Le langage SQL

BD pivot

Tout au long de cette partie, nous allons utiliser la BD suivante:

Station (nomStation, capacité, lieu, région, tarif)
Activité (#nomStation, libellé, prix)
Client (id, nom, prénom, ville, région, solde)
Séjour (#idClient, #station, début, nbPlaces)

Station

NomStation	capacité	lieu	région	tarif
Venusa	350	Guadeloupe	Antilles	1200
Santalba	150	Martinique	Antilles	2000
Passac	400	Alpes	Europe	1000

Activité

NomStation	libellé	prix
Venusa	Voile	150
Venusa	Plongée	120
Passac	Ski	200
Passac	Piscine	20
Santalba	Kayac	50

Séjour

idClient	station	début	nbPlaces
10	Passac	01/07/98	2
30	Santalba	14/08/00	4
20	Venusa	12/12/02	3
30	Pasac	311203	6

Client

id	nom	prénom	ville	région	solde
10	Fogg	Phileas	Londres	Europe	12465
20	Pascal	Blaise	Paris	Europe	6763
30	Kerouac	Jack	New York	Amérique	9812

Introduction à SQL

Langage déclaratif qui permet de faire des opérations (consultation, modification,etc) sur une BD sans se soucier de la représentation interne (physique) des datas, de leur localisation et des chemins d'accès

SQL ne permet pas de faire de la programmation et doit donc être associé avec un langage comme le C ou JAVA pour réaliser des traitements complexes accédant à une BD

Nous allons étudier la variante du langage SQL pour Oracle

SQL

SQL se subdivise en 5 sous- langages:

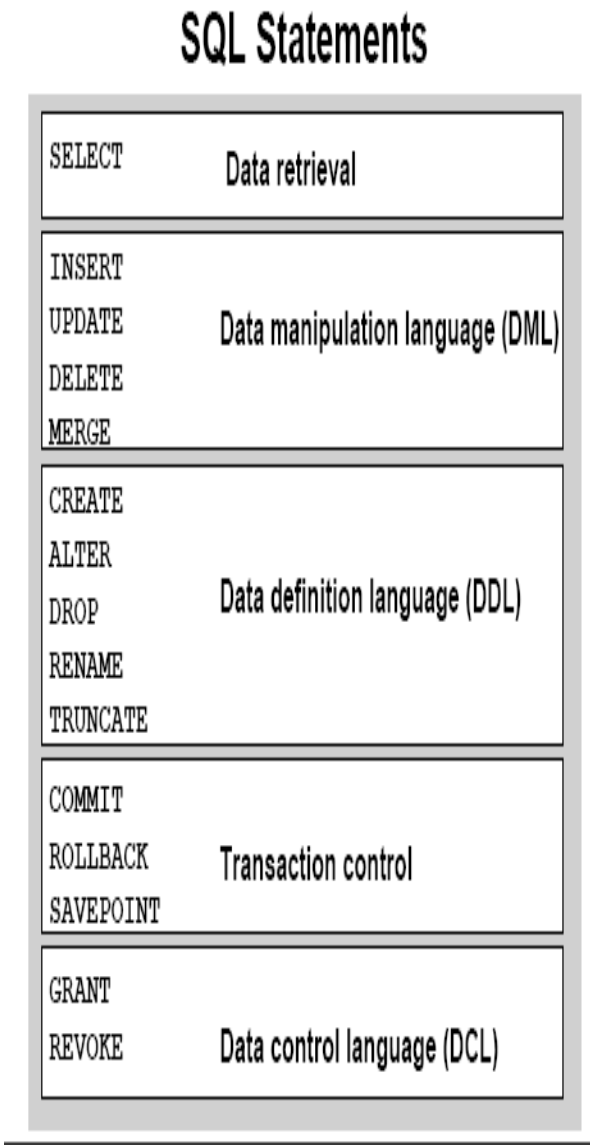
Langage d'interrogation de la base : permet de consulter les datas de la BD

