PETIT GUIDE DE SQL*PLUS

% ShigeruM % 29 octobre 2015

Zeste de savoir

Table des matières

1 Introduction

SQL*Plus est un outil en ligne de commande permettant de se connecter à une base de données Oracle afin d'y exécuter des ordres SQL ou des procédures PL/SQL. Dans ce tutoriel, je vous propose de découvrir ses principales fonctionnalités.

Même s'il est parfois plus confortable d'utiliser un outil graphique, tel que Toad ou SQL Developer, il faut aussi savoir se contenter de la ligne de commande. Tout d'abord, parce que nous n'avons pas toujours le choix ! Mais même au-delà, il peut s'avérer plus simple et plus rapide de lancer une invite SQL*Plus plutôt que l'artillerie lourde d'un client graphique.

Nous allons donc voir que SQL*Plus permet d'envoyer des commandes SQL et PL/SQL au SGBDR, mais aussi d'exécuter ses propres commandes internes. Nous parlerons notamment des scripts et des variables gérés par SQL*Plus. Nous apprendrons également à manier son buffer et paramétrer l'affichage des résultats afin de les rendre plus lisibles. Enfin, nous verrons comment exporter les résultats de nos requêtes dans des fichiers.

Attention, nous n'aborderons pas ici l'installation de SQL*Plus ou même d'une base Oracle, nous partirons du principe que cela a déjà été fait par votre DBA préféré et qu'il ne vous reste plus qu'à vous mettre au travail.

[DBA] : Database Administrator [SGBDR] : Système de Gestion de Base de Données Relationnelle

2 Connexion à une base de données

2.1 Petit rappel sur le modèle client-serveur

Une base de données permet de stocker de l'information, sous quelque forme que ce soit. Bien des façons de faire existent, l'important étant que les données soient stockées de façon structurée selon une norme précise afin de nous permettre de les consulter, de les mettre à jour ou encore d'en insérer de nouvelles. La base doit donc être accompagnée d'un logiciel permettant de la gérer, le SGBD. C'est lui qui va nous permettre d'interroger la base ou encore de gérer la sécurité (les droits d'accès) et l'intégrité des données.

Reste à savoir comment consulter, modifier ou insérer des données dans la base. En effet le SGBD ne fait pas tout, il faut bien lui dire quoi faire avec nos données ! Il nous faut donc être capable de communiquer avec lui, c'est-à-dire lui donner des ordres (appelés « requêtes ») et recevoir ses réponses. On utilise pour cela un langage de requête, tel que le SQL.

La plupart du temps, les SGBD fonctionnent sur le modèle « client-serveur » (vous pouvez en voir une illustration sur la figure ci-dessous). On distingue alors les deux composantes suivantes :

— le client, qui envoie ses requêtes au serveur ;
— le serveur, dont la tâche est d'attendre les requêtes du client, avant de les traiter et de lui envoyer une réponse.

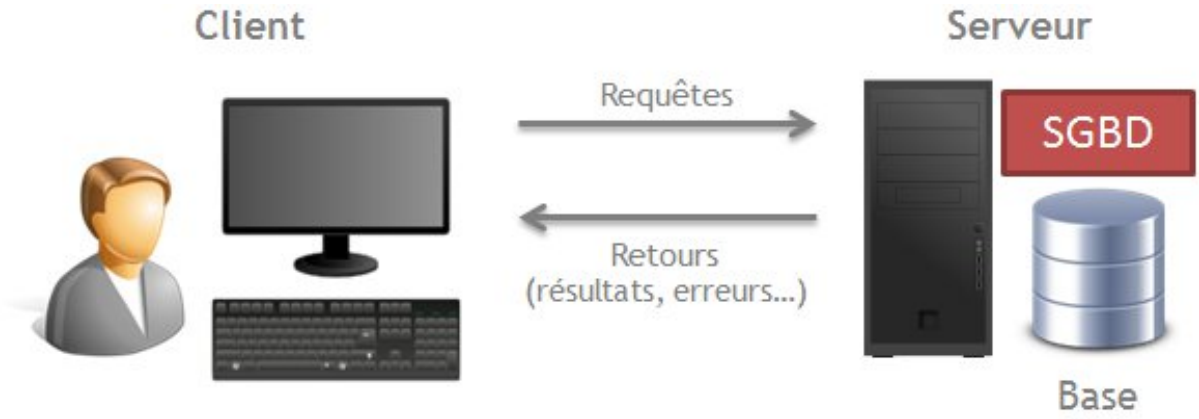

Figure 2.1 – Le modèle client-serveur

Ces deux composantes peuvent être installées sur un seul et même ordinateur ou bien sur deux machines distinctes qui vont communiquer via le réseau.

Dans ce tutoriel, nous allons considérer que les installations de la partie serveur (dont fait partie la base de données) comme de la partie cliente (dont fait partie SQL*Plus) sont faites et opérationnelles.

2.2 Première connexion avec SQL*Plus

Commencez donc par ouvrir un invite de commande. Sur Windows, tapez « cmd » dans le menu Démarrer ou sur l'écran d'accueil et choisissez « Invite de commande ». Côté Linux, je pense que vous savez déjà comment on ouvre un terminal. ;)

Quel que soit votre système, la commande de connexion à SQL*Plus est la même : `sqlplus`, suivie du nom d'utilisateur, du mot de passe et du nom du service auquel se connecter. On peut résumer la commande ainsi :

```
sqlplus nom_utilisateur/mot_de_passe@service
```

Les noms de services disponibles sont paramétrés dans le fichier `tnsnames.ora` de la machine cliente (même si dans notre cas, serveur et client sont confondus). Ce fichier se situe par défaut dans le répertoire `<ORACLE HOME>/network/admin/`, le répertoire ORACLE HOME étant le répertoire qui accueille votre installation. Par exemple, votre `tnsnames.ora` peut contenir un passage tel que le suivant :

```
XE =
  (DESCRIPTION =
    (ADDRESS = (PROTOCOL = TCP)(HOST = shigerum-pc)(PORT = 1521))
    (CONNECT_DATA =
      (SERVER = DEDICATED)
      (SERVICE_NAME = XE)
    )
  )
```

Ici le service se nomme `XE`. Il est décrit par sa chaîne de connexion, comportant notamment une partie `ADDRESS`, avec ses paramètres `HOST`, `PORT`, etc. Dans mon exemple, l'hôte est ma machine, `shigerum-pc`. En voyant la longueur de cette chaîne de connexion, on comprend l'intérêt du fichier `tnsnames.ora` : il permet de donner un nom court et simple, un « alias », à la connexion. Cet alias peut alors être utilisé dans les applications clientes telles que SQL*Plus. Si vous décidez d'utiliser un client graphique tel que SQL Developer ou Toad, vous pourrez là encore utiliser les alias du `tnsnames.ora`.

Bref, je n'entre pas plus que cela dans les détails, la configuration de ce fichier dépasse le cadre de ce cours. Sachez simplement qu'il existe et qu'il est nécessaire pour la connexion du client au serveur.

Une fois le nom de l'utilisateur et le mot de passe renseigné, le prompt `SQL>` apparaît. Voici ce que donne la connexion dans l'invite de commande de Windows par exemple (le résultat est strictement le même sous Linux) :

L'invite de commande `SQL>` signifie que vous vous adressez à Oracle. Les commandes que vous entrez sont alors des ordres SQL destinés au SGBDR et non plus des commandes systèmes pour l'OS. Ainsi, à partir du moment où vous voyez cet invite de commande, il n'y a plus de différence à faire selon que vous ayez installé Oracle sur Linux ou sur Windows.

Et voilà, notre première connexion est ainsi établie ! À l'aide de l'outil SQL*Plus, nous avons connecté notre client à notre serveur de base de données.

