROM Articles
WHERE prix>14 AND prix <30;
```

Cette requête est correcte, mais manque d'élégance. Il existe d'autres mots-clés permettant d'exprimer certaines contraintes.

IN
BETWEEN
LIKE

Reformulons la requête précédente pour la rendre plus élégante :

```
SELECT *
FROM Articles
WHERE prix BETWEEN 14 AND 30;
```

Afficher les clients dont les noms sont Baptiste et la ville est Rodez.

```
SELECT *
FROM Clients
WHERE nom='Baptiste'
AND ville='Rodez';
```

Afficher les clients dont les noms sont Baptiste ou la ville est Rodez.

```
SELECT *
FROM Clients
WHERE nom='Baptiste'
OR ville='Rodez';
```

Afficher les clients dont les numéros de téléphone ont été saisis.

```
SELECT *
FROM Clients
WHERE Téléphone IS NOT NULL;
```

Afficher les clients dont les numéros de téléphone n'ont pas été saisis.

```
SELECT *
FROM Clients
WHERE Téléphone IS NULL;
```

Afficher les clients qui habitent les villes de Rodez, Aurillac ou Tarbes.

```
SELECT *
FROM Clients
WHERE Ville IN ('Rodez', 'Aurillac', 'Tarbes');
```

Afficher les clients dont les noms commencent par B.

```
SELECT *
FROM Clients
WHERE Nom LIKE 'B%';
```

Afficher les clients dont les noms ne commencent pas par B.

```
SELECT *
FROM Clients
WHERE Nom NOT LIKE 'B%';
```

Afficher les clients dont les noms finissent par B.

```
SELECT *
FROM Clients
WHERE Nom LIKE '%B';
```

Afficher les clients dont les noms contiennent un B.

```
SELECT *
FROM Clients
WHERE Nom LIKE '%B%';
```

c. Les tris

Avec SQL il est possible d'effectuer des tris selon différents critères grâce à la clause ORDER BY et aux mots-clés ASC, DESC. Par défaut le tri est par ordre croissant.

Afficher les noms et prénoms des clients triés par noms croissants.

```
SELECT *
FROM Clients
ORDER BY Nom ASC;
```

Afficher les noms et prénoms des clients triés par noms décroissants.

```
SELECT *
FROM Clients
ORDER BY Nom DESC;
```

d. Les jointures

La jointure est l'opération consistant à rapprocher selon une condition les tuples de deux relations RELATION1 et RELATION2 afin de former une troisième relation RELATION3 qui contient l'ensemble de tous les tuples obtenus en concaténant un tuple de RELATION1 et un tuple de RELATION2 vérifiant la condition de rapprochement.

Afficher les noms et prénoms des clients ayant acheté un article le 27 février 2018.

```
SELECT Nom, Prénom
FROM Clients, Achats
WHERE Clients.NumCli=Achats.Numcli
AND Achats.Date='27/02/2018';
```

Afficher le nom, le prénom des clients ainsi que la quantité et désignation des produits achetés.

```
SELECT Nom, Prénom, Qté, Désignation
FROM Clients, Achats, Articles
```

```
WHERE Clients.NumCli=Achats.Numcli
AND Achats.NumArt=Articles.NumArt;
```

Les noms de tables peuvent devenir fastidieux à saisir, on peut simplifier l'écriture en utilisant le mot-clé AS.

Afficher le nom, le prénom des clients ainsi que la quantité et désignation des produits achetés.

```
SELECT Cl.Nom, Cl.Prénom, Ac.Qté, Ar.Désignation
FROM Clients AS Cl, Achats AS Ac, Articles AS Ar
WHERE Cl.NumCli=Ac.Numcli
AND Ac.NumArt=Ar.NumArt;
```

Ces jointures recherchent l'égalité des valeurs entre les éléments des deux champs. Les valeurs seront sélectionnées si elles existent dans les deux tables. Dans l'exemple du dessus ne seront affichés que les clients ayant acheté un produit.

Pour des raisons particulières, il peut être envisagé de vouloir faire afficher l'ensemble des clients, qu'ils aient ou pas acheté un produit.

Cette nouvelle possibilité est apparue avec SQL2 avec des opérateurs de jointure particuliers.

La jointure naturelle

La jointure naturelle est une façon de faire confiance dans le compilateur SQL en lui laissant le soin de trouver les colonnes sur lesquelles faire la jointure. Par exemple :

Afficher les noms et prénoms des clients ayant acheté un article le 27 février 2018.