Langage de Manipulation des Datas (LMD): ajout, modification et suppression de la BD

Langage de Définition des Datas (LDD) : création, modification et suppression des structures des différents objets de la BD

Langage de Contrôle des Transactions

Langage de Contrôle de Datas (LCD) : gestion des protections d'accès



LDD – Création de table

```
SQL>   CREATE TABLE nomTable (
         attribut1 Type1 [NOT NULL | NULL],
         ….,
         attributN  TypeN  [DEFAULT value] ) ;
```

[ ]: Tous les paramètres qui sont entre crochets sont facultatifs

Les types autorisés:

Data Type	Description
VARCHAR2(*size*)	Variable-length character data
CHAR[(*size*)]	Fixed-length character data
NUMBER[(*p*,*s*)]	Variable-length numeric data
DATE	Date and time values
LONG	Variable-length character data up to 2 gigabytes
CLOB	Character data up to 4 gigabytes
RAW and LONG RAW	Raw binary data
BLOB	Binary data up to 4 gigabytes
BFILE	Binary data stored in an external file; up to 4 gigabytes

Création de table- Contraintes d'intégrité…

Pour assurer le maintien de la cohérence des données de la base, nous devons définir des contraintes d'intégrité

Les contraintes d'intégrité sont:

- **Primary key**
- **Foreing key**
- **Unique**
- **Check**
- **NOTNULL**

Les mots clés de la contrainte Foreing key

Foreign key : défini la colonne qui sera considérée comme FK
REFERENCES: permet d'identifier la table et la colonne dans la table maître
ON DELETE CASCADE: supprime les enregistrements de la table esclave associés à l'enregistrement de la table maître supprimé
ON DELETE SET NULL : remet à null les valeurs de la FK

Création de table- Contraintes d'intégrité

Les contraintes sont définies

Lors de la création de la table (CREATE TABLE) au niveau colonne ou au niveau table

Ou lors d'une modification de la structure du table (ALTER TABLE)

Contraintes d'intégrité définies au niveau colonne …

```
SQL>   CREATE TABLE nomTable (
          attribut1 Type1 PRIMARY KEY,
          attribut2 Type2 [NOT NULL | NULL],
          attribut3 Type3 CHECK(condition),
          attribut4 Type4 UNIQUE,
          ….,
          attributN  TypeN  FOREIGN KEY REFERENCES
          nomTable2(attribut21) ON DELETE CASCADE) ;
```

Exemple1 : Station (<u>nomStation</u>, capacité, lieu, région, tarif)

```
SQL>   CREATE TABLE Station (
          nomStation varchar(15) PRIMARY KEY,
          capacité number(4),
          lieu varchar2(20) NOT NULL,
          région varchar2(30) CHECK(région in ('Antilles','Europe')),
          tarif real DEFAULT 0) ;
```

Contraintes d'intégrité définies au niveau colonne

Exemple2 : Activité (#nomStation, libellé, prix)

```
SQL>   CREATE TABLE Activité (
          nomStation varchar2(15)  PRIMARY KEY,
          libellé varchar2(15) PRIMARY KEY,
          prix number(4) NOT NULL
         ) ;
```

Cet ordre SQL est-il correct?

Cet ordre n'est pas correct car on ne peut pas définir 2 clés primaires sur la même table

Comme notre clé est considérée comme une clé composée alors on doit définir la contrainte au niveau table

Contraintes d'intégrité définies au niveau table

```
SQL>   CREATE TABLE nomTable (
          attribut1 Type1,
          attribut2 Type2 [NOT NULL | NULL],
          ….,
          attributN  TypeN ,
          CONSTRAINT nom_contrainte PRIMARY KEY(attribut1),
          CONSTRAINT nom_contrainte CHECK(condition),
          CONSTRAINT NomContrainte FOREIGN KEY (attributN)
          REFERENCES nomTable2(attribut21) ON DELETE  CASCADE) ;
```