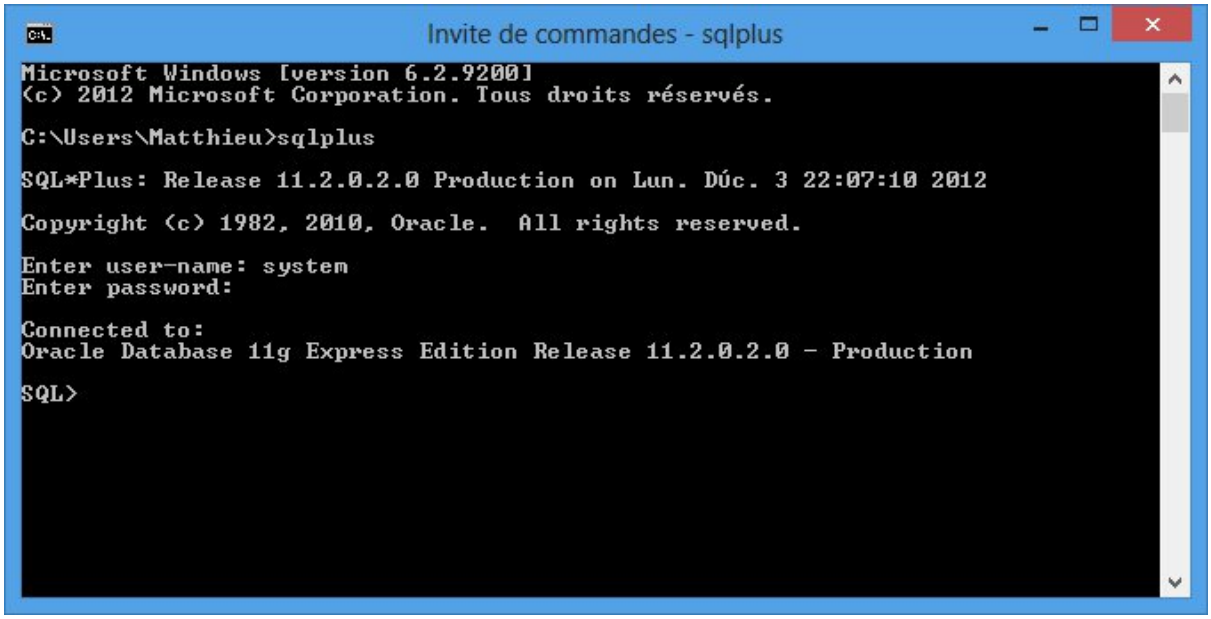

Figure 2.2 – Connexion à la base sous Windows

2.3 Déconnexion

La commande de déconnexion est la commande `exit`. Lorsque vous la tapez, vous sortez de l'outil SQL*Plus et revenez à l'invite de commande de votre système.

Bien, nous savons maintenant nous connecter et envoyer des requêtes SQL à notre base. Entrons maintenant dans le vif du sujet avec les premières fonctionnalités de SQL*Plus.

*[SGBD] : Système de Gestion de Base de Données

3 Utilisation basique de SQL*Plus

SQL*Plus accepte trois types de commandes :

- les commandes SQL : le langage de **requête** utilisé pour extraire les données de la base ;
- les commandes PL/SQL : langage **procédural** propre à Oracle, utilisé pour intégrer des requêtes SQL à des traitements procéduraux (c'est-à-dire en utilisant des variables, des boucles, des structures de contrôles, etc.) ;
- les commandes SQL*Plus : commandes internes de SQL*Plus, permettant entre autre de le paramétrer, comme nous allons le voir dans un instant.

Dans la suite de ce tutoriel, nous allons utiliser l'utilisateur « HR », pré-installé avec Oracle. Celui-ci va nous être utile car son schéma associé contient quelques tables déjà alimentées en données :

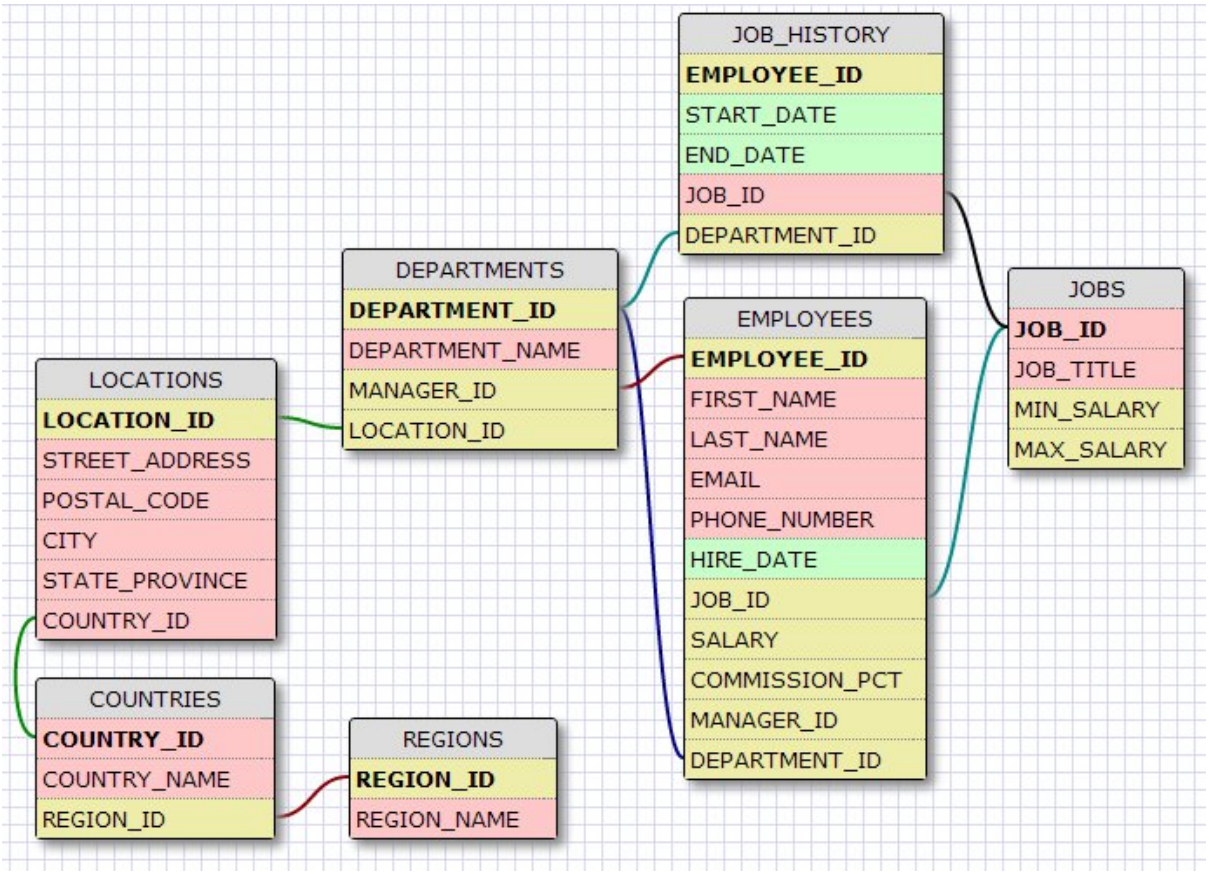

Figure 3.1 – Tables de l'utilisateur HR

Même si vous ne parlez pas parfaitement anglais, vous devriez vite vous apercevoir que ces tables ressemblent à l'organisation d'une entreprise. Plus précisément, cela concerne un service de ressources humaines (« HR » signifie « Human Resources »). On retrouve une table *EMPLOYEES* (employés), une table *JOBS* (métiers), une table *DEPARTMENTS* (départements), etc.