```
SELECT Nom, Prénom
FROM Clients NATURAL JOIN ACHAT
WHERE Achats.Date='27/02/2018';
```

Ici, le compilateur SQL va utiliser la colonne Numcli, commune aux deux tables, pour faire la jointure. Nous pouvons forcer la reconnaissance en utilisant le mot-clé USING. Par exemple :

Afficher les noms et prénoms des clients ayant acheté un article le 27 février 2018.

```
SELECT Nom, Prénom
FROM Clients NATURAL JOIN ACHAT USING (Numcli)
WHERE Achats.Date='27/02/2018';
```

Attention tout de même à vérifier que cette syntaxe est acceptée par votre compilateur SQL.

La jointure interne

C'est le type de jointure le plus usité. Voici son gabarit :

```
SELECT ...
FROM <Table gauche>
[INNER]JOIN <Table droite>
ON <Conditions de jointure>;
```

Exemple :

```
SELECT Nom, Prénom
FROM Clients INNER JOIN ACHAT
On Clients.NumCli=Achats.Numcli
WHERE Achats.Date='27/02/2018';
```

Il est à noter que le mot-clé INNER est facultatif. Si le compilateur SQL ne le détecte pas, il réalise par défaut une jointure interne.

La commande INNER JOINT est très commune.

Soit l'exemple suivant :

```
SELECT Nom, Prénom
FROM Clients INNER JOIN ACHAT
On Clients.NumCli=Achats.Numcli
```

La requête indique qu'il faut sélectionner les enregistrements des tables Clients et Achats lorsque les valeurs des champs Clients.Numcli et Achats.Numcli sont égales.

La jointure externe

Ce type de jointure permet d'afficher un ensemble d'informations plus important. En effet, imaginons vouloir imprimer l'ensemble des clients et même ceux n'ayant pas d'achat à la date du 27 février 2018. Voici le type de requête à employer :

```
SELECT Nom, Prénom, Total
FROM Clients
LEFT OUTER JOIN ACHAT
On Clients.NumCli=Achats.Numcli
WHERE Achats.Date='27/02/2018';
```

La clause LEFT permet de fixer la table de référence qui sera intégralement listée ; ici, LEFT indique la table Clients car elle est à gauche du mot-clé JOIN. Cette commande permet donc de lister tous les résultats de la table de gauche (LEFT) **même** s'il n'y a pas de correspondance dans la deuxième table.

Une représentation de la table pourrait être :

Nom	Prénom	Total
Baptiste	Jean-Luc	54
Martinez	Manuel	24
Durand	Lucien	NULL
Garcia	Patrick	66

Les quatre clients sont extraits et ceux n'ayant pas une existence dans la table Achat via leur Numcli se voient affecter la valeur NULL, qui signifie « indéterminé ».

Bien entendu, il est possible de remplacer LEFT par RIGHT si cela est nécessaire.

Pour aller plus loin sur les jointures et sur SQL en général, nous vous conseillons l'excellent site http://sqlpro.developpez.com/

e. Les fonctions statistiques

SQL offre cinq fonctions mathématiques standard :

Fonctions	Description
AVG(attribut)	Calcule la moyenne des valeurs dans l'attribut.
SUM(attribut)	Calcule la somme des valeurs dans l'attribut.
MIN(attribut)	Détermine la plus petite valeur dans l'attribut.
MAX(attribut)	Détermine la plus grande valeur dans l'attribut.
COUNT(attribut)	Compte le nombre d'occurrences dans l'attribut.

Calculer le prix moyen des articles.

```
SELECT AVG(Prix) AS Prix Moyen
FROM Articles;
```

Calculer le prix moyen des articles, afficher le prix minimum et le prix maximum.

```
SELECT AVG(Prix) AS Prix Moyen, MIN(Prix), MAX(Prix)
FROM Articles;
```

Afficher la somme de toutes les quantités achetées.

```
SELECT SUM(Qté) AS Quantité
FROM Achats;
```

Compter le nombre de catégories.