Solution de l'exemple2 : Activité (#nomStation, libellé, prix)

```
SQL>  CREATE TABLE Activité (
           nomStation varchar2(15),
           libellé varchar2(15),
           prix number(4) NOT NULL,
          CONSTRAINT activité_PK PRIMARY KEY(nomStation,libellé),
          CONSTRAINT activité_FK FOREIGN KEY (nomStation) REFERENCES
          Station(nomStation) ON DELETE CASCADE) ;
```

Contraintes d'intégrité définies après la création de la table

Ajout de contrainte

```
ALTER TABLE table
ADD [CONSTRAINT constraint] type (column);
```

SQL> **ALTER TABLE Activité**
ADD CONSTRAINT activité_FK FOREIGN KEY (nomStation)
REFERENCES Station(nomStation) ;

Suppression de contrainte

SQL> **ALTER TABLE table**
DROP PRIMARY KEY | UNIQUE(column) | CONSTRAINT nom_contrainte [CASCADE];

SQL> **ALTER TABLE Activité**
DROP CONSTRAINT activité_FK ;

ALTER TABLE

Outre la possibilité d'ajouter ou de supprimer les contraintes, l'ordre ALTER TABLE permet:

- **Ajouter une nouvelle colonne**
- **Modifier une colonne existante**
- **Définir une valeur par défaut pour une colonne**
- **Supprimer des colonnes**

```
ALTER TABLE table
ADD         (column datatype [DEFAULT expr]
            [, column datatype]...);
```

```
ALTER TABLE table
MODIFY      (column datatype [DEFAULT expr]
            [, column datatype]...);
```

```
ALTER TABLE table
DROP       (column);
```

Suppression de table

Suppression d'une table

- les données et la structure sont supprimées
- la transaction en cours sera validée (commit implicite)
- les indexes sont supprimés
- l'ordre ne peut pas être annulé

Syntaxe

```
SQL>  DROP TABLE nom_table ;
```

Changement de nom d'une table

- pour pouvoir renommer l'objet, on doit être son propriétaire

Syntaxe

```
SQL>  RENAME ancien_nom_table TO nouveau_nom_table;
```

TRUNCATE TABLE

L'ordre TRUNCATE permet de

- **supprimer les données d'une table**
- **libérer l'espace associé**
- **l'ordre ne peut pas être annulé**

Syntaxe

```
SQL>   TRUNCATE TABLE nom_table ;
```

Résumé

Statement	Description
CREATE TABLE	Creates a table
ALTER TABLE	Modifies table structures
DROP TABLE	Removes the rows and table structure
RENAME	Changes the name of a table, view, sequence, or synonym
TRUNCATE	Removes all rows from a table and releases the storage space

LDD – Pourquoi on utilise les vues?

Le concept de vue permet d'avoir une vision logique des datas contenues dans une ou plusieurs tables

Vue ou table virtuelle n'a pas d'existence propre; aucune donnée ne lui est associée

C'est juste la description de cette vue qui est stockée

On utilise les vues pour une des raisons suivantes:

- **Limiter l'accès aux datas**
- **Remplace le codage de requêtes complexes**

Deux types de vues: simples et complexes

Feature	Simple Views	Complex Views
Number of tables	One	One or more
Contain functions	No	Yes
Contain groups of data	No	Yes
DML operations through a view	Yes	Not always

LDD – Exemple de création des vues

Vue constituant une restriction de la table Client aux clients qui habitent Paris

```
SQL>  CREATE OR REPLACE VIEW Client_vue
          AS SELECT *
              FROM Client
              WHERE ville='Paris';
```

Syntaxe de Suppression d'une vue

```
SQL>   DROP VIEW nom_vue ;
```

Syntaxe de changement de nom d'une vue

```
SQL>   RENAME ancien_nom_vue TO nouveau_nom_vue;
```

LDD – Les utilisateurs

Tout accès à la base s'effectue par la notion d'utilisateur (compte ORACLE)

