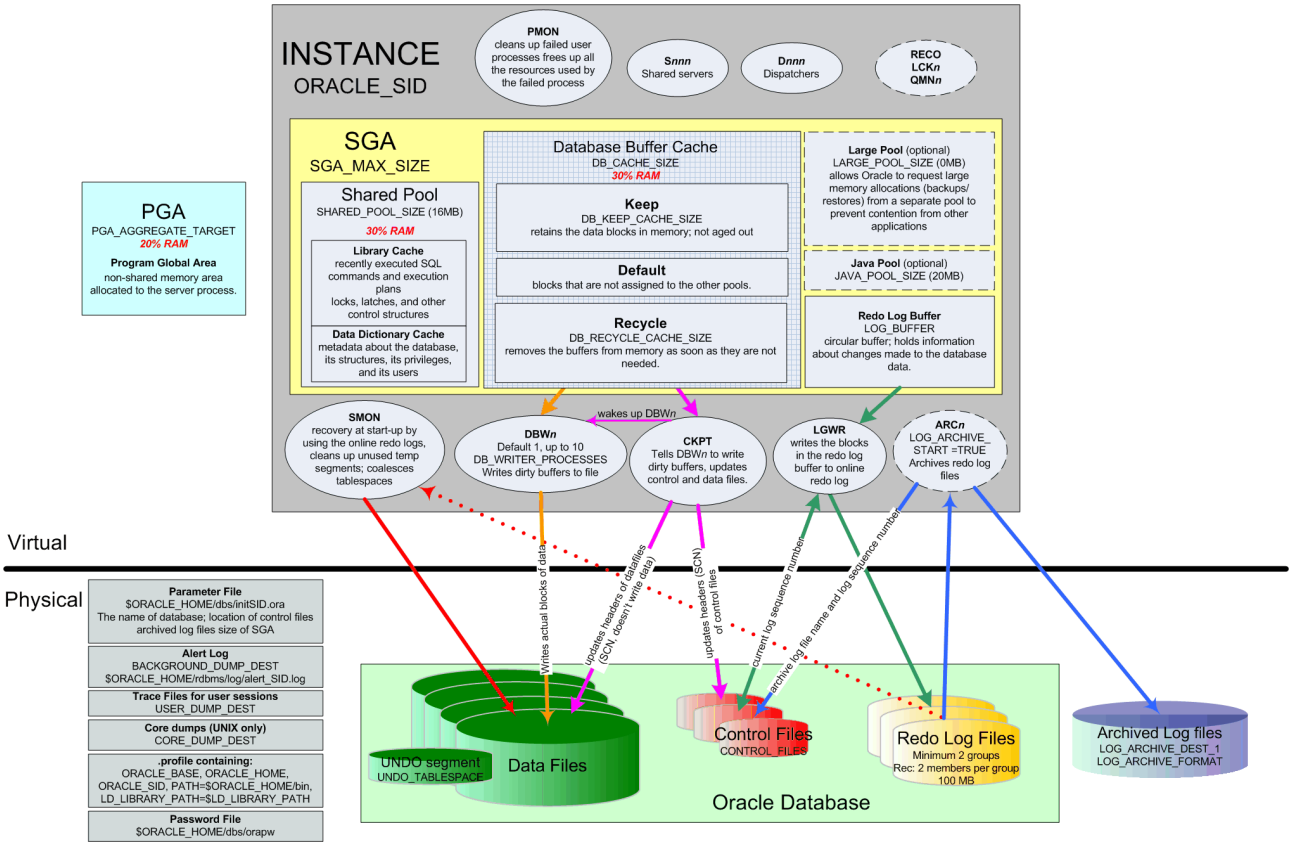
L'instance Oracle

Oracle est une base de données composée de 3 parties différentes :

- L'instance
- Les fichiers de données
- Les fichiers de données facultatifs (fichier d'initialisation, fichier de mots de passe, etc...)