Par défaut, cet utilisateur est désactivé. Si vous pouvez (base de Développement, droits d'accès, etc.), alors vous pouvez l'activer à l'aide de la requête suivante, connecté en tant que SYSTEM :

```
ALTER USER HR ACCOUNT UNLOCK ;
```

Après cela, on donne à cet utilisateur un nouveau mot de passe (on ne connaissait pas l'ancien de toute façon) :

```
ALTER USER HR IDENTIFIED BY MotDePasseTopSecurite2;
```

Bref, si vous le pouvez, je vous invite donc à utiliser HR pour pouvoir exécuter les exemples en même temps que moi. Sinon ce n'est pas bien gênant, rassurez-vous.

3.1 Envoi de commandes à SQL*Plus

À partir du moment où le prompt `SQL>` apparaît, vous vous adressez à SQL*Plus et non plus au système d'exploitation. SQL*Plus étant identique sur Windows et Linux, il n'y a plus de différence à faire selon l'environnement : tout le monde est logé à la même enseigne.

Pour envoyer une requête SQL simple, il suffit de la taper et de valider par la touche ||Entrée||. Attention, n'oubliez pas le point-virgule finale, sans quoi vous reviendriez à la ligne plutôt que d'envoyer votre requête :

```
SQL> select *
  2  from regions;

 REGION_ID REGION_NAME
---------- -------------------------
         1 Europe
         2 Americas
         3 Asia
         4 Middle East and Africa
```

Il est en effet possible d'écrire des requêtes sur plusieurs lignes. Dans ce cas, le retour à la ligne au milieu de la requête est indiqué par la numérotation de la seconde ligne. Cela peut s'avérer pratique pour les requêtes un peu longue. De plus, il est possible de copier-coller une requête depuis n'importe quel éditeur vers la ligne de commande, les retours à la lignes sont alors gérés automatiquement.

Les commandes SQL*Plus ne nécessitent de leur côté pas de point-virgule final. C'est le cas de la commande `EXIT` par exemple. Nous allons voir quelques autres exemples dans quelques instants.

Je laisse volontairement de côté les commandes PL/SQL pour le moment. Nous y reviendrons dans la quatrième partie du tutoriel.

3.2 Les variables SQL*Plus

SQL*Plus est capable de gérer des variables. Une variable, vous le savez, est une zone mémoire dans laquelle on stocke une valeur afin de la réutiliser par la suite, dans diverses situations.

Il faut distinguer deux types de variables SQL*Plus :

— les **variables utilisateurs**, utilisées essentiellement pour les requêtes SQL, mais aussi pour le fonctionnement interne de SQL*Plus ;
— les **variables de lien**, utilisées cette fois pour les commandes PL/SQL.

3.2.1 Les variables utilisateurs

Les variables utilisateurs sont des variables tout ce qu'il y a de plus classiques : on les définit et on leur attribue une valeur avant de pouvoir les réutiliser. Prenez par exemple la requête suivante :

```
SELECT * FROM &ma_table ;
```

Au sein d'une requête SQL, on utilise une variable en préfixant son nom d'une esperluette (« & »). Comme vous pouvez le voir ci-dessus, à la place du nom de la table est positionné la variable &ma_table. Au moment où l'on envoie cette requête au SGBDR, SQL*Plus remplace la variable par sa valeur, à condition qu'elle ait été définie au préalable bien sûr.

Pour définir une variable, on utilise la commande `DEFINE` (également abrégée `DEF`) :

```
DEF[INE] [nom_de_la_variable = contenu_de_la_variable]
```

[[i]] |Ici comme dans tout le reste du tutoriel, lorsque je présenterai une structure de commande, j'utiliserai une présentation courante où les parties optionnelles sont indiquées entre crochets. De plus, même si ici la casse n'importe pas, les parties en majuscules font partie de la commande elle-même alors que celles en minuscules sont à remplacer par vos propres informations.

Ainsi pour définir ma variable &ma_table, j'utilise :

```
DEF ma_table = "REGIONS"
```

Une définition de variable est une commande interne à SQL*Plus, il n'y a donc pas de « ; » finale.

Notez que j'entoure la valeur de ma variable par des guillemets. Ceux-ci ne sont pas nécessaires, sauf si la valeur de la variable contient des espaces. Par habitude, il est donc préférable de toujours utiliser les guillemets.

[[a]] |Il n'y a pas de « & » lors de la définition d'une variable. Celui-ci n'est à écrire que pour l'**utilisation** de la variable.

Je peux à présent utiliser ma requête de tout à l'heure :

```
SQL> DEF ma_table = REGIONS
SQL> SELECT * FROM &ma_table;
old   1: SELECT * FROM &ma_table
new   1: SELECT * FROM REGIONS

 REGION_ID REGION_NAME
---------- -------------------------
         1 Europe
         2 Americas
         3 Asia
         4 Middle East and Africa
```

Lors de l'exécution, SQL*Plus indique la requête initiale (ligne « old ») et la requête contenant la transcription de la variable (ligne « new »).

Pour visualiser une variable déjà définie, utilisez la commande `DEF nom_de_la_variable` (sans lui redonner de valeur). Et pour visualiser l'ensemble des variables définies, tapez la `DEF` sans aucun paramètres :

```
console hl_lines="1" SQL> DEF DEFINE _DATE           = "06/12/12" (CHAR)
DEFINE _CONNECT_IDENTIFIER = "XE" (CHAR) DEFINE _USER          = "HR" (CHAR)
DEFINE _PRIVILEGE      = "" (CHAR) DEFINE _SQLPLUS_RELEASE = "1102000200" (CHAR)
DEFINE _EDITOR         = "Notepad" (CHAR) DEFINE _O_VERSION      = "Oracle Database 11
- Production" (CHAR) DEFINE _O_RELEASE      = "1102000200" (CHAR) DEFINE MA_TABLE
```

Dans la liste qui s'affiche, on retrouve bien notre variable ma_table, accompagnée de toute une série d'autres variables prédéfinies. Ces variables prédéfinies portent des noms réservés et sont préfixées d'un « _ ». Par exemple, la variable _USER contient le nom de l'utilisateur avec lequel vous êtes connecté (ici « HR »).

Pour supprimer une variable, on utilise la commande `UNDEFINE` (ou `UNDEF`) suivie du nom de la variable à supprimer.

3.2.2 Les variables de lien

Le principe est globalement le même pour les variables de lien, si ce n'est qu'elles ne se gèrent pas avec les mêmes commandes et qu'elles sont faites pour être utilisées dans des instructions PL/SQL.

On définit une variable de lien grâce à la commande `VARIABLE` (ou `VAR`) :

```
VAR[IABLE] [nom [NUMBER | CHAR(n)]]
```

Pour afficher une variable de lien dans SQL*Plus (et non un script PL/SQL), on utilise la commande `VARIABLE` suivie du nom de la variable.

Au sein des instructions PL/SQL, la variable doit être utilisée en préfixant son nom de deux points (« : »).

Là encore je ne m'attarde pas plus longtemps sur les variables de lien, nous en reparlerons plus longuement dans la partie consacrée au PL/SQL.

3.3 Utiliser des scripts

3.3.1 Exécution d'un script

Ecrire ou copier-coller des requêtes dans un invite de commande n'est pas ce que l'on fait de plus pratique. SQL*Plus permet donc d'exécuter des requêtes préalablement écrites dans des fichiers dont l'extension est .sql. On utilise pour cela les commandes `START nom_du_fichier.sql` ou encore `@nom_du_fichier.sql`. Les deux méthodes sont équivalentes.

Le contenu du script est alors joué. S'il contient une requête, vous obtiendrez le résultat de cette dernière. S'il en contient plusieurs, vous obtiendrez les différents résultats successivement.

Attention, vous devez indiquer avec précision le chemin vers le script à exécuter. Vous pouvez utiliser pour cela un chemin absolu ou un chemin relatif.

