TELECOM Nancy, 1A
Module Bases de Données

TP 1

Première rencontre avec une base de données et le langage SQL

Connexion au serveur de bases de données Oracle

1) depuis une machine de l'école

Lancez SQL Developer (Menu Démarrer → Tous les programmes → Oracle Instant Client→ SQL Developer). Les adeptes peuvent utiliser SQL Plus, qui se lance en passant par le même chemin, mais qui n'intègre pas d'interface graphique.

Créez une nouvelle connexion avec le menu Fichier → Nouveau, puis :

* Nom de connexion de votre choix
* Nom d'utilisateur : **votre login** UL et
* Mot de passe : **Oracle1A**
* Type de connexion : **TNS**,
* Alias de réseau : **TNCY**

2) depuis votre machine

* Téléchargez et installez SQL Developer (http://www.oracle.com/technetwork/developer-tools/sql-developer/downloads/index.html)
* Connectez-vous au VPN de l'Université de Lorraine
* Lancez SQL Developer et créez une connexion avec les paramètres suivants :
    * Nom de connexion de votre choix
    * Nom d'utilisateur : **votre login** UL et
    * Le mot de passe : **Oracle1A**
    * Type de connexion : Basic
    * Role : Default
    * Host name : oracle.telecomnancy.univ-lorraine.fr
    * Port : **1521**
    * Service Name : **TNCY**

Saisissez vos requêtes dans la feuille de calcul que vous trouverez dans l'onglet qui porte le nom de votre connexion et exécutez-les avec la touche F5 ou le bouton vert en forme de flèche.

Vous sauvegarderez au fur et à mesure vos requêtes et leurs résultats dans des fichiers.

Documentation Oracle en ligne (HTML ou PDF)

http://www.oracle.com/pls/db102

Les documents *SQL Reference* (dans la section Most Popular) est celui qui vous sera utile.

C'est parti !

A) Créez voter première table `Client` en tapant la commande suivante dans SQLPlus Worksheet. Prenez soin de commencer par effacer le message initial qui apparaît en rouge dans la fenêtre de SQLPlus Worksheet.

```
CREATE TABLE Client (
     id NUMBER (5), -- entier de 5 chiffres décimaux
     nom VARCHAR2 (20), -- chaîne d'au plus 20
     caractères
     solde NUMBER (6, 2), -- de -9999.99 à +9999.99
     CONSTRAINT client_PK PRIMARY KEY (id)      --
     garantit l'unicité de id pour chaque ligne
     ) ;
```

Remarquez que :

Les mots en majuscules sont des mots clés SQL.

Les chaînes après deux tirets (`--`) sont des commentaires.

La table `Client` est composé de 3 attributs `id`, `nom` et `solde` et d'une contrainte `client_PK`.

La contrainte `client_PK` est une contrainte de type clé primaire (`PRIMARY KEY`) qui garantit l'unicité des valeurs dans la colonne `id`, càd que chaque valeur associée à l'attribut `id` sera unique dans la table.

La table `Client` est composé de :

1. son **schéma**

   Affichez ce schéma avec la commande suivante :

   ```
   DESCRIBE Client ;
   ```

   Essayez également avec :

   ```
   DESC Client ;
   ```

2. et des valeurs courantes de son **extension** :

   Affichez ces valeurs avec la commande suivante :

   ```
   SELECT *
   FROM Client ;
   ```

Le résultat de cette requête est normalement vide tant que vous n'aurez pas ajouté de données dans votre table.

Les informations sur les schémas sont rangées dans le dictionnaire des données (également appelé catalogue de la base de données).
Ce dictionnaire peut être consulté avec une requête de la forme suivante :

```
SELECT table_name
FROM user_tables;
```

Essayez-la.

On peut modifier l'extension de la table `Client (id, nom, solde)` c'est à dire y ajouter des données avec les commandes suivantes :

```
INSERT INTO Client VALUES ( 1, 'toto', 200.0) ;
INSERT INTO Client VALUES ( 2, 'titi', 20.0) ;
INSERT INTO Client VALUES ( 3, 'titi', 20.0) ;
INSERT INTO Client VALUES ( 5, 'tata', 120.0) ;
INSERT INTO Client VALUES (15, 'bof', 150.0) ;
INSERT INTO Client VALUES (16, 'bif', 150.0) ;
```

Ajoutez ces données.
Par défaut, les valeurs sont données aux colonnes dans leur ordre de déclaration.