Avant de pouvoir créer les objets (table, vue, index, etc…), nous devons créer pour chaque utilisateur un compte dans lequel il va gérer ces différents objets

Chaque USER est défini par:

Un nom utilisateur, Un mot de passe, Un ensemble de privilèges, Un profil

LDD – Syntaxe de création d'un user

```
SQL>  CREATE USER nom_user
        IDENTIFIED {BY password | EXTERNALLY}
        [DEFAULT TABLESPACE nomTablespaceD]
        [TEMPORARY TABLESPACE nomTablespaceT ]
        [ QUOTA {entier [K|M] | UNLIMITED} ON nomTablespace]
        [ PASSWORD EXPIRE ]
        [ ACCOUNT {LOCK | UNLOCK} ]
        [ PROFILE {nomProfil | DEFAULT} ] ;
```

UNLIMITED: permet de spécifier que les objets d'un user peuvent utiliser autant d'espace qu'il y en a dans le tablespace

PASSWORD EXPIRE: oblige l'user à réinitialiser le password lorsqu'il se connecte à la BD par l'intermédiaire de SQL*PLUS (valable juste lors de l'authentification par le Serveur Oracle)

ACCOUNT {LOCK | UNLOCK}: verrouiller/déverrouiller explicitement le compte user

LDD – Exemple de création d'un user

Créer un compte Oracle dans le nom d'utilisateur est TSI3 et son password est TSI3PASS

```
SQL>  CREATE USER TSI3
      IDENTIFIED BY TSI3PASS ;
```

Créer un compte Oracle dans le nom d'utilisateur est TSI3, son password est TSI3PASS et qui sera bloqué

```
SQL>  CREATE USER TSI3
      IDENTIFIED BY TSI3PASS
      ACCOUNT LOCK ;
```

Accès aux datas - SELECT

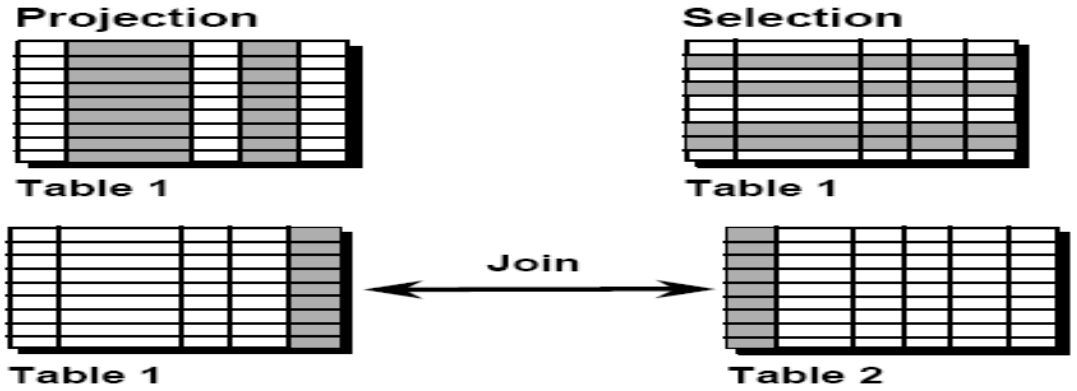
Un ordre SELECT permet d'extraire des informations d'une BD

L'utilisation d'un ordre SELECT offre les possibilités suivantes :

Sélection : SQL permet de choisir dans une table, les lignes que l'on souhaite ramener au moyen d'une requête. Divers critères de sélection sont disponibles à cet effet

Projection : SQL permet de choisir dans une table, les colonnes que l'on souhaite ramener au moyen d'une requête. Vous pouvez déterminer autant de colonnes que vous le souhaitez

Jointure : SQL permet de joindre des données stockées dans différentes tables, en créant un lien par le biais d'une colonne commune à chacune des tables



Basic SELECT

```
SELECT    *|{[DISTINCT] column|expression [alias],...}
FROM      table;
```