Attention on entends par "fichier facultatif" le fait qu'il sera facile de recréer le ou les fichiers en questions (par exemple il sera très simple de recréer le fichier « init.ora » car celui-ci est un fichier texte contenant des paramètres Oracle).



2. Présentation Générale

L'instance est en fait la composition de 2 sous ensembles :

- **Une zone mémoire : La SGA**
- Elle va servir à stocker les données issues des fichiers de données sur le disque dur. Afin de pouvoir les partager entre les différents processus.
- **Des processus d'arrière plan :**
- Ils vont servir à gérer les transferts de données entre la mémoire et le disque dur, plus d'autres actions nécessaires au bon fonctionnement de la base de données.

L'instance est indispensable au bon fonctionnement d'une base de données Oracle. Sans instance il ne vous sera pas possible d’accéder à la base de données.

Il faut savoir qu'une instance ne pourra être assignée qu'à une seule base de données (sauf en environnement RAC).

3. La SGA ou System global Area

La SGA est une zone mémoire qui est utilisée par la base de donnée pour partager les informations entre les différents processus Oracle.

3.1. La zone mémoire : Shared Pool

La Shared Pool ou zone de mémoire partagée est utilisée pour partager les informations sur les objets de la base de données ainsi que sur les droits et privilèges accordés aux utilisateurs.

Cette zone mémoire se découpe en 2 blocs :

- La Library Cache
- Le Dictionnary Cache

On dimensionnera la taille de la Shared Pool avec le paramètre SHARED_POOL_SIZE dans le fichier init.ora.

3.1.1. La zone mémoire : Library Cache

La Library Cache est une zone mémoire qui va stocker les informations sur les ordres SQL exécutés récemment dans une zone SQL Cache qui contiendra le texte de l'ordre SQL, la version compilée de l'ordre SQL et son plan d'exécution.

Cette zone mémoire sera utilisée lorsqu'une requête sera exécutée plusieurs fois, car Oracle n'aura plus alors à recréer la version compilée de la requête ainsi que son plan d'exécution car ceux-ci seront disponible en mémoire.

3.1.2. La zone mémoire : Dictionnary Cache

Le Dictionnary Cache est une zone mémoire qui va contenir les définitions des objets de la base de données qui ont été utilisé récemment.

On entend par "définition" la structure, les composants, les privilèges liés à cet objet.

Cette zone mémoire permettra au serveur Oracle de ne pas avoir à aller chercher ces informations sur le disque à chaque exécution d'une requête SQL.

3.2. La zone mémoire : Database Buffer Cache

Cette zone mémoire sert à stocker les blocs de données utilisés récemment. Ce qui signifie que lorsque vous allez lancer une première fois la requête Oracle, cette dernière va se charger de rapatrier les données à partir du disque dur. Mais lors des exécutions suivantes les blocs de données seront récupérés à partir de cette zone mémoire, entraînant ainsi un gain de temps.

Cette zone fonctionne selon le principe dit du bloc ancien. C'est à dire que l'on peut représenter cette zone mémoire comme étant un tableau dont l'entrée serait en haut à gauche et la sortie en bas à droite. Quand un bloc est appelé pour la première fois, il se situe donc en haut à gauche. Si il n'est pas rappelé ou réutilisé il est alors déplacé lentement vers la sortie. Cependant si il est à nouveau utilisé il sera automatiquement ramené vers l'entrée de la zone mémoire.

Cette zone mémoire est définie par 2 paramètres du fichier init.ora

- **DB_BLOCK_SIZE :** Ce paramètre, défini lors de la création de la base de données, représente la taille d'un bloc de données Oracle. Celui-ci est défini de manière définitive et ne pourra plus être modifié.
- **DB_BLOCK_BUFFERS :** Ce paramètre défini le nombre de blocs Oracle qui pourront être contenus dans le Database Buffer Cache.

