Exercice 1 : Premiére connexion à SQL*Plus

Depuis un terminal, connectez-vous à ORACLE, modifiez votre mot de passe ORACLE puis déconnectez-vous. Pour cela utilisez les informations suivantes :

0) Dans un terminal, tapez les premières lettres de la commande SQL*Plus puis essayez la complétion automatique (touche de tabulation). Si la commande est complétée alors vous pouvez passer directement au point 2).

1) Environnement utilisateur unix bash (.bashrc)
Les variables d'environnement nécessaires à la communication avec ORACLE se trouvent dans un fichier .bashrc. Donc si c'est la première fois que vous utilisez ORACLE , il vous faut :

- si la ligne `#source /export/home/users/COMMUN/.oraclerc` concernant ORACLE n'est pas présente dans votre fichier .bashrc (présent à la racine de votre compte), remplacer votre fichier par le fichier /export/home/users/COMMUN/.bashrc,
- décommenter la ligne concernant ORACLE dans le fichier .bashrc (i.e. enlever le #),
- exécuter dans un shell la commande `source .bashrc`.

2) Désormais vous n'aurez qu'à exécuter la commande unix
`sqlplus login_oracle/password_oracle` pour vous connecter à ORACLE .
Vous pouvez également utiliser `sqlplus login_oracle` ou `sqlplus`.

- votre `login_oracle` a la forme `MACS3_num` où `num` est un numéro différent pour chaque étudiant. Ce numéro vous sera attribué par votre chargé de TP.
- votre `password_oracle` est identique à votre `login_oracle`. Vous devrez le changer lors de votre premiére connexion.

SQL*Plus est le langage qui permet de communiquer avec le **sgbd** ORACLE. Les commandes de SQL*Plus servent à éditer ou executer les commandes de SQL standard implantées sous ORACLE ainsi que des commandes purement SQL*Plus. Elles servent aussi au contrôle du formatage de l'affichage.

`SQL> quit`	quitter SQL*Plus, idem avec `exit`
`SQL> password`	changement du mot de passe ORACLE
`SQL> set pause on/off`	défilement page par page (via touche RETURN) ou continu
`SQL> host COMMANDE`	exécute la commande unix COMMANDE (exemple : `host pwd`)
`SQL> desc table`	description de la structure de la table (exemple : `desc ALL_USERS`)

Remarques :

- Le prompt 'SQL> ' indique que SQL*Plus est prêt à recevoir une commande. Après avoir tapé sur return, la commande s'exécutera ou bien un code d'erreur s'affichera.
- Toute commande SQL se termine par le caractère ';'.
  Si vous oubliez le point-virgule pour un ordre SQL (ou bien si votre ordre tient sur plusieurs lignes), après avoir tapé sur return, "2" apparait à la place du prompt. vous devez compléter votre commande.
- Il est recommandé de ne pas insérer une ligne blanche dans un ordre.
- ORACLE ne fait pas la distinction entre majuscule et minuscule dans les commandes de SQL*Plus, mais cette distinction est faite au niveau du contenu des tables.
- Une ligne de commentaire commence par REM ou bien # ou commence par /* et se termine par */.

Exercice 2 : Création de tables et notion de contraintes

SQL*Plus peut être utilisé en lignes de commandes. Attention, les lignes de commandes ne sont pas mémorisées. Vous devez donc les sauvegarder dans un fichier auquel on donnera une extension .sql pour garder une trace de votre travail.

1) Création de tables

La création d'une table s'effectue grace à la commande `create table` qui s'utilise de la façon suivante :
`create table NOM_TABLE (attribut_1 type_1,...,attribut_n type_n);`
Les types de données usuelles sont
`char(n)` : chaines de caratères de longueur fixe, n $\leq$ 2000.
`varchar2(n)` : chaines de caractères de longueur variable, n $\leq$ 4000.
`number[(n[, k])]` : nombres décimaux de n chiffres en total dont k chiffres après le point décimal, n $\leq$ 38.
`date` : date ; format de défaut : `'DD-MON-YY'` (`'18-OCT-12'` : 18 Octobre 2012).
Valeur NULLE représente l'absence de valeur ou une valeur inconnue.