Pour restreindre les lignes, on utilise la clause Where

```
SELECT    *|{[DISTINCT] column|expression [alias],...}
FROM      table
[WHERE    condition(s)];
```

Exemple 2 :

on souhaite extraire de la base les libellés des activités offerts par la station Santalba et dont le prix est compris entre 50$ et 120$

Requête 2 :

```
SQL>  SELECT nomStation,libelle
        FROM Activite
        WHERE nomStation = 'Santalba'
        AND (prix > 50 AND prix < 120);

SQL>  SELECT nomStation,libelle
        FROM Activite
        WHERE nomStation = 'Santalba'
        AND prix BETWEEN 50 AND 120;
```

≡

Basic SELECT

Exemple 3_1 :

Quels sont les stations dont on ne connaît pas leurs emplacements

Requête 3_1 :

```
SQL>  SELECT nomStation
      FROM Station
      WHERE region IS NULL;
```

Exemple 3_2 :

Quels sont les stations dont on connaît leurs emplacements

Requête 3_2 :

```
SQL>  SELECT nomStation
      FROM Station
      WHERE region IS NOT NULL;
```

Basic SELECT

Remarques :

Les opérateurs de comparaison sont ceux du Pascal : <, >, =, <=, >=, !=

Il faut être attentif aux différences entre chaînes de longueur fixe et chaînes de longueur variable. Les premières sont complétées par des blancs (’ ’) et pas les secondes

Si SQL ne distingue pas majuscules et minuscules pour les mot-clés, il n’en va pas de même pour les valeurs. Donc ’SANTALBA’ est différent de ’Santalba’

Basic SELECT

Les patterns matching (les recherches par motif) :

SQL fournit des options pour les recherches par motif à l'aide de la clause LIKE

Le caractère '_' désigne n'importe quel caractère, et le '%' n'importe quelle chaîne de caractères

Exemple 4 :

Chercher toutes les stations dont le nom termine par un 'a'

Requête 4 :

```
SQL>  SELECT nomStation
        FROM Station
        WHERE nomStation  LIKE  '%a';
```

Tri des données

```
SELECT      *|{[DISTINCT] column|expression [alias],...}
FROM        table
[WHERE      condition(s)]
[ORDER BY   {column, expr, alias} [ASC|DESC]];
```

Jointure – Requêtes sur plusieurs tables

En extrayant des données provenant de plusieurs tables, nous devons les joindre. Plusieurs types de jointures sont définis selon la nature de la condition:

une équi jointure ou jointure : permet de réaliser une liaison logique entre 2 tables (l'égalité entre la CP d'une table et la CE de l'autre)

une inéquijointure ou thêtajointure : est une jointure dont l'expression du pivot utilise des opérateurs autre que l'égalité(<,>,!=,>=,<=, BETWEEN)

Equi jointure

Exemple 7: **Donner le nom des clients avec le nom des stations où ils ont séjourné**

Requête 7:
```
SQL>  SELECT nom, station
      FROM  Client, Sejour
      WHERE id = idClient ;
```

Exemple 8: **Ambiguïté sur la provenance de l'attribut**

Afficher le nom d'une station, son tarif hebdomadaire, ses activités et leurs prix.

Requête 8a:
```
SQL>  SELECT nomStation, tarif, libelle, prix
      FROM  Station,Activite
      WHERE Station.nomStation = Activite.nomStation;
```

Requête 8b: **Définition de synonyme**
```
SQL>  SELECT nomStation, tarif, libelle, prix
      FROM  Station S, Activite A
      WHERE S.nomStation = A.nomStation;
```

SI on exrait des informations provenant de n tables, nous devons avoir au moins (n-1) conditions de jointures

Requêtes sur plusieurs tables…

On construit deux requêtes dont les résultats ont même arité (même nombre de colonnes et mêmes types d'attributs), et on les relie par un des mots-clef UNION, INTERSECT ou EXCEPT

Exemple 11:

Donnez tous les noms de région dans la base.