Ainsi pour obtenir la taille du Database Buffer Cache on effectuera l'opération DB_BLOCK_SIZE*DB_BLOCK_BUFFERS qui nous retournera une taille en Kbytes.

3.3. La zone mémoire : Redo Log Buffer

Cette zone mémoire sert exclusivement à enregistrer toutes les modifications apportées sur les

données de la base.

Cet une zone mémoire de type circulaire, et dont on pourra changer la taille avec le paramètre LOG_BUFFER (en Bytes).

Le fait que cette zone mémoire soit de type circulaire et séquentielle, signifie que les informations des toutes les transactions sont enregistrées en même temps. Le fait que ce buffer soit circulaire signifie que Oracle ne pourra écraser les données contenues dans ce buffer que si elles ont été écrites dans les fichiers REDOLOG FILE.

4. La zone mémoire : Program Global Area

Contrairement aux autres zones mémoire celle-ci n'est pas partagée. Elle est seulement utilisée par des processus serveur ou d'arrière plan. Elle est allouée lors du démarrage du processus et dés-allouée lors de l'arrêt du processus.

Cette zone mémoire contient :

- **La zone de tri :** Appelée SORT AREA, c'est ici que seront effectués les tris pour les requêtes lancées par l'utilisateur.
- **Les informations de sessions :** Cette zone contiendra les informations de sessions, les privilèges de l'utilisateur, ainsi que des statistiques de tuning concernant la session.
- **L'état des curseurs :** Cette zone permettra de connaître l'état des curseurs de l'utilisateur.
- **Le Stack Space :** Cette zone contiendra toutes les autres variables d'environnement et de session de la session de l'utilisateur.

Vous pourrez modifier la taille de la SORT AREA en changeant la valeur du paramètre SORT_AREA_SIZE.

5. Les processus d'arrière plan

5.1. SMON

Le processus SMON (ou System Monitor) est un processus qui va servir à corriger les plantages de l'instance et à vérifier la synchronisation des données. Si l'instance plante, c'est SMON qui va se charger de rejouer le contenu des REDO LOG FILE afin de pouvoir rejouer les transactions et de resynchroniser les données dans les fichiers de données.

Voici les étapes de cette récupération :

- **Etape 1 :** SMON va automatiquement rejouer les transactions qui ont été enregistrées dans les fichiers REDO LOG FILE mais pas sur le disque dur. Le fait de rejouer les transactions contenues dans les fichiers REDO LOG FILE va permettre de valider les transactions qui avaient été validées mais qui n'avaient pas pu être enregistrée sur le disque.
- **Etape 2 :** SMON va ouvrir la base de données pour les utilisateurs. Toutes les informations qui ne sont pas utilisées dans l'étape 1 et qui ont été validées sont alors disponibles immédiatement. Les autres restent verrouillées pour l'étape 3.
- **Etape 3 :** SMON se charge alors d'annuler toutes les transactions qui n'avaient pas été validés. Ce qui permettra d'avoir un état valide de la base de données.

SMON sert aussi à nettoyer les segments temporaires après leur utilisation. Il sert aussi à défragmenter les fichiers de données, tablespaces et autres.

En 9i et plus SMON utilisera la même méthode que précédemment mais avec une évolution significative lors de l'étape 1.

En Oracle 8i, SMON afin de rejouer les transactions lisait de manière séquentielle les fichiers REDO LOG FILE et rejouait toutes les transactions. Cette méthode était fortement pénalisante lors du redémarrage après le crash.

En Oracle 9i et plus, SMON va effectuer 2 lectures successive des fichiers REDO LOG FILE. La première lecture va servir à identifier les blocs qui vont nécessiter une restauration. La deuxième lecture servira à ne rejouer que les blocs identifiés. Cette méthode est évidement moins coûteuse car le temps de lecture des fichiers REDO LOG FILE est négligeable comparée à la répétition de toutes les transactions.

