1. Architecture Oracle

SERVEUR == INSTANCE + BD
INSTANCE = Structures Caches + Processus d'arrière plan (PMON, SMON, DBWR, LGWR)
BD = fichiers de données.

1.1 Éléments mis en jeu

1.2 Superviser les structures et éléments mis en jeu

La même figure vous est à nouveau proposée. Vous indiquerez maintenant les vues du méta-schéma qui servent à renseigner sur les différentes structures mémoire et processus d'arrière plan (ou autres processus). Pour vous faciliter la tâche, une liste de vues vous est fournie.

1. `v$instance` : vue portant sur l'instance (numéro de l'instance, nom de l'instance, nom de l'hôte).

2. `v$database` : vue portant sur la base de données (id, nom, date de création etc.).

3. `v$process` : vue portant sur des processus (arrières plan mais aussi d'autres processus (pas les processus client)).

4. `v$bgprocess` : vue portant sur les processus arrières plan (comme PMON, SMON, etc.).

5. `v$sga` : vue portant sur la taille du SGA.

6. `v$sgainfo` : vue donnant les mêmes informations que v$sga, mais ajoutant en plus différentes informations (telle que la mémoire libre disponible dans le SGA par exemple).

7. `v$sgastat` : vue portant sur la taille des structures du SGA (System Global Area).

8. `v$thread` : information sur les fichiers redo en cours.

9. `v$log` : information sur les fichiers log.

10. `v$sql`, `v$sqlarea`, `v$sqltext` : vues portant sur les requêtes sql en mémoire cache.

11. `v$datafile` : vue portant sur les fichiers de données.

12. `v$logfile` : vue portant sur les fichiers log.

13. `v$controlfile` : vue portant sur le/les fichiers de contrôle de la base de données (informe notamment sur leurs emplacements et sur leurs tailles).

14. `v$bh` : vue portant sur le data buffer cache.

15. `v$rowcache` : vue portant sur le dictionary cache.

16. `v$option` : vue portant sur les fonctionnalités du serveur de données.

17. `v$version` : vue portant sur la version du SGBD Oracle sous-jacent.

18. `v$session` : vue portant sur les différentes sessions en cours sur le serveur (sid, nom d'utilisateur, type de serveur (dédié ou non), adresse sql, etc.).

1.3 Vues statiques et dynamiques du dictionnaire de données

Vous expliquerez, dans un premier temps, les différences entre les vues statiques et les vues dynamiques. Pour ce qui concerne les vues statiques, vous expliquerez dans un second temps, la différence entre les vues préfixées respectivement par USER , ALL et DBA.

Vue statique → USER_%, DBA_%, ALL_%

- USER :: les vues ne concernent que les objets qui appartiennent à l'utilisateur connecté.
- ALL :: les vues ne concernent que les objets pour lesquels l'utilisateur connecté possède des permissions.
- DBA :: les vues concernent tous les objets de la base de données.

Vue dynamique -> v$: affiche des informations à un instant t de la base de données.

- Vous souhaitez connaître les tables qui vous sont disponibles en consultation. Quelle est la vue qu'il vous faut interroger à ce sujet ?
    1. USER TABLES
    2. DBA TABLES
    3. ALL TABLES

- Vous souhaitez connaître les tables qui vous sont accessibles en consultation mais qui ne vous appartiennent pas. Donner un exemple de requête qui vous permettrait d'avoir cette information.

```
SELECT table_schema, table_name, grantor, grantee, privilege
FROM all_tab_privs
WHERE grantor != 'TBONDETTI'
 AND privilege = 'SELECT'
 AND grantee IN ('PUBLIC');
```

(EN FAIT IL FAUT FAIRE UNE JOINTURE AVEC ALL_TABLES CAR ALL_TAB_PRIVS RENVOIE PLEIN D'AUTRES VUES)

- A quoi correspond la notion de session ? Quelles sont les notions qui vous semblent associées à session ?
  Une session est une connexion à la base de données. A chaque session nous sont attribués un SID, notre nom d'utilisateur, le type de serveur utilisé (dédié ou non), une adresse sql, la machine sur laquelle nous sommes connectés, etc.