Requête 11:

```
SQL>  SELECT region FROM  Station
      UNION
      SELECT region FROM  Client ;
```

Exemple 12:

Donnez les régions où l'on trouve à la fois des clients et des stations.

Requêtes sur plusieurs tables

Requête 12:

```
SQL>  SELECT region FROM  Station
      INTERSECT
      SELECT region FROM  Client ;
```

Exemple 13:

Quelles sont les régions où l'on trouve des stations mais pas des clients ?

Requête 13:

```
SQL>  SELECT region FROM  Station
      EXCEPT
      SELECT region FROM  Client ;
```

Requêtes imbriquées

SQL offre la possibilité d'exprimer des conditions sur des relations

Ces relations sont construites par une requête SELECT ... FROM ... WHERE que l'on appelle sous-requête ou requête imbriquée.

Exemple 14:

on veut les noms des stations où ont séjourné des clients parisiens.

Requête 14_a:

```
SQL>  SELECT station  FROM  Sejour, Client
      WHERE id=idClient AND ville = 'Paris' ;
```

Requête 14_b:

```
SQL>  SELECT station FROM Sejour
      WHERE idClient IN (SELECT id
                         FROM Client
                         WHERE ville = 'Paris') ;
```

On peut remplacer le IN par un simple '=' si on est sûr que la sous-requête ramène un et un seul tuple.

Requêtes imbriquées – Formes de conditions

Exemple 15:

Quelle station pratique le tarif le plus élevé ?

Requête 15:

```
SQL>  SELECT nomStation
      FROM  Station
      WHERE tarif >=ALL (SELECT tarif FROM Station) ;
```

Exemple 16:

Dans quelle station pratique-t-on une activité au même prix qu'à Santalba ?

Requête 16:

```
SQL>  SELECT nomStation, libelle
      FROM  Activite
      WHERE prix IN ( SELECT prix
                      FROM Activite
                      WHERE nomStation = 'Santalba') ;
```

Agrégation

Les fonctionnalités d'agrégation de SQL permettent :

d'exprimer des conditions sur des groupes de tuples,

et de constituer le résultat par agrégation de valeurs au sein de chaque groupe.

La syntaxe SQL fournit donc :

Le moyen de partitionner une relation en groupes selon certains critères.

Le moyen d'exprimer des conditions sur ces groupes.

Des fonctions d'agrégation.

Fonctions d'Agrégation…

Ces fonctions s'appliquent à une colonne, en général de type numérique

- **COUNT qui compte le nombre de valeurs non nulles**
- **MAX et MIN**
- **AVG qui calcule la moyenne des valeurs de la colonne**
- **SUM qui effectue le cumul**
- **VARIANCE calcule la variance**
- **STDDEV calcule l'écart type**

Remarques importantes:

Pour le type DATE et chaîne de caractère, on peut utiliser seulement les fonctions COUNT, MIN et MAX

Toutes les fonctions, excepté COUNT, ignorent les valeurs NULL

Fonctions d'Agrégation…

Requête 17:

```
SQL>    SELECT COUNT(nomStation), AVG(tarif), MIN(tarif), MAX(tarif)
        FROM Station ;
```

Remarque importante:

Impossible d'utiliser simultanément dans la clause SELECT des fonctions d'agrégation et des noms d'attributs (sauf dans le cas d'un GROUP BY)

La requête suivante est incorrecte.

```
SQL>    SELECT nomStation, AVG(tarif)
        FROM Station ;
```

Dans la requête 17, on appliquait la fonction d'agrégation à l'ensemble du résultat d'une requête. Une fonctionnalité complémentaire consiste à partitionner ce résultat en groupes, et à appliquer la ou les fonction(s) à chaque groupe

Exemple 18:

afficher les régions avec le nombre de stations.