5.2. PMON

Le processus PMON (ou Process Monitor) va être surtout dédié aux processus des utilisateurs. Il va servir à annuler les transactions d'une session (lors d'un plantage de la session par exemple), mais aussi servir à relâcher tous les verrous posés par la session, et à relâcher toutes les ressources détenues par la session.

5.3. DBWn

Le processus DBWn (ou Database Writer) va être dédié à l'écriture du Database Buffer Cache dans les fichiers de données de la base de données.

Ce processus est aussi là pour vérifier en permanence le nombre de blocs libres dans le Database Buffer Cache afin de laisser assez de place de disponible pour l'écriture des données dans le buffer.

DBWn se déclenchera lors des événements suivants :

- Lorsque le nombre de bloc dirty atteint une certaine limite
- Lorsqu'un processus sera à la recherche de blocs libres dans le Database Buffer Cache, et qu'il ne sera pas en mesure d'en trouver.
- Lors de timeouts (environ toutes les 3 secondes par défaut)
- Lors d'un checkpoint

5.4. LGWR

Le processus LGWR (ou Log Writer) est le processus qui va écrire les informations contenues dans le REDO LOG Buffer dans les fichiers REDOLOG FILE lors des événements suivant :

- Quand une transaction est terminée avec un COMMIT

- Quand le REDO LOG Buffer est au 1/3 plein (peut dépendre de vos propres paramètres)
- Quand il y a plus de 1Mo d'informations de log contenues dans le buffer
- Avant que DBWn n'écrive le contenu du Database Buffer Cache dans les fichiers du disque dur

5.5. CKPT

Ce processus va servir à mettre à jour les en-têtes des fichiers de données, et mettre à jour les fichiers CONTROL FILE afin de spécifier que l'action de CHECKPOINT s'est bien déroulée (par exemple lors d'un changement de groupe de REDO LOG FILES).

Le CHECKPOINT est un événement qui se déclencher lors :

- D'un changement de groupe de REDO LOG FILE.
- D'un arrêt normal de la base de données (c'est à dire sans l'option ABORT)
- D'une demande explicite de l'administrateur
- D'une limite définie par les paramètres d'initialisation LOG_CHECKPOINT_INTERVAL, LOG_CHECKPOINT_TIMEOUT, et FAST_START_IO_TARGET

L’événement CHECKPOINT va ensuite déclencher l'écriture d'un certain nombre de blocs du Database Buffer Cache dans les fichiers de données par DBWn après que LGWR ait fini de vider le REDO LOG Buffer. Le nombre de blocs écris par DBWn est défini avec le paramètre FAST_START_IO_TARGET si celui-ci a été défini.

Voici une description des différents paramètres d'initialisation liés au CHECKPOINT

Paramètre	Description
LOG_CHECKPOINT_INTERVAL	Ce paramètre est défini en nombre de blocs OS (et non pas en blocs Oracle). Avant Oracle 8i ce paramètre définissait le nombre de blocs que LGWR avait à écrire avant qu'un CHECKPOINT se déclenche. Il est toutefois important de noter qu'un CHECKPOINT se déclenchera quand même lors du changement de groupe de REDO LOG FILE. De plus il est inutile de préciser que mettre une valeur 0 à ce paramètre aura une influence considérable sur les performances car LGWR se déclenchera alors à chaque nouveau bloc écris dans le REDO LOG Buffer ce qui déclenchera ensuite un CHECKPOINT. Pour les versions 8i et supérieure, ce paramètre (lorsqu'il est défini) va servir à spécifier le nombre maximum de blocs du REDO LOG Buffer qui seront alors lus lors d'une restauration de l'instance.