- Que retourne la requête suivante ?

```
SELECT status, count(*) AS cxns
FROM v$session
GROUP BY status;
```

La requête renvoie le nombre de sessions actives et le nombre de sessions inactives sur la base de données.

1.4 Questions d'ordre général sur l'architecture

Exploitez les vues du dictionnaire de données vous permettant de répondre aux questions suivantes :

1. Sur quelle instance suis je connecté(e) ?

```
SELECT instance_name
FROM v$instance;
```

2. Que donne comme informations complémentaires la requête suivante ?

```
SELECT instance_name, host_name, version, startup_time
FROM v$instance;
```

La requête ci-dessus, en plus de préciser l'instance sur laquelle nous sommes connectés, nous informe sur le nom du serveur, la version de l'instance et sa date de création.

3. Sur quelle base de données pointe l'instance ? Le mode archivage est-il activé ?

```
SELECT instance_name, host_name, version, startup_time; archiver
FROM v$instance;
```

Le nom de l'instance est le même que celui de la base de données (ici licence). Le mode

archivage est stoppé.

4. Que donne comme informations complémentaires la requête suivante ?

```
ALTER SESSION
SET NLS_DATE_FORMAT = 'mm-dd-yyyy HH24:mi:ss';
SELECT name, created, log_mode, SCN_TO_TIMESTAMP(current_scn)
FROM v$database;
```

log_mode donne une information sur l'archivage des données contenues dans les fichiers log (ici NOARCHIVELOG indique que les données stockées dans les fichiers log sont au bout d'un moment éffacée et donc non archivées). SCN_TO_TIMESTAMP(current_scn) permet d'afficher "l'heure" de la base de données.

5. Traduisez en langue naturelle la requête suivante

```
SELECT addr, username, program
FROM v$process
WHERE addr IN (SELECT paddr
       FROM v$bgprocess
      );
```

Cette requette nous renvoie les adresses, de processus arrière plan (comme PMON, SMON, etc)

6. Quels sont les nom et taille des principales structures mémoire de la SGA rappeler brièvement les rôles dévolus à chacune de ces structures mémoires.

SHARED POOL (40% du SGA) (TAILLE = SOMME Fixed Size + Variable Size)

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 deux blocs :

- La Library Cache
- Le Dictionnary Cache

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 disponibles en 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.

DATABASE BUFFER CACHE (50 % du SGA) 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, entrainant 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. S'il n'est pas rappelé ou réutilisé il est alors déplacé lentement vers la sortie. Cependant s'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.

REDOLOG BUFFER (10 % du SGA)

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 qu'Oracle ne pourra écraser les données contenues dans ce buffer que si elles ont été écrites dans les fichiers REDOLOG FILE.

7. chassez l'intrus : laquelle de ces vues n'est pas associée à la "shared pool". Les vues dynamiques de performance associées à la mémoire partagée sont : `v$rowcache`, `v$librarycache`, `v$sqlarea` et `v$lock` ?

`v$rowcache` --> dictionnary cache (permet de gérer les objets utilisateurs "invalides" qui entravent les performences du dictionnary cache (par exemple création d'une table puis d'une vue sur la table puis suppression de la table. La vue reste alors présente dans le dictionnary cache mais est invalide puisque la table a été supprimée). Le dictionnary cache contient des métas donnés (par exemples des vues sur le noms des tables).