Créez dans un fichier nommé `tp1ex2_creation.sql` les tables suivantes :

```
EMP(Nom, Noss, Salaire, Departement)
DIR(Noss, Departement)
```

Un n-uplet de EMP désigne le nom, le numéro de sécurité sociale, le salaire d'un employé et le nom du département dans lequel il travail. Un n-uplet de DIR désigne le numéro de sécurité sociale d'un directeur et le nom du département qu'il dirige.

2) Description et suppression de tables

`describe NOM_TABLE;`
permet d'obtenir le descriptif d'une table.

`drop table NOM_DE_TABLE [, ...];`
supprime des tables de la base de données. Seul son propriétaire peut détruire une table.

Vérifiez la description de ces tables, puis supprimez les.
Vous prendrez la bonne habitude de commencer un fichier de création de tables... en les supprimant.

3) Création de tables avec contraintes

L'ajout de contraintes s'effectue lors de la création d'une table en utilisant
`constraint NOM_CTE contrainte`
où les contraintes peuvent être de types
`not null` : pour imposer que toutes les valeurs d'une colonne soient connues.
`primary key(A1,..., Ak)` : pour définir que l'ensemble de colonnes `A1,..., Ak` forment la clé primaire d'une table.
`foreign key(A1, .., Ak) references NOM_TABLE(B1, .., Bk)` : pour imposer que les colonnes `A1, .., Ak` se référencent respectivement aux colonnes `B1,.., Bk` de la table `NOM_TABLE`.
`check (condition simple)` : pour imposer que les valeurs d'un attribut (colonne) doivent satisfaire une condition simple.

Recréez les tables `EMP` et `DIR` en ajoutant les contraintes de clés primaires et étrangères. On imposera également qu'un employé doit avoir un nom et son salaire doit être positif.

4) Insertion de n-uplets et contraintes

`insert into NOM_TABLE values n_uplet ;`
permet d'insérer un n-uplet dans la table `NOM_TABLE`.
`select * from NOM_TABLE ;`
affiche le contenu de la table `NOM_TABLE`.

- Ajoutez plusieurs n-uplets dans les tables `EMP` et `DIR`. Vous prendrez soin de sauvegarder les commandes dans un fichier nommé `tp1ex2_insertion.sql`.
- Affichez le contenu des tables.
- Testez les contraintes en essayant d'insérer deux fois la même valeur de clé, en insérant un employé de nom inconnu, en insérant un numéro `Noss` trop grand, un nom d'employé trop long.

Exercice 3 : Création de tables

Pour traiter de la vente par correspondance on considère la modélisation suivante (prenez un cas réel que vous connaissez) :

`CLIENT(numClient, nom, prenom, adresse, codePostal)`
`PRODUIT(numProduit, designation, prixUnitaire)`
`FOURNISSEUR(numFournisseur, raisonSociale)`

Les clés des relations ont été soulignées. On dispose également de la table suivante reliant les informations contenues dans les précédentes :

`COMMANDE_PRODUIT(numClient, numProduit, quantite, numFournisseur)`

Utilisez des fichiers comme précédement pour :

1) Créez ces quatre tables en tenant compte du fait que chaque numéro est sur 3 chiffres, que toute contrainte est identifiée à l'aide d'un nom et qu'un client commande des produits dans une quantité donnée. Ces tables sont-elles en troisième forme normale ?

2) Il est usuel de n'avoir dans la table `COMMANDE_PRODUIT` que les commandes en cours non traitées avec le moins de redondances possibles : si un client commande plusieurs fois le même produit, on met à jour le n-uplet correspondant en additionnant les quantités. Sachant que chaque produit est fourni par un unique fournisseur, la table `COMMANDE_PRODUIT` est-elle en troisième forme normale ? Si ce n'est pas le cas comment peut-on la transformer ?
3) On considère les tables ci dessous à la place de la précédente, est-il possible de les créer en ne précisant que des contraintes de clés étrangères ? Créez les tables sachant qu'un fournisseur peut fournir plusieurs produits.