LOG_CHECKPOINT_TIMEOUT	Ce paramètre est défini en secondes. Avant Oracle 8i, ce paramètre permettait de définir le temps maximum entre 2 CHECKPOINTS. Pour désactiver cette fonctionnalité, il suffit de mettre la valeur 0 à ce paramètre. Pour les versions 8i et supérieure, ce paramètre, si il est défini, va servir à définir le temps maximal de lecture du processus LGWR.
FAST_START_IO_TARGET	Ce paramètre a été ajouté sur la version Oracle 8i, il va servir à fixer le nombre maximum de blocs qui devront être relus lors de la restauration d'une instance. Ce paramètre va demander à DBWn d'écrire plus fréquemment sur le disque diminuant alors le nombre de blocs à rejouer. Attention en 9i ce paramètre reste disponible mais ne représente plus le nombre de blocs maximum à rejouer mais le temps maximum pour restaurer une instance. Ce paramètre devra donc être défini en secondes. Il faudra cependant tenir compte de plusieurs choses essentielles. En effet nous serions tous tenté de donner une valeur proche de 0 dans les 2 cas. Cependant si nous appliquions cette méthode cela impliquerait que DBWn et LOGW devraient écrire tout le temps d'où une baisse significative des performances. Il est donc impératif de bien étudier vos besoins et de ne pas mettre des valeurs trop petites.

5.6. ARCn

Ce paramètre va avoir pour seule fonction de copier un fichier REDO LOG FILE à un autre emplacement. Cette copie se déclenchera automatiquement en mode ARCHIVELOG (en mode NOARCHIVELOG le processus n'existe pas) lors du changement de groupe de REDO LOG FILE.

6. Comment ça marche ?

Dans cette partie nous allons voir comment se déroulent les différentes actions dans l'instance Oracle.

6.1. Lors d'un SELECT

Dans cette explication on supposera que la base viens d'être démarrée et que aucune connexion n'a encore eu lieu. Une requête SQL va être exécutée en plusieurs étapes. Tout d'abord l'utilisateur se connecte au serveur, ce qui génère la création d'un processus serveur.

L'utilisateur va donc ensuite demander l'exécution de la requête suivante :

```
    SELECT *
FROM emp;
```

Vous remarquerez la richesse de mon exemple :). Cette requête va donc être passée au processus serveur qui va ensuite la traiter en plusieurs étapes :

- **Etape 1 :** Le parsage
- **Etape 2 :** L'exécution
- **Etape 3 :** Le fetch

6.1.1. Le parsage

Cette phase va se dérouler selon la chronologie suivante :

La première chose que le serveur va faire est de vérifier qu'il n'existe pas de requête identique. Cette action est effectuée à l'aide d'un algorithme de hachage qui va générer un numéro de série pour cette requête. Il va ensuite regarder si il ne dispose pas en mémoire (dans la LIBRARY CACHE) d'une requête disposant du même numéro.

Attention il est important de savoir que les deux requêtes suivantes ne donneront pas le même numéro de hachage

```
SELECT *
FROM emp;
-- et
select *
from emp
```

Dans ce cas précis Oracle deviens sensible à la casse, c'est pour cela que Orale vous conseille d'utiliser une convention d'écriture de vos requêtes afin d'augmenter vos chances de trouver plus rapidement vos requêtes en mémoire.

Si il ne la trouve pas en mémoire il passe à l'étape suivante qui va être de vérifier la syntaxe de votre requête, les noms des objets ainsi que vos privilèges. Encore une fois la première chose que fera le serveur sera d'aller regarder dans la zone mémoire DICTIONNARY CACHE si il dispose des informations nécessaires, sinon il ira les chercher sur le disque.

Une fois que Oracle a réussi à regrouper toutes les informations nécessaire il va verrouiller les objets en question durant la phase de parsage afin d'éviter toutes modifications de structure.

Ensuite il va générer le meilleur plan d’exécution de la requête (grâce à son optimisateur de requête) qu'il enregistrera ensuite dans la SQL CACHE de la zone LIBRARY CACHE afin d'optimiser les prochaines exécutions de la requête.