```
SELECT COUNT(Catégorie) AS Nombre de catégorie
FROM Articles;
```

Le problème avec cette requête c'est qu'elle va retourner le nombre total de lignes dans la colonne catégorie (5 dans la table exemple), mais pas le nombre de catégories uniques (3 dans la table exemple). Voici comment corriger ce problème.

Compter le nombre de catégories sans doublons.

```
SELECT COUNT(DISTINCT Catégorie) AS Nombre de catégorie
FROM Articles;
```

Faire la somme des quantités totales des achats réalisés pour l'article numéro 3.

```
SELECT SUM(Qté) AS Quantité totale
FROM Achats
WHERE NumArt=3;
```

En dehors de ces fonctions, il est possible aussi d'effectuer des calculs dans les requêtes. Imaginons que nous désirions afficher les prix augmentés de 10 %. Voici une façon de l'écrire :

```
SELECT Désignation, Prix*1,10 AS Prix augmenté
FROM Articles;
```

La valeur n'est pas modifiée dans le fichier, la valeur est calculée juste pour l'affichage.

f. Les opérations portant sur des ensembles

L'algèbre relationnelle permet l'utilisation d'opérateurs ensemblistes. Il en existe trois :

- UNION
- INTERSECT
- EXCEPT

Les mots-clés INTERSECT et EXCEPT n'étant pas normalisés dans SQL, ils risquent de ne pas être fonctionnels dans le SGBD. Le mot-clé EXCEPT peut être remplacé par NOT EXISTS et INTERSECT par EXISTS.

Le mot-clé UNION retourne l'ensemble des tuples appartenant aux relations de la requête.

Par exemple :

Afficher les numéros des articles dont le prix est supérieur à 20 euros ainsi que les numéros des articles achetés par le client numéro 1.

```
SELECT NumArt
FROM Articles
WHERE Prix > 20
UNION
SELECT NumArt
FROM Achats
WHERE NumCli=1;
```

g. Les regroupements

Les regroupements permettent de créer des sous-ensembles d'occurrences. Une seule ligne regroupe ainsi les valeurs identiques en fonction de l'attribut spécifié.

Imaginons que dans le fichier achat nous souhaitons avoir le total des prix par catégorie. Par exemple ceci :

Catégorie	**Prix**
Cd	12
Dvd	19
Informatique	80

La requête permettant ce regroupement s'exprime en utilisant la clause GROUP BY.

Effectuer le regroupement des articles par catégorie et cumuler les prix.

```
SELECT Catégorie, Sum(Prix) AS Prix cumulés
FROM Articles
GROUP BY Catégorie;
```

Calculer le prix de vente moyen par catégorie.

```
SELECT Catégorie, AVG(Prix) AS Prix moyen
FROM Articles
GROUP BY Catégorie;
```

Pour chaque client, afficher le nombre d'achats et le montant cumulé des achats.

```
SELECT Clients.Nom, Clients.Prénom, Count(*) AS Nbre,
Sum(Articles.prix*Achats.Qté) As Total
FROM Achats, Clients, Articles
WHERE Achats.NumCli=Clients.NumCli
AND Achats.NumArt=Articles.NumArt
GROUP BY Clients.Nom, Clients.Prénom;
```

Calculer le prix de vente moyen des articles par catégorie, dont le prix de vente est inférieur ou égal à 15.

```
SELECT Catégorie, AVG(Prix)
FROM Articles
WHERE Prix <= 15
GROUP BY Catégorie;
```

Lorsque l'on souhaite réaliser des restrictions sur la clause de regroupement, il faut utiliser la clause HAVING.

Afficher les numéros de clients ayant plus de 1 achat.

```
SELECT NumCli, count(*) AS Nbr de commande
FROM Achats
GROUP BY NumCli
HAVING COUNT(*)>1;
```

Afficher les clients dont le prix moyen d'achat des articles est supérieur à 10 euros dans l'ordre croissant des noms.