`v$lock` EST L'INTRU puisqu'il concerne les informations sur les verrous au niveau du database buffer cache

8. Comment afficher la taille du bloc de données ?

On retrouve cette information à plusieurs endroits, par exemple :

```
SELECT name, value
FROM v$parameter
WHERE name = 'db_block_size';
```

1.5 Organisation logique

1. A quoi correspond la notion d'espace de table (tablespace) ? En quoi se décompose-t-elle ?

c'est une organisation logique en structure cache. Elle se décompose en segment, puis en extensions de blocks puis en blocks de données

2. Quelle est la structure mémoire qui est la plus impactée par l'organisation en tablespaces ?

du coup ça doit être la Database Buffer Cache

3. Comment connaître le nom de l'espace de table dans lequel les tables de votre schéma utilisateur sont organisées ? idem pour les index ?

```
SELECT table_name, tablespace_name, blocks
FROM user_tables;
```

OU

```
SELECT table_name, tablespace_name, blocks
FROM dba_tables
WHERE owner = 'TBONDETTI';
```

Pour les indexes ::
la table user_indexes ne donne pas d'information sur la taille des index, il faut regarder la table dba_segments ou user_segments

```
SELECT segment_type, sum(blocks)
FROM user_segments
GROUP BY segment_type;
```

Si nous voulons plus d'informations sur les feuilles ou sur les branches des indexes, il faut raffraichir les stats et aller regarder dans v$index_stats

4. Quels sont les types de segments les plus habituels ?
table et index

```
SELECT segment_type, count(*) AS nombre
FROM dba_segments
GROUP BY segment_type
ORDER BY nombre DESC;
```

5. Comment savoir à quel fichier de données correspond un espace de table ?
Espace de table == contient l'ensemble des segments
segment -> correspond à une table, ou un index, ou autre

```
SELECT file_name, tablespace_name
FROM dba_data_files;
```

6. Quel est le paquetage qui vous renseigne sur le bloc de données qui contient un tuple précis
d'une table en particulier ?
ex : dbms_rowid (voir tp sur les index)

1.6 Library Cache

1. Donnez votre compréhension de la requête qui suit

```
set linesize 300
SELECT Sql_FullText, (cpu_time/100000) "Cpu Time (s)", (elapsed_time/1000000) "Elapsed time (s)", fetches, buffer_gets, disk_reads, executions
FROM v$sqlarea
WHERE Parsing_Schema_Name = USER
 AND ROWNUM < 50
ORDER BY 3 DESC;
```

Affiche les 50 différentes dernières requêtes executées par l'utilisateurs avec leurs temps d'execution en seconde le nombre de résultats ressortis à l'execution, le nombre de blocks qui ont été lu dans le datta buffer, le nombre de lectures sur le disque et le nombre de fois ou la requete a été exécutées.

2. Index

1. quel est le rôle d'un index ? Quelle est la structure d'index la plus exploitée dans les SGBDR ?

Le rôle d'un index est de permettre de retrouver rapidement les enregistrements possédant la propriété (search key) d'entrée dudit index. La structure d'index la plus exploitée dans les SGBDR est l'arbre B (B-tree).

2. Vous expliquerez l'ordre suivant :

```
set verify off
SELECT index_name
FROM dba_indexes
WHERE upper(table_name) = upper(&matable);
```

Retourne l'ensemble des noms des index sur lesquels l'utilisateur a des permissions et dont le nom de la table associée sera égal (sans considération de la casse) à la valeur de la variable `&matable` qui sera demandée à l'utilisateur lors de l'execution de la requête.

3. Vous disposez d'une table `EMP` qui comporte un attribut `NUM` sur lequel est posé une contrainte de clé primaire.

(a) `EMP` se voit appliquer un index unique ? (justifier votre réponse)

i. OUI

ii. NON

Lors de la création de la contrainte de clé primaire sur num, un index avec la search key num sera automatiquement créé. Toutefois, rien n'empêche d'avoir d'autres indexes sur la table.

(b) Des requêtes de consultation vous sont données. Lesquelles vont bénéficier de l'index ?

i. select num from emp ;

ii. select nom from emp where num = 20 ;

iii. select nom, fonction from emp ;

iv. select nom from emp where num > 10 ;

(c) remplacer `EMP` par une table indexée (IOT)

i. donner l'ordre de création de cette table IOT

ii. quelles seraient les requêtes précédentes qui seraient rendues encore plus efficaces par

cette nouvelle organisation ?