Le chemin absolu est le chemin complet, depuis la racine du disque dur vers le nom du fichier. Par exemple, si vous souhaitez lancer le fichier test.sql se trouvant dans le répertoire `C:\Users\Matthieu\`, vous devez taper la commande suivante :

```
SQL>@C:\Users\Matthieu\test.sql
```

Le chemin relatif est le chemin partant du répertoire dans lequel vous vous trouviez **au moment où vous avez lancé le programme sqlplus**. Si votre script se trouve dans ce répertoire, vous pouvez simplement vous contenter de la commande suivante :

```
SQL>@test.sql
```

Notez également que vous pouvez omettre l'extension .sql lors de l'appel du fichier.

3.3.2 Utilisation de variables utilisateurs dans les scripts

À partir de maintenant, on commence à pouvoir faire des choses rigolotes : il est tout à fait possible d'utiliser des variables utilisateurs au sein des scripts .sql.

Vous pouvez par exemple reprendre notre requête de tout à l'heure (`SELECT * FROM &ma_table;`), la placer dans un fichier mon_script.sql et exécuter ce dernier avec SQL*Plus.

[[q]] |D'accord, mais si la variable &ma_table n'a pas été définie, que va-t-il se passer ?

Eh bien si SQL*Plus trouve une variable non définie dans le script, pragmatique comme il est, il vous demandera sa valeur :

"'console hl_lines="2" SQL> @mon_script.sql Enter value for ma_table : REGIONS old 1 : SELECT *FROM &ma_table new 1 : SELECT* FROM REGIONS

REGION_ID REGION_NAME ———- ———————— 1 Europe 2 Americas 3 Asia 4 Middle East and Africa "'

Mieux encore : grâce à la commande `ACCEPT`, vous pouvez indiquer un message à l'utilisateur pour l'inviter à entrer la valeur de la variable. Voici sa structure :

```
ACCEPT nom_de_la_variable [PROMPT texte_a_afficher] [HIDE]
```

On indique le nom de la variable à renseigner et le texte à afficher à l'utilisateur du script. En prime, vous pouvez indiquer l'option HIDE, qui rendra la saisie « aveugle » (les caractères saisis par l'utilisateur ne s'affichent pas à la frappe). Cela peut être utile si la valeur à saisir est sensible et que n'importe qui ne doit pas pouvoir la voir par dessus l'épaule.

Reprenons par exemple notre script précédent et ajoutons-y un appel à la commande ACCEPT :

```
console hl_lines="1" ACCEPT ma_table PROMPT "Quelle table voulez-vous interroger ? "
SELECT * FROM $ma_table;
```

L'utilisateur se verra afficher un joli message lors de l'exécution du script.

De plus, la commande ACCCEPT a le bon goût de créer la variable. Sans elle, la valeur de la variable était automatiquement demandée mais celle-ci était oubliée à la fin de l'exécution du script. À présent, la variable est définie et apparaît bien dans la liste des variables (commande `DEF`).

Passons à présent à un autre outil proposé par SQL*Plus : le buffer.

4 Le buffer

Le **buffer** est une zone mémoire contenant la dernière commande SQL (ou PL/SQL) envoyée. Grâce à lui, il va être possible de retravailler la dernière requête envoyée ou encore de la sauvegarder.

4.1 Utilisation du buffer

Attention, le buffer contient la dernière requête envoyée et non la dernière ligne uniquement. Si vous avez envoyé une requête sur plusieurs lignes, alors le buffer les contient toutes. Pour vous en convaincre, tapez la requête suivante avec les même retour à la lignes que moi (nous utilisons toujours l'utilisateur HR) :

```
select
*
from
regions;
```

SQL*Plus vous affiche alors le résultat de cette requête comme nous l'avons vu précédemment. Tapez à présent la commande `LIST` (ou simplement `L`) pour obtenir le contenu du buffer :

```
SQL> LIST
  1  select
  2  *
  3  from
  4* regions
```

On récupère bien l'ensemble des lignes de la requête.

Pour rejouer le contenu du buffer, on utilise la commande `RUN` (ou plus court, `/`). Essayez, vous verrez.

4.2 Modifier le contenu du buffer

L'une des fonctionnalités les plus intéressantes du buffer est sans doute la possibilité de modifier une partie de son contenu. On peut ainsi modifier la requête enregistrée sans avoir à la retaper entièrement. On utilise pour cela la commande `CHANGE` (abrégée en `C`), dont la syntaxe est la suivante :

```
C[HANGE]/texte_a_remplacer/[texte_de_remplacement]
```

Reprenons par exemple notre buffer où nous l'avions laissé, avec sa requête sur plusieurs lignes. Tapez alors la commande suivante :

```
C/regions/jobs
```

A priori, rien ne se passe. Mais en visualisant à nouveau le contenu du buffer à l'aide de la commande `LIST`, on se rend compte que le buffer a été modifié :

```
SQL> LIST
  1  select
  2  *
  3  from
  4* jobs
```

Il ne vous reste alors plus qu'à jouer du `/` pour lancer votre requête retravaillée. Ici le gain est minime car la requête est courte, mais imaginez le temps gagné avec des requêtes plus complexes.

4.3 Enregistrer le contenu du buffer dans un fichier

Enfin, il est possible d'écrire le contenu du buffer dans un fichier à l'aide de la commande `SAVE` (ou `S` tout court) :

```
S[AVE] nom_du_fichier { [CREATE] | [REPLACE] | [APPEND] }
```

[[i]] |Cette fois, j'ai mis certains mots entre accolades, en les séparant par des barres verticales. Ces barres signifient « ou ». Autrement dit, une seule de ces options est attendue. Là encore, cette notation sera vraie dans tout le tutoriel. Notez que dans le cas présent les valeurs entre accolades sont également entre crochets, elles sont donc optionnelles.

Le nom du fichier est bien sûr le nom du fichier qui recevra le contenu du buffer. Vous n'êtes pas obligé de spécifier d'extension, auquel cas SQL*Plus ajoutera lui-même « .sql ».

L'emplacement de destination du fichier est le répertoire dans lequel vous vous trouviez au moment du lancement de SQL*Plus (c'est le même principe que pour l'utilisation des scripts de tout à l'heure).

L'option CREATE n'est pas nécessaire si le fichier n'existe pas déjà : il sera créé d'office. En revanche s'il existe, il vous faudra utiliser l'option REPLACE. Enfin l'option APPEND sert à ajouter le contenu du buffer à la suite d'un fichier déjà existant (pratique pour conserver quelques requêtes choisies au cours de votre travail).

La sauvegarde de buffer est très utile car elle permet de créer facilement des scripts comme on en a vu précédemment.

5 Paramétrer l'affichage des résultats

Peut-être l'avez-vous remarqué si vous avez essayé de lancer quelques requêtes avec SQL*Plus : l'affichage n'est pas ce que l'on fait de plus agréable. Alors bien sûr, en lignes de commandes, il ne faut pas s'attendre à une ergonomie parfaite… Mais tout de même, quand on voit un retour tel que celui de la figure suivante, on se dit qu'on peut améliorer les choses !