```
SELECT Clients.nom, AVG(Articles.prix)
FROM articles, Achats, Clients
WHERE Clients.NumCli = Achats.NumCli
AND Achats.NumArt=Articles.NumArt
GROUP BY Clients.nom
HAVING AVG(Articles.Prix)>10
ORDER BY Clients.nom;
```

h. Les sous-requêtes

Les sous-requêtes, appelées aussi requêtes imbriquées, permettent de réaliser de façon élégante des traitements qui pourraient s'avérer fastidieux voire difficilement réalisables à l'aide de plusieurs requêtes simples. Pour simplifier, le rôle d'une sous-requête est d'envoyer le résultat de son traitement dans la requête principale. Voyons grâce à un exemple.

Lister les achats du client « Auguy »

```
SELECT NumArt, Date, Qté
FROM Achats
WHERE NumCli = (SELECT NumCli
        FROM Clients
        Where Nom = 'Auguy');
```

La sous-requête renvoie le numéro du client recherché à la requête principale.

Nous aurions pu, pour cet exemple, traiter la requête de cette façon à l'aide d'une jointure :

```
SELECT Achats.NumArt, Achats.Date, Achats.Qté
FROM Achats, Clients
WHERE Clients.NumCli=Achats.NumCli
And Clients.Nom='Auguy';
```

Mais si nous compliquons les exemples, nous allons nous rendre compte que les sous-requêtes deviennent de bonnes alliées.

Par exemple :

Nous désirons connaître les articles de prix supérieur au prix moyen de tous les articles.

```
SELECT *
FROM Articles
WHERE Prix >    (SELECT AVG(PRIX)
        FROM Articles);
```

La sous-requête calcule le prix moyen et le retourne à la requête principale qui peut ainsi effectuer son travail de recherche des prix supérieurs au prix moyen.

Maintenant, observons la puissance de sous-requêtes.

Rechercher les clients habitant la même ville que le client numéro 2 et ayant acheté des articles de prix supérieur à 15 euros.

```
SELECT Nom, Prénom
```

```
FROM Clients, Achats
WHERE Clients.Ville = (SELECT Ville
                       FROM Clients
                       WHERE NumCli=2)
AND Clients.NumCli=Achats.NumCli
AND Achats.NumArt IN (SELECT NumArt
                      FROM Articles
                      WHERE Prix>15);
```

Observons la deuxième sous-requête. Le mot-clé IN permet de faire rechercher un numéro d'article dans un ensemble de numéros renvoyés par la sous-requête.

Afficher les numéros de clients ayant acheté un produit en quantité supérieure à chacun des produits achetés par le client numéro 1.

```
SELECT DISTINCT NumCli
FROM Achats
WHERE Qté > ALL
      (SELECT Qté
       FROM Achats
       WHERE NumCli=1);
```

Le mot-clé ALL signifie que Qté va être testé avec l'ensemble des quantités renvoyées par la sous-requête.

Donner la liste des articles dont le prix de vente est supérieur au prix de vente de tous les articles dont la catégorie est CD.

```
SELECT *
FROM articles
WHERE Prix > ALL    (SELECT Prix
                     FROM articles
                     WHERE Catégorie= 'Cd');
```

Une autre façon d'écrire cette requête aurait pu être la suivante :

```
SELECT *
FROM articles
WHERE Prix > (SELECT MAX(Prix)
      FROM articles
```

```
        WHERE Catégorie= 'Cd');
```

Utilisation du mot-clé EXISTS

Le mot-clé EXISTS permet de tester que la sous-requête renvoie un résultat :

Afficher la liste de tous les clients si l'un d'eux habite Aurillac.

```
SELECT NumCli, Nom, Prénom
FROM Clients
WHERE EXISTS (Select *
        FROM Clients
        WHERE Ville='Aurillac');
```

Ou l'inverse :

Afficher la liste de tous les clients si aucun d'eux n'habite Aurillac.

```
SELECT NumCli, Nom, Prénom
FROM Clients
WHERE NOT EXISTS (Select *
        FROM Clients
        WHERE Ville='Aurillac');
```

2. L'insertion des données

a. Insertion simple

Le mot-clé INSERT permet l'ajout d'enregistrements dans les fichiers.

Syntaxe :

```
INSERT INTO "nom de table" ("colonne 1", "colonne 2", ...)
VALUES ("valeur 1", "valeur 2", ...);
```

Ajouter un article.