`COMMANDE(numClient, numProduit, quantite)`
`FOURNIT(numFournisseur, numProduit)`

4) Insérez au moins 5 n-uplets dans chaque table.

Exercice 4 : Requêtes

En résumé, pour traiter de la vente par correspondance on considère la modélisation suivante :
`CLIENT(client, nom, prenom, adresse, codePostal)`
`PRODUIT(produit, designation, prixUnitaire)`
`FOURNISSEUR(fournisseur, raisonSociale)`
`COMMANDE(client, produit, dateCommande, quantite)`
`FOURNIT(fournisseur, produit)`
Les clés des relations ont été soulignées.
On rappelle que chaque numéro est sur 3 chiffres, que toute contrainte est identifiée à l'aide d'un nom et qu'un client commande des produits dans une quantité donnée, que chaque produit est fourni par un unique fournisseur et qu'un fournisseur peut fournir plusieurs produits. La table des commande renseigne les commandes non traitées.

Exprimez les requêtes suivantes en `SQL` en vous servant des schémas et instances créés précédemment. Lorsqu'il est indiqué de se servir du résultat d'une requête précédente, il ne s'agit ici que d'utiliser la **valeur** précédement trouvée/affichée et pas d'utiliser une sous-requête.

1. Donnez la liste des fournisseurs (numéro et raison sociale).
2. Donnez la liste des produits (numéro, désignation et prix unitaire), classée par ordre croissant de prix unitaire ? (Pour la suite gardez en mémoire le numéro de produit de la première ligne)
3. Quels sont les numeros de produits en commande ?
4. Quels sont les noms et prénoms des clients ?
5. Quel est le fournisseur (et le numéro de produit) du produit de prix unitaire le plus bas (en vous servant du résultat d'une requête précédente) ? (dans la suite on l'appelera F) Est-il possible d'avoir plusieurs de ces fournisseurs ? Est-il possible d'avoir plusieurs de ces produits ? Conclusion ?
6. Quel sont les numéros de produit (et le numéro de fournisseur) fournis par le fournisseur F ?
7. Choisissez deux numéros de produits. Quels en sont leurs fournisseurs (et le numéro de produit) ? Pouvez-vous formuler une autre requête pour cette question ?
8. Quel sont les numéros de produit (SANS `#fournisseur`) fournis par le fournisseur F ?
9. Quel sont les noms des clients qui habitent en Seine-Saint-Denis ?
10. Quel sont les numéros de fournisseurs dont la raison sociale contient `'SARL'` ?
11. Quel est le numéro du client qui a commandé le produit de prix unitaire le plus bas (en vous servant du résultat d'une requête précédente) ? Est-il possible d'obtenir plusieurs de ces numéros ? Est-il possible d'avoir plusieurs de ces produits ? Conclusion ?
12. Quels sont les numéros de clients (et les autres indications) qui ont passés une commande, classés par ordre décroissant de numéro de client ? De même classés par ordre décroissant de date de commande ? De même classés par ordre décroissant de quantité ? Exprimez les mêmes requêtes en n'indiquant que les numéros de clients mais pas les autres indications. Conclusion ?
13. Quel sont les numéros de clients qui n'ont pas de commande en cours ?
14. Quels sont les numéros de produits en commande dont le prix unitaire est au moins 10 euros ?

Annexes

1) Création de tables et contraintes

```
CREATE TABLE nom_de_table ( { nom_de_colonne type_de_donnees [ DEFAULT expression
] [ contrainte_de_colonne [ ... ] ] | contrainte_de_table } [, ... ] ) ;

contrainte_de_colonne ::= [ CONSTRAINT nom_de_contrainte ] {
NOT NULL | NULL | UNIQUE | PRIMARY KEY | CHECK (condition) |
REFERENCES table_de_reference [ (colonne_de_reference) ] [ ON DELETE action ]
} [ DEFERRABLE | NOT DEFERRABLE ]

contrainte_de_table ::= [ CONSTRAINT nom_de_contrainte ]{
UNIQUE (liste_de_colonne) | PRIMARY KEY (liste_de_colonne) | CHECK (condition) |
FOREIGN KEY (liste_de_col) REFERENCES table_de_ref [ (liste_de_col_de_ref) ]
[ ON DELETE action ]
} [ DEFERRABLE | NOT DEFERRABLE ]
```