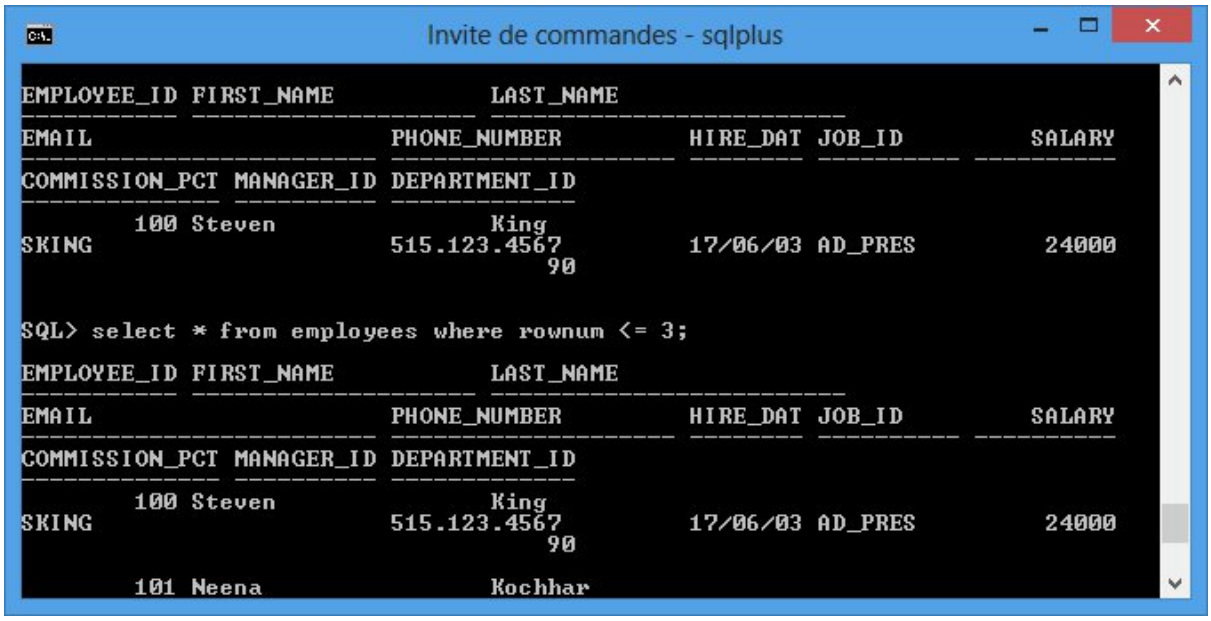

Figure 5.1 – Retour SQL*Plus non exploitable

Certains paramètres d'affichage doivent être définis à l'aide de commandes spécifiques, d'autres via la commande `SET`. Dans tous les cas, les options disponibles sont très nombreuses. Voyons donc les plus courantes.

5.1 Les paramètres de la commande SET

La commande SET permet de positionner certaines options de l'environnement. On l'utilise toujours de la même forme :

```
SET nom_du_parametre valeur_données
```

5.1.1 PAGESIZE : Longueur des pages

Parmi les problèmes d'affichage fréquemment rencontrés, on trouve la répétition régulière des noms de colonnes :

"'console hl_lines="3 17" SQL> select * from jobs ;

JOB_ID JOB_TITLE MIN_SALARY MAX_SALARY ———— ———————————————— ———— ———— AD_PRES President 20080 40000 AD_VP Administration Vice President 15000 30000 AD_ASST Administration Assistant 3000 6000 FI_MGR Finance Manager 8200 16000 FI_ACCOUNT Accountant 4200 9000 AC_MGR Accounting Manager 8200 16000 AC_ACCOUNT Public Accountant 4200 9000 SA_MAN Sales Manager 10000 20080 SA_REP Sales Representative 6000 12008 PU_MAN Purchasing Manager 8000 15000 PU_CLERK Purchasing Clerk 2500 5500

JOB_ID JOB_TITLE MIN_SALARY MAX_SALARY ———— ———————— ———— ———— ST_MAN Stock Manager 5500 8500 ST_CLERK Stock Clerk 2008 5000 SH_CLERK Shipping Clerk 2500 5500 IT_PROG Programmer 4000 10000 MK_MAN Marketing Manager 9000 15000 MK_REP Marketing Representative 4000 9000 HR_REP Human Resources Representative 4000 9000 PR_REP Public Relations Representative 4500 10500 "'

Cela provient du fait que SQL*Plus affiche les résultats sous forme de « pages » de longueur définie. Lorsque les résultats dépassent la première page, ils sont affichés à la suite dans une deuxième page et ainsi de suite. Sur chaque nouvelle page, les noms des colonnes (ainsi que d'autres éléments que nous verrons plus loin) sont ré-affichés. Aussi, pour éviter ce problème, il faut augmenter le nombre de lignes par page à l'aide du paramètre « PAGESIZE », celui-ci étant par défaut à 14 :

```
SQL> SET pagesize 100
SQL> select * from jobs;

JOB_ID     JOB_TITLE                           MIN_SALARY MAX_SALARY
---------- ---------------------------------- ---------- ----------
AD_PRES    President                               20080      40000
AD_VP      Administration Vice President           15000      30000
AD_ASST    Administration Assistant                 3000       6000
FI_MGR     Finance Manager                          8200      16000
FI_ACCOUNT Accountant                               4200       9000
AC_MGR     Accounting Manager                       8200      16000
AC_ACCOUNT Public Accountant                        4200       9000
SA_MAN     Sales Manager                           10000      20080
SA_REP     Sales Representative                     6000      12008
PU_MAN     Purchasing Manager                       8000      15000
PU_CLERK   Purchasing Clerk                         2500       5500
ST_MAN     Stock Manager                            5500       8500
ST_CLERK   Stock Clerk                              2008       5000
SH_CLERK   Shipping Clerk                           2500       5500
IT_PROG    Programmer                               4000      10000
MK_MAN     Marketing Manager                        9000      15000
MK_REP     Marketing Representative                 4000       9000
HR_REP     Human Resources Representative           4000       9000
PR_REP     Public Relations Representative          4500      10500
```

5.1.2 LINESIZE : Longueurs de lignes

Par défaut, SQL*Plus affiche les résultats sur une largeur de 80 caractères. La plupart de vos requêtes dépasseront allègrement cette taille, ce qui pourrait donner des résultats tels que celui-ci :

```
SQL> SELECT EMPLOYEE_ID, FIRST_NAME, LAST_NAME, EMAIL, PHONE_NUMBER
  2  FROM EMPLOYEES;

EMPLOYEE_ID FIRST_NAME           LAST_NAME
----------- -------------------- -------------------------
EMAIL                     PHONE_NUMBER
------------------------- --------------------
        100 Steven               King
SKING                     515.123.4567

        101 Neena                Kochhar
NKOCHHAR                  515.123.4568
```

Dans cet exemple, 5 colonnes ont été sélectionnées mais celles-ci ne s'affichent pas les unes à côtés des autres comme on pourrait s'y attendre. Les colonnes « EMAIL » et « PHONE_NUMBER » sont affichées sous les trois premières colonnes, ce qui rend l'affichage très difficile à comprendre.

Pour remédier à cela, on préférera souvent agrandir cette largeur à l'aide du paramètre « LINESIZE » :

```
SQL> SET LINESIZE 300
SQL> SELECT EMPLOYEE_ID, FIRST_NAME, LAST_NAME, EMAIL, PHONE_NUMBER
  2  FROM EMPLOYEES;

EMPLOYEE_ID FIRST_NAME             LAST_NAME                 EMAIL                     --
----------- -------------------- ------------------------- ------------------------- --
   100 Steven             King                           SKING                     515
   101 Neena              Kochhar                     NKOCHHAR                  515.12
```

Cette fois l'affichage en colonnes est bien plus présentable.

[[a]] |Si vous utilisez SQL*Plus dans l'invite de commande de Windows, vous allez tout de même rencontrer un problème de largeur dû à la fenêtre elle-même et non au paramétrage de SQL*Plus. Pour le résoudre, ouvrez les propriétés de l'invite de commande et sous l'onglet « Configuration », mettez un nombre assez grand pour les largeurs de la fenêtre et de la mémoire tampon de l'écran (attention à mettre le même nombre pour les deux largeurs).