???

3. Transactions

Vous travaillerez sur les transactions et les verrous actifs.

1. Donnez votre compréhension de la requête qui suit

```
SELECT id1, id2, request, decode(lmode, 0, 'None(0)', 1, 'Null(1)', 2, 'Row Share(2)', 3, 'Row Exclu(3)', 4, 'Share(4)', 5, 'Share Row Ex(5)', 6, 'Exclusive(6)') lmode, sid
FROM v$lock;
```

Les colonnes id1 et id2 de la table v$lock permettent de mettre en correspondance les couples session bloquante - session bloquée (qui partagent ces mêmes identifiants, corrélés au segment d'annulation exploité de manière sous-jacente). La session bloquée est dans l'impossibilité de voir sa requête exécutée, et de sorte sa requête (colonne request) a une valeur supérieure à 0 (numéro correspondant au type de verrou demandé).

Il est à noter que l'attribut block prend la valeur 1, lorsque la session détient un verrou qui est en train de bloquer une autre transaction (et 0 dans le cas contraire). L'attribut lmode indique le type de verrou détenu par la transaction et a pour domaine de valeurs :

0 -> None; **1 -> Null** : Select; **2 -> Row Share** : les accès concurrents peuvent exploiter la ressource, mais aucun verrou exclusif ne peut être posé sur la table (dans son

intégralité) (posé de manière automatique avec un ordre select ... for update); **3 -> Row Exclusive** : posé de manière automatique avec tout ordre update, delete et insert; **4 -> Share** : acquis avec l'ordre LOCK TABLE <table> IN SHARE MODE ; les modifications pouvant être apportées par les autres transactions ne sont pas admises; **5 -> Share Row Exclusive** : plus restrictif que Share, ce mode ne permet pas aux transactions de poser un verrou en mode Share sur la table considérée; **6 -> Exclusive** : une seule transaction est autorisée à poser un verrou exclusif sur une table, les autres transactions doivent alors se contenter de simples lectures de la table.

La requête affiche les différents verrous en cours sur le serveur (id1 correspondant à la session bloquante et id2 à la session bloquée), decode(...) permet d'afficher le type de verrou de manière littéraire (et non numérique).

2. Expliquer la séquence suivante (placez vous en début de transaction)

```
SELECT *
FROM employe;
SELECT username, v$lock.sid, id1, id2, lmode, request, block,
v$lock.type
FROM v$lock, v$session
WHERE v$lock.sid = v$session.sid
   AND v$session.username = USER;
```

3. Expliquer la nouvelle séquence

```
INSERT INTO employe (num)
VALUES (55555);
SELECT username, v$lock.sid, id1, id2, lmode, request, block,
v$lock.type
FROM v$lock, v$session
WHERE v$lock.sid = v$session.sid
 AND v$session.username = USER;
```

4. Surcouche procédurale PL/SQL

4.1 Déclencheurs

Trois ordres de déclencheurs vous sont donnés. Un seul ordre est correct. Vous indiquerez lequel et vous expliquerez pourquoi les deux autres ne le sont pas.

```
CREATE OR REPLACE TRIGGER t1
BEFORE UPDATE ON commune
BEGIN
 :new.nom_com := lower(:old.nom_com) ;
END t1;
/
```

```
CREATE OR REPLACE TRIGGER t1
BEFORE UPDATE ON commune
FOR EACH ROW
BEGIN
 :new.nom_com := upper(:old.nom_com) ;
END t1;
/
```

```
CREATE OR REPLACE TRIGGER t1
AFTER UPDATE ON commune
FOR EACH ROW
BEGIN
 :new.nom_com := upper(:old.nom_com) ;
END t1;
/
```

En premier lieu, nous pouvons éliminer la troisième proposition, puisque l'ordre `AFTER` rend impossible la modification des valeurs des attributs des tuples considérés dans les corps du déclencheur.
Deuxièmement, nous pouvons éliminer la première proposition puisque les valeurs `:new.nom_com` et `:old.nom_com` ne sont définis que pour des tuples et non pour des ensemble de tuples (Il manque ici le `FOR EACH ROW` avant le `BEGIN`).

