Il sera tenu compte de la clarté des solutions qui seront données. Document non autorisés.

**Questions de cours**

1- Quelle est la vue du dictionnaire de données Oracle qui permet de donner (requête explicite):

a. la liste des Les contraintes d’un utilisateur donné

b. la liste de tous les objets accessibles par un utilisateur donné.

c. toutes les procédures créées dans une instance donnée

d. la liste des vues créées par un utilisateur

1- Définir une instance Oracle

2- Indiquez ci-dessous cinq processus d'arrière-plan obligatoires et le processus facultatif

3- Peut-on se connecter au compte SYS sans fournir le mot de passe, si oui pourquoi

4- Comment utilise t-on l’exception raise\_application\_error

5- Comment peut-on savoir l'intérêt de la vue du dictionnaire de données USER\_COL\_PRIVS

**Problème**

On désire mettre en place un paquetage logiciel permettant de gérer la table

**PILOTE (PlNum , PlNom, PlPrenom, Ville, Salaire, Nbannées)**

. L’objectif est de disposer de procédures permettant de :

•afficher le contenu de la table au format Numéro : Prénom NOM (Ville) – Salaire ;

•ajouter un pilote ; •supprimer un pilote (connaissant son numéro) ; •modifier un pilote ; ...

1. Donnez la liste des pilotes qui ont assuré plus de 500h de vols [table supplémentaire

**VOL (Num\_vol, PlNum, NumVol, NbH)**]

Select p.plnum, p.plnom, p.plprenom, p.ville, p.salaire

From pilote p

Join vol v on p.plnum = v.plnum

Group by p.plnum, p.plnom, p.plprenom, p.ville, p.salaire

Having sum(v.nbh) > 500;

1. Définir les spécifications d’un paquetage nommé « pilotes\_pack » contenant:

a. une procédure nommée « afficher » (sans paramètre) permettant d’afficher les pilotes à l’écran au format désiré.

b. une procédure nommée « ajouter » permettant d’ajouter dans la table PILOTE un pilote dont les numéro, nom, prénom, ville et salaire sont passés en paramètres. Tester la procédure.

c. une procédure nommée « supprimer »permettant de supprimer de la table PILOTE un

pilote dont le numéro est passé en paramètre. Tester la procédure.

d. une fonction qui fournit le nombre d'heures de vol pour un pilote donné

e. Ecrire une fonction PL/SQL permettant de retourner le salaire moyen de tous les employés d’un département donné qui ne sont pas de managers.

2. Créer un trigger qui permet –avant de supprimer un pilote- de vérifier si Nbannées (nb

d’années de travail) est supérieur à 7.

3. Créer un trigger qui permet de supprimer tous les vols après suppression d'un pilote donné

4. Pour archiver l’historique des opérations de mise à jour de la table PILOTE, on a créé une table **History (PlNum, operation\_date, operation\_type)**. Ecrire un trigger permettant, après l’exécution de chaque opération sur PILOTE, d’ajouter un tuple concernant cette opération à la table History

Pour répondre à vos questions concernant le cours sur Oracle, voici des réponses détaillées :

**### 1. Vue du dictionnaire de données Oracle**

**a. \*\*La liste des contraintes d’un utilisateur donné\*\***

Utilisez la vue `ALL\_CONSTRAINTS` pour obtenir les contraintes d'un utilisateur donné :

```sql

SELECT \* FROM ALL\_CONSTRAINTS WHERE OWNER = 'NOM\_UTILISATEUR';

```

**b. \*\*La liste de tous les objets accessibles par un utilisateur donné\*\***

Utilisez la vue `ALL\_OBJECTS` pour lister tous les objets accessibles par un utilisateur :

```sql

SELECT \* FROM ALL\_OBJECTS WHERE OWNER = 'NOM\_UTILISATEUR';

```

**c. \*\*Toutes les procédures créées dans une instance donnée\*\***

Utilisez la vue `ALL\_PROCEDURES` pour lister toutes les procédures créées dans une instance :

```sql

SELECT \* FROM ALL\_PROCEDURES;

```

**d. \*\*La liste des vues créées par un utilisateur\*\***

Utilisez la vue `ALL\_VIEWS` pour obtenir la liste des vues créées par un utilisateur :

```sql

SELECT \* FROM ALL\_VIEWS WHERE OWNER = 'NOM\_UTILISATEUR';

```

**### 2. Définir une instance Oracle**

Une instance Oracle est l'ensemble des processus en arrière-plan et de la mémoire partagée qui permet d'accéder à une base de données Oracle. Elle se compose principalement du SGA (System Global Area) et des processus d'arrière-plan qui gèrent les tâches comme la gestion de la mémoire, l'écriture des données sur disque et la récupération en cas de panne.