5.1.3 TIMING : Chronométrage des requêtes

Oracle est l'un des SGBD les plus performants du marché, on le sait. Cela dit, si des requêtes complexes lui sont envoyées, il peut tout de même mettre un petit bout de temps avant de renvoyer un résultat. Connaître le temps d'exécution d'une requête peut donc être très intéressant et même très utile dans le cadre d'optimisations. On utilise alors la commande `SET TIMING [ON | OFF]` pour afficher ou masquer le temps d'exécution d'une requête :

```
SQL> SET TIMING ON
SQL> select * from regions;

 REGION_ID REGION_NAME
---------- -------------------------
         1 Europe
         2 Americas
         3 Asia
         4 Middle East and Africa

Elapsed: 00:00:00.06
```

Dans mon cas, le résultat est arrivé plutôt rapidement. ^^

5.1.4 TIME : Heure des commandes

La commande `SET TIME [ON | OFF]` permet d'ajouter l'heure avant l'invite de commandes SQL> :

```
SQL> SET TIME ON
18:59:48 SQL>
```

Afficher l'heure peut être utile, mais il est possible d'aller plus loin dans la modification de l'invite de commandes.

5.1.5 SQLPROMPT : Modification de l'invite de commande SQL>

Par défaut, l'invite de commande est paramétré pour afficher SQL>, comme on l'a vu ci-dessus. La commande `SET SQLPROMPT "texte"` permet de placer le texte voulu à la place (texte à placer entre guillemets dans la commande). Par exemple :

```
SQL> SET SQLPROMPT "SALUT> "
SALUT>
```

Il est conseillé, comme je l'ai fait ci-dessus, de positionner un caractère bien visible en fin de prompt (le chevron « > ») ainsi qu'un espace, afin que les requêtes ne « collent » pas à l'invite de commandes.

Il est possible d'utiliser les variables utilisateurs dont nous parlions plus tôt pour les afficher dans le prompt. Bien souvent, on souhaitera afficher le nom de l'utilisateur connecté ainsi que celui de la base. Dans mon cas cela donne donc :

```
SQL> SET SQLPROMPT "_user on _connect_identifier> "
HR on XE>
```

Ceci est indépendant de l'affichage de l'heure (commande TIME), vous pouvez donc combiner les deux et obtenir le plus merveilleux prompt du monde (ou presque, car il parait qu'un prompt encore plus beau a été défini quelque part dans le sud de la Bolivie).

Les options accessibles via la commande SET sont très nombreuses, nous n'avons vu ici que les principales. Passons à présent à des paramètres d'affichages à positionner grâce à d'autres commandes.

5.2 BTITLE, TTITLE : Afficher un titre et un pied de page au résultat

Plus haut, nous avons vu que l'affichage des résultats était fait par « pages », dont nous avons appris à régler la longueur. Les commandes TTITLE et BTITLE permettent d'afficher respectivement une en-tête et un pied de page. Elles s'utilisent de la même façon :

```
TTITLE [option] [texte]
```

Là encore, vous pouvez utiliser les variables utilisateurs au sein du texte.

Concernant les options, nombreuses, je ne citerai que LEFT, CENTER et RIGHT, qui permettent de définir l'alignement du texte dans la page.

[[a]] |Attention, l'en-tête et le pied de page ne sont pas affichés uniquement au début et la fin des résultats de la requêtes, mais bien pour chaque « page ». Si vous avez défini un « PAGESIZE » trop petit, ils se répéteront trop fréquemment, ce qui ne sera pas du plus bel effet.

Pour ne plus afficher l'en-tête ou le pied de page, on utilise simplement :

```
TTITLE OFF
```

5.3 COLUMN : Paramétrer l'affichage d'une colonne en particulier

Les commandes précédentes impactaient l'affichage de l'ensemble des résultats. La commande COLUMN en revanche, ne s'applique qu'à l'une des colonnes à la fois. On ne paramètre donc pas l'affichage de toutes les colonnes mais bien celui de l'une d'entre elle, que l'on désigne par son nom. De nombreuses options sont disponibles et là encore, nous ne les verrons pas toutes. Voici cependant les principales.

5.3.1 Format des colonnes de « texte »

Vous l'avez peut-être remarqué, les largeurs de colonnes contenant du texte sont parfois trop grandes par rapport à leur contenu :

```
SQL> select EMAIL FROM EMPLOYEES;

EMAIL
-------------------------
ABANDA
ABULL
ACABRIO
AERRAZUR
AFRIPP
AHUNOLD
AHUTTON
AKHOO
AMCEWEN
AWALSH
```

Cela vient du fait que SQL*Plus se base sur la longueur maximale de la colonne, définie lors de sa création. Par exemple, si votre colonne est faite pour accueillir des données d'au plus 100 caractères, alors SQL*Plus affichera une colonne de 100 caractères de large. Si vos données ne dépassent pas quelques caractères, vous perdrez donc beaucoup de place pour rien à l'écran. L'idée est donc ici d'imposer une largeur à votre colonne. Si vous savez pertinemment qu'aucune donnée ne fait davantage que 10 caractères, vous pouvez alors utiliser la commande suivante :

```
COLUMN nom_de_la_colonne FORMAT A10
```

Le « A » signifie que vous paramétrez la longueur d'un texte. Il ne faut surtout pas l'oublier car alors la commande n'a plus du tout le même sens : elle s'appliquerait à une colonne contenant des nombres.

5.3.2 Format des colonnes de « nombres »

L'option `FORMAT` de la commande `COLUMN` est également utilisée pour les colonnes de type « nombre » (là encore, nous reviendrons plus longuement sur les types de données en temps voulu).

Pour paramétrer le format d'affichage voulu pour une colonne, il est nécessaire de définir un *modèle* composé de « 9 » et de « 0 » :

- le « 9 » correspond à l'affichage de n'importe quel chiffre ;
- le « 0 » quant à lui correspond à l'affichage d'un zéro **non significatif**.

Prenons un exemple, ça sera tout de suite plus parlant. Nous voulons afficher une colonne « SCORE » contenant des nombres décimaux, comme par exemple 123,45. Par défaut, l'affichage se limite à 123,45 :

```
SQL> select SCORE from ma_table;

     SCORE
----------
       123,45
```

Parfois, il se peut qu'un besoin précis impose un affichage sous un certain format. Admettons que nous ayons besoin d'afficher ce même résultat avec exactement 5 chiffres après la virgule. Le modèle a définir serait alors le suivant :

```
COLUMN SCORE FORMAT 999.99999
```

Ce qui vous donnera :

```
SQL> select SCORE from ma_table;

     SCORE
----------
      123,45000
```

[[a]] |Le modèle défini ne doit pas être trop restrictif. Si le nombre à afficher ne « rentre » pas dans le modèle, celui-ci ne pourra pas être présenté correctement. S vous définissez par exemple le modèle « 99.99 », alors le nombre 123,45 ne sera pas affiché et sera remplacé par une série de dièses (« ######### »). De même, si vous omettez de définir le format pour les chiffres après la virgule, seule la partie décimale sera affichée. Ainsi avec un modèle en « 999 », seul « 123 » sera retourné. Fâcheux, surtout si vous cherchez à retourner des résultats comptables par exemple !

La « 0 » quant à lui, est utilisé pour l'affichage des zéros non significatifs, c'est-à-dire ici les 0 situés à gauche du premier chiffre « utile ». Là encore, cela peut répondre à un besoin particulier. Si vous souhaitez par exemple obtenir exactement exactement 5 chiffres après et avant la virgule, vous pouvez utiliser le modèle d'affichage suivant :

```
COLUMN SCORE FORMAT 00000.00000
```

Ainsi, le 123,45 sera affiché `00123.45000`.

Tout cela est bien sûr à adapter selon vos propres données.

5.3.3 Affichage des valeurs null

Dans Oracle comme dans la majorité des SGBDR, il est tout à fait possible que certains champs d'un enregistrement ne contiennent aucune valeur. On parle de valeur *null*.

SQL*Plus affiche ces valeur avec… eh bien avec rien du tout, ce qui est somme toute logique. Reprenons l'exemple de notre colonne « SCORE », qui contient ici quatre enregistrements dont un vide :

```
SCORE
----------
       ,005

     123,45
       2,15
```

Il peut être utile de tout de même afficher une valeur à la place de ce trou béant. Pour cela, on utilise la commande suivante :

```
COLUMN nom_de_la_colonne NULL texte_a_afficher
```

Si le texte à afficher contient des espaces, entourez-le de guillemets.