C’est donc la deuxième réponse qui est correcte.

5. Fonctions et procédures

Expliquez la différence entre une procédure et une fonction ?
Répondez au préalable aux questions suivantes (plusieurs réponses possibles).

1. Les procédures et fonctions doivent-elles retourner des valeurs ?
Les procédures non, les fonctions oui.

2. Les procédures et fonctions peuvent-elles retourner des valeurs ?
Les fonctions obligatoirement, les procédures peuvent modifiers des valeurs qui leurs sont passées en paramètre, mais ne retournent pas de valeurs..

3. Les procédures peuvent-elles retourner des valeurs ?
Non, elles peuvent par contre modifier certaines des variables qui leurs sont passées en paramètre.

4. Les fonctions doivent-elles retourner des valeurs ?
Oui.

Les fonctions sont semblables aux procédures, mais retournent une valeur résultat. Une fonction diffère d'une procédure par le fait qu'on peut l'utiliser dans une expression.

6. Bases de données réparties (et solution Db Link Oracle)

6.1 Question 1

Expliquer les grands principes de l’approche ascendante. A cet effet, vous définirez la notion de schéma global et de vue intégrante (vous pouvez vous aider d’un schéma et/ou d’un exemple illustratif).

6.2 Question 2

Quelle est la différence entre une vue et une vue matérialisée ? Quel peut être l'apport d'une vue matérialisée dans le contexte des bases de données réparties ? Vous expliquerez les ordres suivants :

```
CREATE DATABASE LINK lk_M
CONNECT TO user1
IDENTIFIED BY user1
USING 'licence';


CREATE MATERIALIZED VIEW test AS
SELECT *
FROM commune@lk_M;
```

6.3 Question 3

Nous nous concentrons sur la base de données relative aux communes françaises déjà étudiée en TP. Les relations Commune, Departement et Region sont à l'origine dans une seule base de données. Chaque nouvelle grande région française souhaite disposer des informations concernant les communes sous leur autorité. Vous ferez une proposition de fragmentation de la relation Commune dans un contexte d'approche descendante qui répond à cette relative autonomie (donnez le schéma de fragmentation).

7. Exercices sur l'énoncé : la parfumerie

7.1 Énoncé

La base de données considérée, s'attache à décrire le monde du parfum. La gestion qui en est faite, est très partielle, et porte sur les types d'entité suivants. Un type d'entité **`Marque`** est identifié par un nom de marque, et est caractérisé par un siège social et un attribut nommé couturier qui indique si la marque est aussi une maison de couture (valeurs parmi l'ensemble `{o, n}` pour oui ou non). Un type d'entité **`Parfum`** est

identifié par les attributs nom de parfum et nom de marque, et est caractérisé par un nom de créateur, une date de création, un public cible (valeurs parmi l'ensemble `{homme,femme,mixte}`), et un type de parfum (valeurs parmi l'ensemble `{boise, floral, oriental, hesperide, fougere, ambre, chypre, cuir}`). Un type d'entité **`Fragrance`** (odeur agréable) est identifié par son nom et est caractérisé par une origine (valeurs parmi l'ensemble `{animal, vegetal, synthetique}`. Un parfum se compose de plusieurs fragrances.

7.2 Schémas EA et Relationnel

Les schémas conceptuel et relationnel vous sont donnés. Un jeu de tuples est fourni, en annexe, à titre illustratif.
Les attributs portant les contraintes de clés primaires sont en gras. Les contraintes de clés étrangères vous sont données sous la forme de contraintes d'inclusion. Les types des attributs vous sont également indiqués.