`CREATE TABLE` créera une nouvelle table initialement vide dans la base de données courante. La table sera la propriété par l'utilisateur lançant la commande.
Si un nom de schéma est donné (par exemple, `CREATE TABLE monschema.matable ...`), alors la table est créée dans le schéma spécifié. Sinon, il est créé dans le schéma actuel. Le nom de la table doit être distinct des noms des autres tables, séquences, index ou vues dans le même schéma.
`CREATE TABLE` crée aussi automatiquement un type de données qui représente le type composé correspondant à une ligne de la table. Du coup, les tables ne peuvent pas avoir le même nom que tout type de données du même schéma.
Les clauses de contrainte optionnelle spécifient les contraintes que les nouvelles lignes ou les lignes mises à jour doivent satisfaire pour qu'une opération d'insertion ou de mise à jour réussisse. Une contrainte est un objet `SQL` qui aide à définir l'ensemble de valeurs valides de plusieurs façons.
Il existe deux façons de définir des contraintes : les contraintes de table et celles des colonnes. Une contrainte de colonne est définie pour faire partie d'une définition de la colonne. Une définition de la contrainte des tables n'est pas liée à une colonne particulière. Chaque contrainte de colonne peut aussi être écrite comme une contrainte de table. La contrainte de clé primaire spécifie qu'une ou plusieurs colonnes d'une table pourraient contenir seulement des valeurs uniques, non null. Techniquement, `PRIMARY KEY` est simplement une combinaison de `UNIQUE` et `NOT NULL`, mais identifier un ensemble de colonnes comme clé primaire fournit aussi des métadonnées sur le concept du schéma, car une clé primaire implique que d'autres tables pourraient se lier à cet ensemble de colonnes comme un unique identifiant pour les lignes. Une seule clé primaire peut être spécifiée pour une table, qu'il s'agisse d'une contrainte de colonne ou de table.

Lorsque les donnés des colonnes référencées sont modifiées, certaines actions sont réalisées sur les données dans les colonnes de cette table. La clause `ON DELETE` spécifie l'action à réaliser lorsqu'une ligne référencée de la table référencée est en cours de suppression. Les actions suivantes sont possibles : `NO ACTION, RESTRICT` (jamais déferrée), `CASCADE , SET NULL, SET DEFAULT`. Remarque : `NOT DEFERRABLE` = vérifiée immédiatement après chaque commande.

2) Suppression de tables

```
DROP TABLE nom_de_table [, ...] [ CASCADE | RESTRICT ] ;
```

`DROP TABLE` supprime des tables de la base de données. Seul son propriétaire peut détruire une table. Pour vider une table de lignes, sans détruire la table, utilisez `DELETE`.
`DROP TABLE` supprime tout index, règle, déclencheur et contrainte existant sur la table cible. Néanmoins, pour supprimer une table qui est référencé par une contrainte de clé étrangère d'une autre table, `CASCADE` doit être spécifié (`CASCADE` supprimera la contrainte de clé étrangère, pas l'autre table elle-même) .