```
INSERT INTO Articles(NumArt,Désignation,Catégorie,Prix)
VALUES(6,'Pocket Pc HP','Pda',175);
```

Une autre façon d'écrire la requête est la suivante :

```
INSERT INTO Articles
VALUES(6,'Pocket Pc HP','Pda',175);
```

Cette dernière requête est fonctionnelle, car l'ensemble des champs est renseigné. Le nombre de valeurs de la requête doit coïncider avec le nombre de champs de destination ; ainsi la requête suivante va retourner une erreur :

```
INSERT INTO Articles
VALUES(7,'Pocket Pc HP');
```

Lorsque certaines valeurs ne sont pas connues à l'exécution de la requête, il est possible de la compléter avec le mot-clé NULL :

```
INSERT INTO Articles
VALUES(8,'Pocket Pc HP',NULL,NULL);
```

ou encore il est possible de tout préfixer :

```
INSERT INTO Articles( NumArt,Désignation)
VALUES(9,'Pocket Pc');
```

b. Insertion en masse

Il est possible d'insérer dans une table un ensemble d'enregistrements. Par exemple, imaginons que nous souhaitons envoyer un mailing ciblé sur les clients habitant la ville de Rodez. Un fichier nommé mailing de même structure que le fichier Clients est créé. Voici la requête qui permet de copier les clients habitant Rodez dans le fichier mailing.

```
INSERT INTO mailing
  SELECT *
  FROM Clients
  WHERE Ville = 'Rodez';
```

3. La modification des données

SQL permet la modification des données grâce au mot-clé UPDATE.

Syntaxe :

```
UPDATE "nom de table"
SET "colonne 1" = [nouvelle valeur]
WHERE {condition};
```

Augmenter tous les prix de 15 %.

```
UPDATE Articles
SET Prix = Prix*1.15;
```

Augmenter de 5 % les articles de la catégorie Informatique.

```
UPDATE Articles
SET Prix = Prix*1.15
WHERE Catégorie='Informatique';
```

4. La suppression des données

La suppression des données se réalise avec le mot-clé DELETE.

Syntaxe :

```
DELETE FROM "nom de table"
WHERE {condition};
```

Supprimer l'ensemble des lignes de la table Clients.

```
DELETE *
FROM Clients;
```

Supprimer les clients habitant Rodez.

```
DELETE
FROM Clients
WHERE Ville='Rodez';
```

III-2-3 Le langage de contrôle des données

Le langage de contrôle des données comprend deux ordres :

- GRANT
- REVOKE

GRANT permet de donner des droits à un utilisateur sur une base de données, REVOKE supprime des droits acquis.

1. L'ordre GRANT

Cet ordre utilise différentes options pour définir au mieux les droits.

Option	Explication
ALTER	Donne le droit de modifier la structure d'une table.
DELETE	Donne le droit de supprimer des enregistrements.
INSERT	Donne le droit d'insérer des enregistrements dans une table.
SELECT	Donne le droit d'exécuter des requêtes de sélection.

UPDATE	Donne le droit de modifier les données d'une table.
ALL	Donne tous les droits.

Donner le droit à l'utilisateur Jean-Luc de modifier la structure de la table Clients.

```
GRANT ALTER
  ON Clients
  TO Jean-Luc;
```

Donner le droit à l'utilisateur Jean-Luc d'afficher le contenu de la table Clients.

```
GRANT SELECT
  ON Clients
  TO Jean-Luc;
```

Donner tous les droits à l'utilisateur Jean-Luc sur la table Clients.

```
GRANT ALL
  ON Clients
  TO Jean-Luc;
```

Permettre à Jean-Luc de lire, modifier, insérer dans la table Clients.

```
GRANT SELECT, UPDATE,INSERT
  ON Clients
  TO Jean-Luc;
```

Permettre à Jean-Luc de modifier seulement les prix et les catégories de la table Articles.

```
GRANT UPDATE(Catégorie, Prix)
  ON Articles
  TO Jean-Luc;
```

Permettre à tout le monde d'afficher le contenu de la table Clients.

```
GRANT SELECT
  ON Clients
  TO PUBLIC;
```

Donner le droit à Jean-Luc de faire des sélections dans la table Articles et de pouvoir redistribuer ce droit à d'autres utilisateurs.

```
GRANT SELECT
```

```
    ON Articles
    TO Jean-Luc
    WITH GRANT OPTION;
```

2. L'ordre REVOKE

L'ordre REVOKE permet de reprendre tous les droits accordés aux utilisateurs.

Enlever le droit donné à Jean-Luc d'afficher la table Clients.

```
REVOKE SELECT
    ON Clients
    FROM Jean-Luc;
```

Enlever tous les droits donnés sur la table Clients à tous les utilisateurs.

```
REVOKE ALL
    ON Clients
    FROM PUBLIC;
```