- `Marque(`**`nomM varchar(15)`**`, siegeSocial varchar(15), couturier char(1))`
- `Parfum(`**`nomP varchar(10)`**`, nomM varchar(15), nomCreateur varchar(15), dateCreation date, cible varchar(8), typeP varchar(12))`
  avec `Parfum(nomM)⊆Marque(nomM)`
- `Fragrance(`**`nomF varchar(20)`**`, origine varchar(15))`
- `ComposeDe(`**`nomP varchar(10), nomM varchar(15), nomF varchar(20)`**`)`
  avec `ComposeDe(nomF)⊆Fragrance(nomF)`
  avec `ComposeDe(nomP, nomM)⊆Parfum(nomP, nomM)`

7.3 Exercice optimisation

7.4 Index

Vous répondrez (avec une justification courte) aux questions suivantes.
1. Quels sont les index déjà présents au sein du schéma de la base de données, et sur quels attributs sont ils implicitement posés ?
Il y a exactement quatre index créés lors de la création de la base de données : un pour chacune des clés primaires des différentes tables. Nous les nommerons respectivement `Marque_PK, Parfum_PK, Fragrance_PK` et `ComposeDe_PK. Marque_PK` est posé

sur l'attribut `nomM`. `Parfum_PK` est posé sur l'attribut `nomP`. `Fragrance_PK` est posé sur l'attribut `nomF`. `ComposeDe` est posé sur le triplet d'attributs `(nomP, nomM, nomF)`.

2. Sur quels autres attributs poseriez vous d'autres index ? Donnez les ordres de création d'index correspondants.

Sur les clés étrangères ?

3. Expliquez la requête suivante et les résultats obtenus.

```
SELECT index_name, uniqueness, table_name, num_rows
FROM user_indexes;


INDEX_NAME         UNIQUENESS TABLE_NAME          NUM_ROWS
------------------ ---------- ---------------- -----------
COMPOSEDE_PK       UNIQUE COMPOSEDE                   11
FRAGRANCE_PK       UNIQUE FRAGRANCE                   13
PARFUM_PK          UNIQUE PARFUM                      2
MARQUE_PK          UNIQUE MARQUE                      2
```

La requête retourne le noms des indexes de l'utilisateur ainsi que leur unicité, le nom de leurs tables associées et le nombre de clés qu'ils possèdent.

7.5 Plan d'exécution d'une requête

1. Expliquez l'intérêt de la requête suivante et décrivez le plan d'exécution.

```
EXPLAIN PLAN FOR
SELECT /*+ use_nl(p m) */ m.nomM
FROM marque m, parfum p
WHERE m.nomM=p.nomM;


SELECT plan_table_output
FROM table (dbms_xplan.display());


Plan hash value: 3812789126
-------------------------------------------------------------------------------------
|  Id | Operation              | Name | Rows | Bytes | Cost (%CPU) |     Time |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT       | | 2 | 36 |       2 (0) | 00:00:01 |
|   1 | NESTED LOOPS           | | 2 | 36 |       2 (0) | 00:00:01 |
|   2 |  INDEX FAST FULL SCAN | PARFUM_PK |    2 | 18 | 2 (0) | 00:00:01 |
| * 3 |   INDEX UNIQUE SCAN    | MARQUE_PK | 1 |    9 | 0 (0) | 00:00:01 |
-------------------------------------------------------------------------------------
PLAN_TABLE_OUTPUT
----------------------------------------------------
Predicate Information (identified by operation id):
----------------------------------------------------
3 - access("M"."NOMM"="P"."NOMM")
```

La première requête demande à Oracle de calculer le plan pour la requête `SELECT m.nomM FROM marque m, parfum p WHERE .nomM = p.nomM;` (sélection des noms des marques qui apparaissent à la fois dans la table `Marque` et la table `Parfum`) en utilisant un `NESTED LOOP`, c'est à dire, en forçant la comparaison ligne par ligne. La deuxième requête affiche le résultat du précédent calcul. Nous pouvons voir qu'en effet, nous avons un `NESTED LOOP`. Au lieu de parcourir directement la table `Parfum`, Oracle utilise l'index posé sur `nomP` pour accéder aux différents tuples de la table `Parfum`. Pour chacun de ces tuples, Oracle compare la valeur de l'attribut `nomM` du tuple avec les attributs `nomM` des tuples de la table `Marque`. Pour cela, il utilise l'index posé sur l'attribut `nomM` de la table `Marque`.