Requête 18:

```
SQL>    SELECT COUNT(nomStation), région
        FROM Station
        GROUP BY region;
```

Fonctions d'Agrégation

On peut faire porter des conditions sur les groupes avec la clause HAVING

La clause WHERE ne peut exprimer des conditions que sur les tuples pris un à un

Exemple 19:

on souhaite consulter le nombre de places réservées, par client, pour les clients ayant réservés plus de 10 places

Requête 19:

```
SQL>  SELECT nom, SUM (nbPlaces)
      FROM Client, Sejour
      WHERE id = idClient
      GROUP BY nom
      HAVING SUM(nbPlaces) >= 10;
```

Résumé

```
SELECT      column, group_function(column)
FROM        table
[WHERE      condition]
[GROUP BY   group_by_expression]
[HAVING     group_condition]
[ORDER BY   column];
```

LMD- Insert

Syntaxe:

SQL> INSERT INTO Table (A_1,.....,A_n)
VALUES (V_1,.....,V_n);

Table est le nom d'une table

A_1,.....,A_n sont les noms des attributs dans lesquels on souhaite placer une valeur. Les autres attributs seront donc a NULL (ou a la valeur par defaut). Tous les attributs spécifiés NOT NULL (et sans valeur par défaut) doivent donc figurer dans une clause INSERT.

V_1,.....,V_n sont les valeurs des attributs

Exemple 20:

SQL> INSERT INTO Client (id, nom, prenom)
VALUES (40,'MORIARTY','Dean'),
(41,'MARTY','Jean');

Donc, à l'issue de cette insertion, les attributs ville et region seront à NULL

LMD- Insert

On peut aussi insérer dans une table le résultat d'une requête.

Le nombre d'attributs et le type de ces derniers issus de la requête doivent être cohérents avec ceux de la table dont on va insérer les valeurs.

Syntaxe:

SQL> INSERT INTO R (A_1,.....,A_n)
SELECT...;

Exemple 21:

on a créé une table Sites (lieu,région) et on souhaite y copier les couples (lieu, région) déjà existant dans la table Station.

Requête 21:

SQL> INSERT INTO Sites (lieu, region)
SELECT lieu, region FROM Station;

LMD- Update, Delete

Syntaxe:

```
SQL>    DELETE FROM  table
        [WHERE condition];
```

Exemple 22:

Supprimer tous les clients dont le nom commence par ’M’.

Requête 22:

```
SQL>    DELETE FROM Client
        WHERE nom LIKE 'M%‘;
```

Syntaxe:

```
SQL>    UPDATE Table
        SET A1=v1, …, An=vn
        [WHERE condition];
```

Exemple 23:

Augmenter le prix des activités de la station Passac de 10%.

Requête 23:

```
SQL>    UPDATE Activité
        SET prix=prix*1.1
        WHERE nomStation=‘Passac’;
```

LMD- remarques

Toutes les mises-a-jour (insert, update, delete) ne deviennent définitives qu'à l'issue d'une validation par commit.

Entre-temps elles peuvent être annulées par rollback.

Exemple 24:

```
SQL>  UPDATE Activité
      SET prix=prix*1.1
      WHERE nomStation='Passac';
SQL>  COMMIT;
SQL>  DELETE FROM Client
      WHERE nom LIKE 'M%';
SQL>  ROLLBACK;
SQL>  INSERT INTO  Client (id, nom, prenom)
      VALUES  (40,'MORIARTY','Dean');
SQL>  COMMIT;
```

Exercices

Donnez l'expression SQL des requêtes suivantes:

Nom des stations ayant strictement plus de 200 places

Noms des clients dont le nom commence par 'P' ou dont le solde est supérieur à 10000

Quelles sont les régions dont l'intitulé comprend (au moins) deux mots ?

Nom des stations qui proposent de la plongée

Nom des clients qui sont allés à Santalba

Donnez les couples de clients qui habitent dans la même région. Attention : un couple doit apparaître une seule fois

Nom des régions qu'a visité Mr Pascal

Nom des stations visitées par des européens

Qui n'est pas allé dans la station Farniente ?

Quelles stations ne proposent pas de la plongée ?