6.1.2. L'exécution

Cette étape ne sera utile que pour des requêtes de type SELECT. A ce stade le serveur dispose de toute les informations nécessaires pour exécuter la requête. Le serveur se prépare à récupérer les données.

Si les données ne sont pas présentes en mémoire, le serveur va alors les stocker en mémoire dans le DATA BUFFER CACHE.

6.1.3. Le fetch

Lors de cette étape les lignes sont récupérées, ordonnées (si nécessaire) et renvoyées au processus utilisateur. Le nombre de fetch pourra varier en fonction du nombre de lignes renvoyées par la requête.

Au final les données sont retournées sous leur forme brute. C'est le client utilisé par l'utilisateur qui se chargera de formater les données en fonction de ses paramètres. Le serveur ne renvoie pas le tableau mais seulement les données. C'est le logiciel client qui se chargera de les formater correctement.

6.2. Lors d'un ordre DML

L'exécution d'un ordre DML (Update, Insert, Delete) va seulement se décomposer en 2 phases :

- **Etape 1 :** Le parsage
- **Etape 2 :** L'exécution

6.2.1. Le parsage

Cette étape est totalement identique à la phase de parsage d'une requête SELECT.

6.2.2. L'exécution

Le serveur Oracle va alors effectuer une suite d'opérations précises. La première sera de mettre en mémoire les blocs concernés (si ils ne sont pas déjà en mémoire) puis de placer un verrou sur les données concernées.

Le serveur va ensuite enregistrer dans le REDO LOG Buffer, les informations de transactions, ainsi que les valeurs des données modifiées.

L'étape suivante va être de générer les images avant et les images après des données.

En effet afin de pouvoir consulter des données cohérentes, le serveur crée une image avant et une image après. L'image avant va servir aux autres utilisateurs. Ils pourront ainsi avoir des données cohérentes jusqu'à tant que vous terminez votre transaction. L'image après vous sera attribué le temps de votre transaction afin de pouvoir consulter les données avec vos modifications.

De plus les images avant seront très utiles car elles serviront en cas d'annulation de la transaction.

L'image avant est ensuite enregistrée dans les ROLLBACK SEGMENTS et les modifications des données sont stockées en mémoire dans l'image après. A la fin de toutes vos actions, tout bloc modifié sera alors appelé bloc DIRTY. Cela signifie que ce bloc aura besoin d'être écrit sur le disque dur car il est différent de la version présente sur le disque dur.

Il est bon de savoir que si un crash de l'instance se produit à ce moment les données seront alors automatiquement annulées car la transaction n'aura pas été terminée correctement.

6.3. Lors d'un COMMIT

Le COMMIT va permettre de terminer une transaction en validant les données qui ont été modifiées. Afin de valider de manière sure les données, Oracle utilise un processus appelé FAST COMMIT. En effet la première chose que va faire Oracle en recevant une commande COMMIT, va être d'enregistrer cette commande directement dans le REDO LOG BUFFER afin d'enregistrer le fait que la transaction a été validée au moyen de LOGWR afin que celui-ci écrive la fin de cette transaction dans les REDO LOG FILE. A ce moment précis l'utilisateur n'a toujours pas reçu le message de confirmation.

Le temps utilisé pour ces actions peut être considéré comme négligeable car il est pratiquement impossible que la base de données plante durant ces actions. Cela signifie que si la base plante après ces actions, nous serons alors en mesure de restaurer de manière totalement transparente les données non écrites sur le disque grâce à SMON.

Une fois la transaction enregistrée dans les REDO LOG BUFFER, l'utilisateur est alors informé que l'ordre a bien été effectué. Puis le processus serveur se chargera de relâcher tous les verrous posés pendant la transaction.

7. En pratique

7.1. Comment configurer une instance, le fichier init.ora

Avant de pouvoir créer notre première instance il est indispensable de créer le fichier init.ora. En effet c'est ce fichier qui va contenir les paramètres d'initialisation qui vont nous être nécessaires pour démarrer l'instance.