**### 3. Cinq processus d'arrière-plan obligatoires et un processus facultatif**

\*\*Processus d'arrière-plan obligatoires :\*\*

1. \*\*DBWn (Database Writer)\*\* : écrit les modifications de la mémoire tampon vers le disque.

2. \*\*LGWR (Log Writer)\*\* : écrit les entrées du journal des transactions (redo log) sur le disque.

3. \*\*CKPT (Checkpoint)\*\* : synchronise les fichiers de données et les fichiers de contrôle.

4. \*\*SMON (System Monitor)\*\* : effectue la récupération après un crash et nettoie les segments temporaires.

5. \*\*PMON (Process Monitor)\*\* : nettoie les processus utilisateur défaillants et libère les ressources.

\*\*Processus facultatif :\*\*

- \*\*ARCn (Archiver)\*\* : copie les fichiers de journal des transactions pour la sauvegarde et la récupération (utilisé dans les configurations de journalisation archivées).

**### 4. Peut-on se connecter au compte SYS sans fournir le mot de passe ?**

Oui, il est possible de se connecter au compte SYS sans fournir le mot de passe en utilisant l'authentification de l'OS, par exemple avec l'utilisateur Oracle sous Unix/Linux :

```bash

sqlplus / as sysdba

```

Cette méthode fonctionne car l'utilisateur Oracle est reconnu par le système d'exploitation comme ayant les privilèges appropriés pour accéder à la base de données en tant que SYSDBA.

**### 5. Utilisation de l'exception `raise\_application\_error`**

`raise\_application\_error` est utilisé pour lever des exceptions définies par l'utilisateur dans PL/SQL. Il permet de générer une erreur avec un message personnalisé et un code d'erreur spécifique. Voici un exemple :

```plsql

BEGIN

IF some\_condition THEN

raise\_application\_error(-20001, 'Erreur personnalisée : condition non remplie');

END IF;

END;

```

Dans cet exemple, si `some\_condition` est vraie, une erreur avec le code `-20001` et le message "Erreur personnalisée : condition non remplie" sera levée.

**### 6. Intérêt de la vue du dictionnaire de données `USER\_COL\_PRIVS`**

La vue `USER\_COL\_PRIVS` fournit des informations sur les privilèges de colonnes accordés à l'utilisateur. Elle est utile pour vérifier les permissions spécifiques sur les colonnes, telles que les privilèges de SELECT, INSERT, UPDATE ou DELETE. Cette vue permet de voir les privilèges détaillés qui ont été accordés sur des colonnes individuelles dans les tables accessibles à l'utilisateur.

Pour mettre en place le paquetage logiciel permettant de gérer la table PILOTE, nous allons d'abord définir les spécifications du paquetage `pilotes\_pack` et ensuite créer les triggers nécessaires. Nous allons utiliser le langage PL/SQL pour cela.

**### 1. Liste des pilotes ayant assuré plus de 500 heures de vol**

Pour obtenir cette liste, nous utiliserons une requête SQL qui joint les tables PILOTE et VOL, agrège les heures de vol par pilote et filtre ceux ayant plus de 500 heures de vol.

```sql

SELECT p.PlNum, p.PlNom, p.PlPrenom

FROM PILOTE p

JOIN VOL v ON p.PlNum = v.PlNum

GROUP BY p.PlNum, p.PlNom, p.PlPrenom

HAVING SUM(v.NbH) > 500;