Je définie par exemple de la façon suivante l'affichage des null de ma colonne « SCORE » :

```
COLUMN SCORE NULL VIDE
```

J'obtiens donc le résultat suivant :

```
SCORE
----------
       ,005
VIDE
     123,45
       2,15
```

5.3.4 Effacer ou désactiver les mises en forme d'une colonne

Pour effacer tous les paramétrage liés à une colonne, on utilise le paramètre `CLEAR`, appliquée à la colonne concernée :

```
COLUMN nom_de_la_colonne CLEAR
```

Après cela, tout ce que vous aviez paramétré pour la colonne indiquée est perdu. La colonne retrouve donc son affichage par défaut.

Mais bien souvent, on souhaite simplement retrouver temporairement l'affichage par défaut et non supprimer complètement toutes les personnalisations effectuées. Pour cela, il est possible de désactiver (et ré-activer) les mises en formes d'une colonne avec `ON` ou `OFF` :

```
COLUMN nom_de_la_colonne { ON | OFF }
```

C'est aussi simple que d'appuyer sur l'interrupteur pour allumer ou éteindre la lumière.

5.4 Enregistrer les paramètres dans un fichier

Malheureusement, tous les paramètres d'affichage que nous venons de voir sont oubliés lors de la déconnexion de SQL*Plus. Cela signifie qu'en l'état, vous devez tout redéfinir à chaque nouvelle connexion… Tout cela est fastidieux, je vous l'accorde. Ça l'est d'autant plus si vous utilisez quotidiennement SQL*Plus sur la même base de données et qu'à chaque lancement de SQL*Plus vous devez redéfinir les mêmes paramétrages.

Pour que nos paramétrages soient pris en compte à chaque nouvelle session, on les enregistre alors dans le fichier `<ORACLE_HOME>/sqlplus/admin/glogin.sql` (représentant le chemin vers le répertoire « Oracle Home »).

Ce fichier est ni plus ni moins qu'un script joué à la connexion. C'est en quelque sorte l'équivalent du .bashrc ou du .profile que les linuxiens connaissent peut-être. Selon votre installation, il contient peut-être déjà des commandes. Si tel est le cas, je vous invite à essayer de comprendre à quoi ces dernières peuvent bien servir. Ajoutez simplement vos propres commandes à la fin de ce fichier.

Le principe est le même avec le contenu du buffer et les variables (variables utilisateur et variables de lien) : tous ces éléments sont perdus à la déconnexion. Vous pouvez donc les définir dans le fichier glogin.sql, bien que cela ait peut-être moins d'intérêt que pour les paramètres d'affichage.

6 Exporter les résultats dans un fichier

SQL*Plus permet d'enregistrer les retours d'une requête ou d'un script directement dans un fichier. Principal avantage : il n'y a plus besoin de copier-coller le résultat de la requête à la main dans un fichier, avant de l'envoyer au patron qui le demandait pour hier. Couplé aux possibilités d'affichages vues dans la sous-partie précédente, on génère des fichiers quasi-exploitables en l'état.

6.1 Écrire dans un fichier

Écrire dans un fichier avec SQL*Plus, c'est comme filmer avec une caméra : l'enregistrement commence lorsqu'on appuie sur un bouton et il se poursuit jusqu'à ce qu'on le coupe. Ici, le bouton est la commande `SPOOL` (ou `SPO`) :

```
SPO[OL] { nom_du_fichier | OFF }
```

Comme l'indique la structure de commande ci-dessus, le paramètre est obligatoire et doit être

- soit le nom du fichier dans lequel écrire, ce qui revient à démarrer l'enregistrement ;
- soit « OFF », ce qui arrête l'enregistrement.

Le fichier sera placé dans le répertoire courant, c'est-à-dire où vous vous trouviez lors du lancement de SQL*Plus. Le principe est le même qu'avec les scripts vus plus tôt.

Attention à ne pas confondre le spool avec la commande `SAVE`, qui pour mémoire servait à enregistrer le contenu du buffer dans un fichier. Voici un petit schéma récapitulatif des entrées et sorties de fichiers avec SQL*Plus :

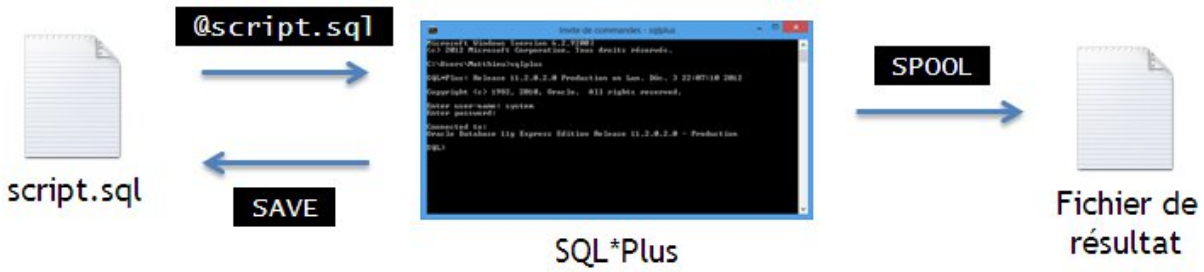

Figure 6.1 – Résumé des entrées et sorties de fichiers

6.2 Cachez ces espaces que je ne saurais voir

Si vous avez positionné un grand « LINESIZE », l'export dans un fichier va souffrir d'un petit problème : des espaces seront ajoutés à droite des résultats de requête. Si par exemple vous avez

positionné le « LINESIZE » à 1000 et qu'une ligne de votre résultat ne fait que 100 caractères de large, alors 900 espaces seront ajoutés !

Si l'on ne fait pas attention, ça ne se « voit » pas. En revanche le fichier sera plus volumineux (les espaces sont tout de même des caractères). De plus, essayez de copier-coller cela dans un document Word ou dans un mail, vous verrez le résultat…

Bref, il y a un moyen très simple d'éviter le problème, il suffit de demander à SQL*Plus de ne plus afficher ces espaces grâce au paramètre `TRIMSPOOL` de la commande `SET` :

```
SET TRIMSPOOL { ON | OFF }
```

Si ce paramètre est positionné à « ON », les espaces ne s'afficheront pas.

C'est un paramètre tellement utile et courant (en fait je ne vois pas bien la raison de le laisser à « OFF ») que je vous conseille vivement de l'ajouter à votre fichier glogin.sql.

7 Que dirirez-vous de quelques petits exercices ?

Pour terminer ce tutoriel consacré à SQL*Plus, je vous propose une petite série d'exercices, histoire de mettre en pratique ce que nous avons appris. Je vous conseille fortement d'essayer de les faire vous-même, sans regarder la solution. Toutes les informations nécessaires pour les réussir sont dans le tutoriel.

[[i]] |De petits exercices du même type seront présentés tout au long du tutoriel. Rien ne vous oblige à les faire, ils ne contiennent aucune informations supplémentaires. Cela dit la pratique est parfois ce qu'il y a de plus important pour apprendre. Amusez-vous bien !

7.1 Exercice 1 : Lancement de requête et buffer

7.1.1 Énoncé

Se connecter avec le user HR (afin d'avoir accès à ses tables). Exécuter la requête suivante dans SQL*Plus :

```
SELECT * FROM REGIONS;
```

Visualiser ensuite le contenu du buffer.

Utiliser le buffer pour modifier la requête en `SELECT * FROM DEPARTMENTS;`. Envoyer alors la commande contenu dans le buffer.