La lecture du plan d'exécution indique la présence de boucles imbriquées après accès à la table Marque via l'index nomM et accès aux index nomM de la table Parfum
A COMPLETER

2. même question pour la requête suivante

```
EXPLAIN PLAN FOR
SELECT /*+use_hash(m p) */ m.nomM
FROM marque m, parfum p
WHERE m.nomM=p.nomM;


SELECT plan_table_output
FROM table (dbms_xplan.display()) ;
Plan hash value: 1371628531
-------------------------------------------------------------------------------
---
|  Id | Operation          | Name | Rows | Bytes | Cost (%CPU) |     Time |
-------------------------------------------------------------------------------
---
|   0 | SELECT STATEMENT   | | 2 | 52 |      3 (34) | 00:00:01 |
| * 1 | HASH JOIN          | | 2 | 52 |      3 (34) | 00:00:01 |
|   2 |  INDEX FULL SCAN   | MARQUE_PK | 2 |    26 | 1 (0) | 00:00:01 |
|   3 |  INDEX FULL SCAN   | PARFUM_PK | 2 |    26 | 1 (0) | 00:00:01 |
-------------------------------------------------------------------------------
---
PLAN_TABLE_OUTPUT
----------------------------------------------------
Predicate Information (identified by operation id):
----------------------------------------------------
1 - access("M"."NOMM"="P"."NOMM")
```

La première requête demande à Oracle de calculer le plan pour la requête `SELECT m.nomM FROM marque m, parfum p WHERE .nomM = p.nomM;` (sélection des noms des marques qui apparaissent à la fois dans la table `Marque` et la table `Parfum`) en utilisant un `NESTED LOOP`, c'est à dire, en forçant la comparaison ligne par ligne. La deuxième requête affiche le résultat du précédent calcul. Nous pouvons voir qu'en effet, nous avons un `NESTED LOOP`. Au lieu de parcourir directement la table `Parfum`, Oracle utilise l'index posé sur `nomP` pour accéder aux différents tuples de la table `Parfum`. Pour chacun de ces tuples, Oracle compare la valeur de l'attribut `nomM` du tuple avec les attributs `nomM` des tuples de la table `Marque`. Pour cela, il utilise l'index posé sur l'attribut `nomM` de la table `Marque`.
INDEX FULL SCAN permet d'éviter d'aller interroger dans les blocs de données car les blocs d'index suffisent

8. Exercice transaction

Vous répondrez avec brièveté aux questions suivantes :

1. Une séquence d'ordres SQL vous est fourni. Vous définirez le rôle de l'ordre rollback en général et en expliquerez les effets sur la séquence proposée. Quels sont les ordres SQL impactés et pourquoi ?

```
INSERT INTO Fragrance
VALUES ('violette','vegetal');
INSERT INTO Fragrance
VALUES ('verveine','vegetal');
ALTER TABLE FRAGRANCE
ADD prix float;
INSERT INTO Fragrance
VALUES ('musc nitre','animal',100);
UPDATE Fragrance
SET origine='synthetique'
WHERE nomF='musc nitre';
ROLLBACK;
```

2. Quelles sont les propriétés caractéristiques d'une session dont le début est marqué par l'ordre SQL suivant ?

```
ALTER SESSION
SET ISOLATION_LEVEL = SERIALIZABLE;
```

3. À partir du schéma de l'examen, proposez un exemple d'étreinte mortelle (deadlock) pouvant survenir entre deux transactions concurrentes.

9. Exercice PL/SQL

Vous donnerez l'écriture d'un déclencheur d'avertissement qui renvoie le message : *odeur trop*
forte si un parfum se compose de plus de deux fragrances d'origine animale (déclencheur portant sur l'insertion et la mise à jour de tuples dans la table `composeDe`).