Toute tentative d'ajouter une nouvelle ligne dont la valeur de la clé primaire existe déjà dans la table provoque une erreur et la table ne change pas d'état, par exemple essayez:

```
INSERT INTO Client VALUES ( 5, 'tutu', 120.0) ;
```

Lisez le message d'erreur et vérifiez que le contenu de la table n'a pas changé.

B) Requêtes sur une table

1. Projection ou commande `SELECT`

Testez les requêtes suivantes :

```
SELECT solde
FROM Client ;

SELECT nom, solde
FROM Client ;
```

Q.1 L'algèbre relationnelle (que vous verrez en cours) manipule des ensembles. En est-il de même de SQL ?

```
SELECT DISTINCT solde FROM Client ;
SELECT DISTINCT nom, solde FROM Client ;
SELECT DISTINCT solde, nom FROM Client ;
```

Q.2 A quoi sert `DISTINCT` et sur quoi porte-t-il ?

2. Restriction ou `SELECT/WHERE`

Q.3 Ecrivez et testez une requête qui a pour résultat les noms des clients dont le solde est supérieur ou égal à 150 euros.

Q.4 Proposez une requête qui affiche les noms des clients dont la clé primaire est paire. La fonction prédéfinie utile est `mod (n1, n2)` qui donne le reste de la division entière de n1 par n2.

Q.5 Ecrivez une requête qui donne les noms et identifiants des clients dont le nom contient un 'o'. Vous pouvez utiliser le mot clé `LIKE` et chercher dans la documentation comment il s'utilise.

C) A propos de la valeur NULL

Dans le monde des bases de données, la valeur `NULL` signifie valeur inconnue. C'est la valeur qui est mise par défaut quand aucune valeur n'est fournie lors d'une insertion ou modification d'un tuple.

Q.6 Testez les requêtes suivantes, puis examinez le contenu de la table `Client`.

```
INSERT INTO Client VALUES (20,'riri',null);
INSERT INTO Client(id,nom) VALUES (21,'fifi');
INSERT INTO Client(id,solde) VALUES (22,150.0) ;
```

Q.7 Essayez la requête suivante, et expliquez la réponse.

```
INSERT INTO client(nom) VALUES ('loulou');
```

Les fonctions de groupes `COUNT` et `AVG` permettent respectivement de compter des valeurs et de calculer une moyenne de valeurs.

Q.8 Testez les requêtes suivantes, afin de savoir si les valeurs `NULL` sont prises en compte dans l'évaluation des fonctions de groupe :

```
SELECT COUNT(*) FROM Client ;
SELECT COUNT(nom) FROM Client ;
SELECT AVG(solde) FROM Client ;
```

D) Modification du contenu d'une table

3. La commande `UPDATE` permet de modifier les valeurs enregistrées dans une table. Exemple : augmentez le solde des clients avec la commande suivante

```
SELECT * FROM Client ;
UPDATE Client SET solde = solde + 10.0 ;
SELECT * FROM Client ;
```

4. Modification de plusieurs colonnes. Testez la commande suivante :

```
UPDATE Client
SET solde = solde - 30.0, nom='toto'
WHERE id = 2 ;
SELECT * FROM Client ;
```

Q.9 Pourquoi peut-on être sûr que le `UPDATE` précédent ne modifiera jamais plus d'un client ?

5. Une autre modification à tester :

```
UPDATE Client
SET solde = solde + 30.0
WHERE solde BETWEEN 100.0 AND 180.0 ;
SELECT * FROM Client ;
```

Q.10 Pourquoi plusieurs clients peuvent-ils être modifiés par la requête `UPDATE` précédente ?

6. Supprimez un client avec la commande le mot clé `DELETE` :

```
DELETE FROM Client WHERE id = 4 ;
SELECT * FROM Client ;
```

Q.11 Pourquoi la table n'a-t-elle pas changé de contenu suite au DELETE ?

7. Supprimez un client :

```
DELETE FROM client WHERE id = 1 ;
SELECT * FROM Client ;
```

Attention, `DELETE FROM Client` sans clause `WHERE` supprime toutes les lignes de la table Client !

On peut supprimer une table (schéma et extension)

```
DROP TABLE Client ;
SELECT * FROM Client ;
```

Testez cette suppression puis recréez et repeuplez la table `Client`.