7.1.2 Solution

```
[[s]]|console hl_lines="13 22 24 26" | C:\Users\Matthieu>sqlplus |  | SQL\*Plus: Release
|  | Copyright (c) 1982, 2010, Oracle.  All rights reserved. |  | Enter user-name: hr
| Enter password: |  | Connected to: | Oracle Database 11g Express Edition Release 11.
|  | SQL> SELECT * FROM REGIONS; |  |  REGION_ID REGION_NAME | ---------- ------------
|          1 Europe |          2 Americas |          3 Asia |          4 Middle East a
|  | SQL> list |   1* SELECT * FROM REGIONS | SQL> c/REGIONS/DEPARTMENTS |
1* SELECT * FROM DEPARTMENTS | SQL> / |  | DEPARTMENT_ID DEPARTMENT_NAME
| ------------ ------------------------------ ---------- ----------- |
10 Administration                         200        1700 |            20 Marketing
|            30 Purchasing                                 114        1700 |
40 Human Resources                        203        2400 |            50 Shipping
|            60 IT                                        103        1400 |
70 Public Relations                       204        2700 |            80 Sales
```

```
|            90 Executive                                100        1700 |
100 Finance                                108        1700 |           110 Accounting
|  | DEPARTMENT_ID DEPARTMENT_NAME                MANAGER_ID LOCATION_ID
| ------------ ------------------------------ ---------- ----------- |
120 Treasury                                          1700 |           130 Corporate Ta
|           140 Control And Credit                              1700 |
150 Shareholder Services                              1700 |           160 Benefits
|           170 Manufacturing                                   1700 |
180 Construction                                      1700 |           190 Contracting
|           200 Operations                                      1700 |
210 IT Support                                        1700 |           220 NOC
|  | DEPARTMENT_ID DEPARTMENT_NAME                MANAGER_ID LOCATION_ID
| ------------ ------------------------------ ---------- ----------- |
230 IT Helpdesk                                       1700 |           240 Government S
|           250 Retail Sales                                    1700 |
260 Recruiting                                        1700 |           270 Payroll
|  | 27 rows selected. |
```

Les parties importantes sont surlignées.

7.2 Exercice 2

7.2.1 Énoncé

Créer un script contenant la requête suivante :

```
SELECT * FROM REGIONS;
```

Se connecter à SQL*Plus avec l'utilisateur HR et lancer ce script.

7.2.2 Solution

[[s]] | La création du script consiste simplement en la création d'un fichier portant l'extension .sql. Par exemple, je crée un fichier mon_script.sql sur mon Bureau et j'y place la requête : `SELECT * FROM REGIONS;` | | Dans l'invite de commande (Linux ou Windows peu importe), je me déplace tout d'abord dans le répertoire contenant mon script et je lance SQL*Plus. | | Il ne reste plus qu'à lancer le script à l'aide de la commande START ou bien avec la commande courte @ : | | `console | SQL> @mon_script.sql |  |  REGION_ID REGION_NAME |`
`---------- ------------------------ |          1 Europe |          2 Americas`
`|          3 Asia |          4 Middle East and Africa | | |` Autre méthode, j'utilise le chemin absolu vers le fichier (dans ce cas il n'est pas nécessaire de se placer dans le réperoire contenant le script avant le lancement de SQL*Plus) : | | `console |`
`SQL> @/home/shigerum/Desktop/mon_script.sql |  |  REGION_ID REGION_NAME |`
`---------- ------------------------ |          1 Europe |          2 Americas`
`|          3 Asia |          4 Middle East and Africa |`

7.3 Exercice 3

7.3.1 Énoncé

Modifier le script précédent pour que le nom de la table soit une variable et que celle-ci soit demandée avec un prompt « Quelle table voulez-vous interroger ? » lors du lancement du script.

7.3.2 Solution

[[s]] | Il suffit d'ajouter un appel à la commande ACCEPT en début de script : | | `sql | ACCEPT var PROMPT "Quelle table voulez-vous interroger ? " | SELECT * FROM &var; |` | | Ainsi, la variable est créée et sa valeur est demandée : | | `console | SQL> @mon_script.sql | Quelle table voulez-vous interroger ? REGIONS | old 1: SELECT * FROM &var | new 1: SELECT * FROM REGIONS | | REGION_ID REGION_NAME | ---------- ------------ | 1 Europe | 2 Americas | 3 Asia | 4 Middle East a | |` | De plus, si on affiche les variables définies (commande DEF), alors on constate que la variable VAR est bien toujours en place : | | `console hl_lines="10" | SQL> DEF | DEFINE _DATE = "18-DEC-12" (CHAR) | DEFINE _CONNECT_IDENTIFIER = "XE" (C | DEFINE _USER = "HR" (CHAR) | DEFINE _PRIVILEGE = "" (CHAR) | DEFINE _SQLPLUS_RELEASE = "1102000200" (CHAR) | DEFINE _EDITOR = "ed" | DEFINE _O_VERSION = "Oracle Database 11g Express Edition Release 11.2.0.2.0 - 6 | DEFINE _O_RELEASE = "1102000200" (CHAR) | DEFINE VAR = "REGIONS" |`

7.4 Exercice 4

7.4.1 Énoncé

Modifier à nouveau le script pour que cette fois le résultat soit enregistré dans un fichier. Faire en sorte que l'affichage soir correct, même en interrogeant la table *EMPLOYEES*.

7.4.2 Solution

[[s]] | Tout d'abord on ouvre la commande SPOOL avec le nom du fichier cible avant la requête, puis on la referme en fin de script : | | `sql | SPOOL resultat.txt | ACCEPT var PROMPT "Quelle table voulez-vous interroger ? " | SELECT * FROM &var; | SPOOL OFF` | | | Ici le chemin utilisé est relatif, le fichier resultat.txt va alors être créé dans le répertoire dans lequel vous étiez lors du lancement de SQL*Plus. Vous pouvez également indiquer un chemin absolu. | | Si vous exécutez ce script en utilisant une table qui contient de nombreuses colonnes (comme la table EMPLOYEES), alors le résultat ne sera pas exploitable car la longueur des lignes est par défaut trop courte. On utilise donc le paramètre « LINESIZE » pour agrandir cette dernière. Je mets par exemple la valeur 10000 afin d'être tranquille. | | Problème : de nombreux espaces sont insérés à droite de mon résultat. Le fichier est alors très lourd pour le peu de données qu'il contient (environ 1 Mo dans mon cas, pour seulement une centaine de lignes !). Je mets donc à « ON » le paramètre « TRIMSPOOL ». | | Enfin, pour éviter que les noms de

colonnes ne se répètent régulièrement, je positionne également le paramètre « PAGESIZE » à une valeur élevée. | | Finalement, voici à quoi ressemble mon script : | |

```
sql | SET TRIMSPOOL on | SET LINESIZE 10000 | SPOOL resultat.txt | ACCEPT var PROMPT "Quelle table voulez-vous interroger ? " | SELECT * FROM &var; | SPOOL OFF |
```

8 Conclusion

SQL*Plus est un outil en lignes de commandes permettant d'établir une connexion entre le client et le serveur de base de données dans le but d'envoyer des requêtes et de recevoir des résultats.

Qu'avons-nous vu dans ce tutoriel ?

- Les requêtes peuvent être tapées directement dans l'invite de commande ou bien à partir de scripts (fichiers en .sql).
- SQL*Plus gère deux types de variables : les variables utilisateurs (pour les requêtes SQL et les commandes internes de SQL*Plus) et les variables de lien (pour les commandes PL/SQL).
- Certains outils de SQL*Plus peuvent être très utile, comme le buffer ou l'export des résultats dans des fichiers (SPOOL).
- Enfin, de nombreux paramètres sont à définir pour paramétrer l'affichage des résultats. Ces paramétrages peuvent être généraux ou s'appliquer à des colonnes particulières.

Avec tout cela, vous devriez pouvoir vous débrouiller sans trop de soucis avec SQL*Plus.

N'hésitez surtout pas à commenter ce tutoriel !