En 9i les paramètres d'initialisation peuvent aussi être stockés dans le fichier SPFILE de la base, permettant dans certain cas de les modifier en ligne et en connexion client/serveur.

Le fichier init.ora sera un fichier texte comprenant les paramètres d'initialisation, alors que le SPFILE sera un fichier binaire.

7.2. Comment créer une instance

7.2.1. Sous Windows

Sous Windows, les processus nécessitent d'être exécuté sous forme de service. Il va donc falloir créer notre processus d'instance comme étant un service Windows. Pour ce faire nous utiliserons l'utilitaire ORADIM.

Cet utilitaire permet de créer modifier et supprimer le service Windows qui gérera notre instance.

Voici un exemple de création d'un service

```
c:\> oradim -new -sid ORCL -intpwd oracle -startmode auto -pfile
c:\oracle\ora92\admin\orcl\pfile\init.ora
```

Avec cette commande nous demandons à l'utilitaire oradim de créer (-new) un service dédié à l'instance ORCL (-sid) de mettre ce service en démarrage automatique, c'est à dire qui démarrera tout seul lors du démarrage de Windows (-startmode). Nous spécifions aussi que cette instance sera basée sur le fichier init c:\oracle\ora92\admin\orcl\pfile\init.ora (-pfile). En même temps nous demandons à oradim de nous générer un fichier de mot de passe avec le mots de passe oracle pour l'utilisateur INTERNAL.

7.2.2. Sous Unix/Linux

Sous linux nous n'aurons pas tout ces problèmes de gestion de service. Les seules choses à faire seront d'initialiser la variable d'environnement ORACLE_SID puis de créer un fichier init.ora puis de démarrer l'instance sous sqlplus.

7.3. Comment arrêter et démarrer l'instance

Que ce soit sous Windows ou Linux/Unix la méthode est identique. La première étape sera de définir la variable d'environnement ORACLE_SID.

Ensuite il va falloir se connecter à sqlplus en mode SYSDBA afin d'être en mesure de lancer et démarrer l'instance.

```
sqlplus /nolog
connect / AS sysdba
```

Une fois connecté il ne restera plus qu'à démarrer l'instance avec la commande startup.

Le démarrage d'une base de données complète se déroulera en plusieurs étapes telles que NOMOUNT, MOUNT, OPEN.

Ces trois étapes sont obligatoires et devront être exécutées dans cet ordre. Cependant la commande STARTUP permettra d'effectuer automatiquement ces 3 actions.

- **NOUMOUNT :** Cette étape va consister à lire le fichier init.ora, à démarrer l'instance, allouer la mémoire, et démarrer les processus d'arrière plan.
- **MOUNT :** Cette étape va consister à ouvrir le ou les fichiers CONTROLEFILE afin de mettre en mémoire les informations contenues par les fichiers CONTROLEFILE. Durant cette étape les fichiers de données ne sont pas accessible car ils n'ont pas encore été ouverts.
- **STARTUP :** Cette étape va consister à ouvrir tous les fichiers de données enregistrés dans les fichiers CONTROLEFILE. Puis une fois tous les fichiers ouverts et disponible, à ouvrir complètement la base de données aux utilisateurs.

Voici des exemples d'utilisation :

```
-- Démarrage de l'instance
STARTUP NOMOUNT PFILE="<emplacement du init.ora>"
-- Lecture des controles files
ALTER DATABASE MOUNT;
-- Ouverture de la base de données
ALTER DATABASE OPEN;

-- --

-- NOMOUNT et MOUNT automatique
ALTER DATABASE MOUNT PFILE="<emplacement du init.ora>"
-- Ouverture de la base de données
ALTER DATABASE OPEN;

-- --

-- Ouverture directe de la base de données sans spécifier les 3 étapes

STARTUP PFILE="<emplacement du init.ora>"
```