E) Les Producteurs

Créez la table Producteur avec la structure suivante :

```
CREATE TABLE Producteur (
     id Number (5),
     nom Varchar2 (20),
     CONSTRAINT Producteur_PK PRIMARY KEY (id)
     );


INSERT INTO Producteur VALUES ( 1, 'Bricolot') ;
```

```
INSERT INTO Producteur VALUES ( 2, 'Fruit and Fibre') ;
```

Q.12 Ecrivez une requête qui réalise le produit cartésien de `Client` et de `Producteur`. Vérifiez que le nombre de lignes du produit cartésien est correct.

On peut écrire une requête qui utilise plusieurs fois la même table, par exemple le produit cartésien d'une table avec elle-même. Pour cela, il suffit de donner deux alias différents à deux occurrences de la même table comme suit :

```
SELECT p1.*, p2.*
FROM Producteur p1 , Producteur p2 ;
```

Testez.

F) Les Produits

```
CREATE TABLE Produit (
     id Number (5),
     nom Varchar2 (20),
     prix_unitaire Number (8, 2),
     producteur Number (5),
     CONSTRAINT Produit_Producteur_FK
     FOREIGN KEY (producteur) REFERENCES Producteur
(id),
     CONSTRAINT Produit_PK PRIMARY KEY (id)
     ) ;
```

La contrainte `Produit_Producteur_FK` portant sur la colonne `producteur` de `Produit` s'appelle une clé étrangère (`FOREIGN KEY`), elle référence la clé primaire du producteur du produit, donc elle désigne exactement une ligne de `Producteur`.

Cette clé étrangère peut être indéfinie (inconnue), en revanche si elle est définie le producteur correspondant doit exister obligatoirement dans la table Producteur. Si lors d'une insertion d'un produit le producteur n'existe pas dans Produit le système retournera une erreur et la ligne ne sera pas insérée. Essayez :

```
INSERT INTO Produit (id, nom, prix_unitaire,
producteur)
VALUES ( 1, 'clou', 11.0, 1) ;
INSERT INTO Produit (id, nom, prix_unitaire,
producteur)
VALUES ( 2, 'robinet', 30.0, 1) ;
INSERT INTO Produit (id, nom, prix_unitaire,
producteur)
VALUES ( 3, 'kiwi', 0.3, 2) ;
SELECT * FROM Produit ;
```

Exemple d'erreur de référence au producteur :

```
INSERT INTO Produit (id, nom, prix_unitaire,
producteur)
VALUES ( 4, 'pomme', 0.5, 3) ;
SELECT * FROM Produit ;
```

Q.13 Ecrivez une requête qui donne l'ensemble des produits avec leur producteur. Le schéma de la relation résultat est (nomProduit, nomProducteur).

Q.14 Ecrivez une requête qui donne tous les couples de noms de produits (p1,p2) tels que le prix de p1 est inférieur strictement au prix de p2. Le schéma du résultat est (nomproduit1, nomproduit2, prix1, prix2).

G) Les commandes

Un client peut commander un produit :

```
CREATE TABLE Commande (
     idClient Number (5),
     idProduit Number (5),
     quantite Number (5),
     CONSTRAINT Commande_PK PRIMARY KEY (idClient,
     idProduit),
     CONSTRAINT Commande_Client_FK
     FOREIGN KEY (idClient) REFERENCES Client (id),
     CONSTRAINT Commande_Produit_FK
     FOREIGN KEY (idProduit) REFERENCES Produit (id)
     ) ;
```

Remarquez que la clé primaire de Commande est constituée des deux colonnes `idClient` et `idProduit`. Remarquez aussi qu'on a deux contraintes de clé étrangère.

Ajoutez les lignes suivantes à la table des commandes :

```
INSERT INTO Commande VALUES (2, 2, 3) ;
INSERT INTO Commande VALUES (2, 3, 20) ;
INSERT INTO Commande VALUES (5, 1, 100) ;
INSERT INTO Commande VALUES (5, 3, 10) ;
SELECT * FROM Commande ;
```

Q.15 Ecrivez une requête qui donne le détail des commandes sous la forme (nomClient, nomProduit, quantite)

Q.16 Ecrivez une requête qui donne la liste des noms des clients qui ont passé commande. Proposer au moins deux solutions : l'une avec une jointure, l'autre sans jointure mais avec une sous-requête.