3) Altération de tables

```
ALTER TABLE nom_de_table une_alteration ;
une_alteration ::=
    ADD nom_de_colonne domaine [DEFAULT expression] [ contrainte_de_colonne ] |
    ADD contrainte_de_table |
    DROP COLUMN nom_de_colonne [ RESTRICT | CASCADE ] |
    MODIFY nom_de_colonne [domaine] [DEFAULT expression] [contrainte_de_colonne] |
    RENAME COLUMN nom_de_colonne TO nouvelle_colonne |
    RENAME TO nouveau_nom |
    DROP CONSTRAINT nom_de_contrainte [ RESTRICT | CASCADE ] |
```

`ALTER TABLE` change la définition d'une table existante. Il y a plusieurs variantes :

- `ADD nom_de_colonne domaine` et `ADD (liste_de_colonnes)` ajoutent une nouvelle colonne à la table en utilisant la même syntaxe que `CREATE TABLE`. Une nouvelle colonne ne peut avoir la contrainte `NOT NULL` que si la table est vide.
- `ADD contrainte_de_table` ajoute une nouvelle contrainte à une table en utilisant la même syntaxe que `CREATE TABLE`.
- `DROP COLUMN nom_de_colonne` supprime une colonne d'une table. Les indexes et les contraintes de table référençant cette colonne sont automatiquement supprimés. Il faut utiliser l'option `CASCADE` si certains objets hors de la table dépendent de cette colonne, comme par exemple des références de clés étrangères ou des vues.
- `DEFAULT expression` ne modifient pas les lignes déja présentes dans la table. Les valeurs par défaut ne s'appliquent qu'aux prochaines commandes `INSERT`. Des valeurs par défaut peuvent aussi être créées pour les vues.
- `RENAME TO` change le nom d'une table (ou d'un index, d'une séquence, d'une vue) ou le nom d'une colonne de la table. Elle est sans effet sur les données stockées.
- `DROP CONSTRAINT` supprime des contraintes d'une table. Actuellement, les contraintes n'ont pas nécessairement un nom unique, si bien qu'il peut y avoir plusieurs contraintes qui ont le nom donné en paramètre. Toutes ces contraintes sont supprimées.

`ALTER TABLE` effectue une copie temporaire de la table originale. Les modifications sont faites sur cette copie, puis l'original est effacée, et enfin la copie est renommée pour remplacer l'originale. Cette méthode permet de rediriger toutes les commandes automatiquement vers la nouvelle table sans pertes. Durant l'exécution de `ALTER TABLE`, la table originale est lisible par d'autres clients. Les modifications et insertions sont reportées jusqu'à ce que la nouvelle table soit prête.

Annexe pour le TP : la clause `SELECT`

Voici une partie de la clause `SELECT` :

— `SELECT [ALL|DISTINCT] colonne | (fct(colonne) [AS nom]) [, liste_col_ou_fct(col)]`
Liste de colonne ou fonctions sur une colonne (`AVG|MAX|MIN|SUM [ALL|DISTINCT]`) ou sur le nombre de ligne d'une table (`COUNT [ALL|DISTINCT]`).

— `FROM nom_de_table [alias] [,liste_de_nom_de_table]`
Indique les tables à partir desquelles récupérer les données. Un alias permet un renommage local. On utilise l'alias à la place du nom de la table afin d'enlever des ambiguités, de simplifier l'écriture ou d'être plus explicite.

— `[WHERE condition]`
Exemples d'opérateurs utilisables dans une condition :
- opérateurs logiques (`AND, OR, NOT, IS NULL, IS NOT NULL`)
- opérateurs de chaines (`[NOT] IN|BETWEEN|LIKE`)
- opérateurs arithmétiques (`+, -, *, /, mod(,)`)
- comparateurs arithmétiques (`<, >, =, <>, <=, >=, [NOT] IN`)

— `[ORDER BY colonne [ASC|DESC] [,list_col]]`
Tri des résultats selon une (des) colonne(s). Toutes colonnes du select qui n'est pas sous une fonction doit etre spécifiée dans la liste des colonnes de `ORDER BY` si on souhaite effectuer un tri.

Clauses obtenues par opération emsembliste :

`(clause_select) UNION [ALL] (clause_select)`
`(clause_select) INTERSECT [ALL] (clause_select)`
`(clause_select) MINUS [ALL] (clause_select)`

Sans l'option `ALL` seuls les n-uplets distinct sont conservés (à tester : support de `ALL` sous Oracle 9i pour les deux dernières opérations). Rappel : il faut respecter les types (i.e. les domaines en algèbre relationnelle) colonne à colonne.