```

**### 2. Spécifications du paquetage `pilotes\_pack`**

**#### a. Procédure `afficher`**

```sql

CREATE OR REPLACE PACKAGE pilotes\_pack AS

PROCEDURE afficher;

PROCEDURE ajouter(p\_num IN NUMBER, p\_nom IN VARCHAR2, p\_prenom IN VARCHAR2, p\_ville IN VARCHAR2, p\_salaire IN NUMBER);

PROCEDURE supprimer(p\_num IN NUMBER);

FUNCTION nb\_heures\_vol(p\_num IN NUMBER) RETURN NUMBER;

FUNCTION salaire\_moyen\_dept(p\_deptno IN NUMBER) RETURN NUMBER;

END pilotes\_pack;

/

CREATE OR REPLACE PACKAGE BODY pilotes\_pack AS

PROCEDURE afficher IS

CURSOR c\_pilotes IS

SELECT PlNum, PlPrenom, PlNom, Ville, Salaire

FROM PILOTE;

pilote\_rec c\_pilotes%ROWTYPE;

BEGIN

OPEN c\_pilotes;

LOOP

FETCH c\_pilotes INTO pilote\_rec;

EXIT WHEN c\_pilotes%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE(pilote\_rec.PlNum || ' : ' || pilote\_rec.PlPrenom || ' ' || pilote\_rec.PlNom || ' (' || pilote\_rec.Ville || ') - ' || pilote\_rec.Salaire);

END LOOP;

CLOSE c\_pilotes;

END afficher;

PROCEDURE ajouter(p\_num IN NUMBER, p\_nom IN VARCHAR2, p\_prenom IN VARCHAR2, p\_ville IN VARCHAR2, p\_salaire IN NUMBER) IS

BEGIN

INSERT INTO PILOTE (PlNum, PlNom, PlPrenom, Ville, Salaire, Nbannées)

VALUES (p\_num, p\_nom, p\_prenom, p\_ville, p\_salaire, 0); -- Nbannées initialisé à 0

END ajouter;

PROCEDURE supprimer(p\_num IN NUMBER) IS

BEGIN

DELETE FROM PILOTE WHERE PlNum = p\_num;

END supprimer;

FUNCTION nb\_heures\_vol(p\_num IN NUMBER) RETURN NUMBER IS

total\_heures NUMBER;

BEGIN

SELECT SUM(NbH) INTO total\_heures

FROM VOL

WHERE PlNum = p\_num;

RETURN NVL(total\_heures, 0);

END nb\_heures\_vol;

FUNCTION salaire\_moyen\_dept(p\_deptno IN NUMBER) RETURN NUMBER IS

avg\_salaire NUMBER;

BEGIN

SELECT AVG(Salaire) INTO avg\_salaire

FROM EMP

WHERE Deptno = p\_deptno AND Job != 'MANAGER';

RETURN NVL(avg\_salaire, 0);

END salaire\_moyen\_dept;

END pilotes\_pack;

/

```

**#### Test des procédures**

Pour tester les procédures, on peut utiliser des blocs PL/SQL anonymes comme suit :

```sql

BEGIN

pilotes\_pack.ajouter(1, 'Doe', 'John', 'Paris', 50000);

pilotes\_pack.ajouter(2, 'Smith', 'Jane', 'Lyon', 55000);

pilotes\_pack.afficher;

pilotes\_pack.supprimer(1);

pilotes\_pack.afficher;

END;

/

```

**### 3. Création des triggers**

**#### a. Trigger pour vérifier Nbannées avant de supprimer un pilote**

```sql

CREATE OR REPLACE TRIGGER check\_nbannées\_before\_delete

BEFORE DELETE ON PILOTE

FOR EACH ROW

BEGIN

IF :OLD.Nbannées <= 7 THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Cannot delete pilot with 7 or fewer years of service.');

END IF;

END;

/

```

**#### b. Trigger pour supprimer les vols après suppression d'un pilote**

```sql

CREATE OR REPLACE TRIGGER delete\_vols\_after\_pilote

AFTER DELETE ON PILOTE

FOR EACH ROW

BEGIN

DELETE FROM VOL WHERE PlNum = :OLD.PlNum;

END;

/

```

**#### c. Trigger pour archiver l'historique des opérations**

```sql

CREATE OR REPLACE TRIGGER archive\_operations\_on\_pilote

AFTER INSERT OR UPDATE OR DELETE ON PILOTE

FOR EACH ROW

BEGIN

INSERT INTO History (PlNum, operation\_date, operation\_type)

VALUES (:OLD.PlNum, SYSDATE, CASE

WHEN INSERTING THEN 'INSERT'

WHEN UPDATING THEN 'UPDATE'

WHEN DELETING THEN 'DELETE'

END);

END;

/

```

Ces spécifications et codes devraient fournir un bon point de départ pour la gestion de la table PILOTE et l'implémentation des diverses procédures et triggers nécessaires pour la maintenance des données et la traçabilité des opérations.