

Secteur Tertiaire Informatique
Filière étude - développement

Activité « Développer la persistance des données »

Cas PAPYRUS

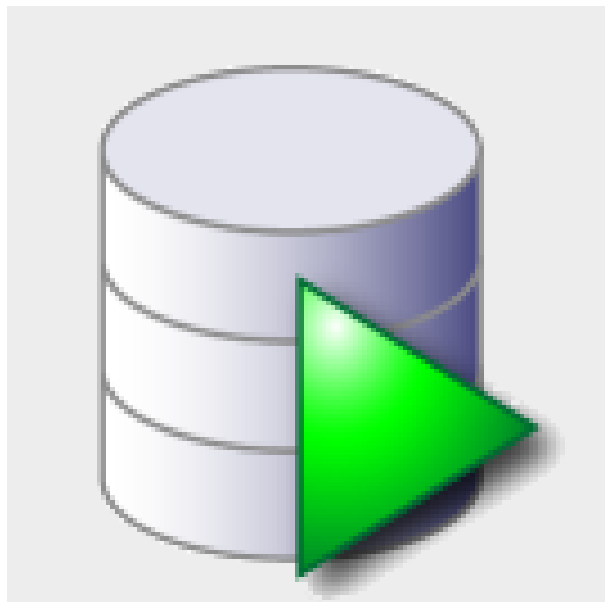
Programmer dans le langage du SGBD ORACLE

Accueil

Apprentissage

Période en
entreprise

Evaluation



Code barre

SOMMAIRE

I	L'EXISTANT.....	3
II	LES FONCTIONS	4
II.1	Première fonction.....	4
II.2	Création d'une fonction scalaire	4
II.3	Création d'une fonction table en ligne.....	5
III	LES PROCEDURES STOCKEES	6
III.1	Création d'une procédure sans paramètre	6
III.2	Création d'une procédure avec un paramètre en entrée	7
III.3	Création d'une procédure avec des paramètres en entrée et en sortie	8
III.4	Création d'une fonction avec des paramètres en entrée et en sortie et une gestion d'erreur utilisateur	9
IV	LES DECLENCHEURS	10
IV.1	Création d'un déclencheur AFTER INSERT, UPDATE et DELETE	11
IV.2	Modification d'un déclencheur AFTER UPDATE	12
IV.3	Création d'un déclencheur INSERTION sur la table PRODUIT	14
IV.4	Création d'un déclencheur LDD (DDL)	15
IV.5	Création d'un déclencheur pour gérer les clés primaires.....	17

I L'EXISTANT

La base de données relationnelle PAPYRUS est constituée des tables suivantes :

```
PRODUIT (CODART, LIBART, STKALE, STKPHY, QTEANN, UNIMES)
COMMANDE (NUMCOM, #NUMFOU, OBSCOM, DATECOM)
LIGCOM (#NUMCOM, #CODART, NUMLIG, QTECDE, PRIXUNI, QTELIV, DERLIV)
FOURNISSEUR (NUMFOU, NOMFOU, RUEFOU, POSFOU, VILFOU, CONFOU, SATISF)
VENTE (#CODART, #NUMFOU, DELLIV, QTE, PRIX)
```

Convention : les clés primaires sont soulignées. Les clés étrangères sont précédées du caractère #.

Le jeu d'essais a été constitué lors d'une séance précédente.

II LES FONCTIONS

II.1 PREMIERE FONCTION

Pour bien comprendre l'utilité d'une fonction, créez la fonction **FN_DATEFORMAT** codée ci-joint, et utilisez la dans la requête 6 de la séance précédente qui affichait les commandes d'un mois donné.

```
CREATE or REPLACE FUNCTION FN_DATEFORMAT (pdate DATE, psep VARCHAR)
RETURN VARCHAR2
AS
BEGIN
    RETURN
        EXTRACT (day from pdate)
        || psep || EXTRACT (month from pdate)
        || psep || EXTRACT (year from pdate) ;
END FN_DATEFORMAT;
```


II.2 CREATION D'UNE FONCTION SCALAIRE

Créez la fonction scalaire **FN_SATISFACTION**, qui avec l'indice de satisfaction en paramètre d'entrée, affiche un niveau de satisfaction en clair :

- Indice = Null → 'Sans commentaire'
- Indice = 1 ou 2 → 'Mauvais'
- Indice = 3 ou 4 → 'Passable'
- Indice = 5 ou 6 → 'Moyen'
- Indice = 7 ou 8 → 'Bon'
- Indice = 9 → 'Excellent'

Squelette de la fonction : **en rouge le code à écrire.**

```
CREATE or REPLACE FUNCTION FN_SATISFACTION (PSATISF IN NUMBER)
RETURN CODE
IS
BEGIN
    code
    RETURN CODE;
END;
```

 Testez votre fonction à l'aide de la requête ci-dessous qui permet d'afficher le niveau de satisfaction des fournisseurs.

```
SELECT NOMFOU AS FOURNISSEUR,
FN_SATISFACTION(SATISF) AS "Indice de satisfaction"
FROM FOURNISSEUR;
```

II.3 CREATION D'UNE FONCTION TABLE EN LIGNE


Créez la fonction **FN_CA_FOURNISSEUR**, qui en fonction d'un **code fournisseur** et d'une **année** entrés en paramètre, restituera le CA potentiel de ce fournisseur pour l'année souhaitée.


Squelette de la fonction : **en rouge le code à écrire.**

```
CREATE OR REPLACE FUNCTION FN_CA_FOURNISSEUR_1 (P_CODEFOURNIS
NUMBER, P_ANNEE NUMBER)
RETURN CODE
IS
    CODE
BEGIN
    CODE
    RETURN CODE;
END;
```

La requête utilisée au sein de la fonction sera construite à partir de la requête ci-dessous (requête 19 de la séance précédente).

```
SELECT FOURNISSEUR.numfou AS "NUMERO FOURNISSEUR",
       nomfou AS "NOM FOURNISSEUR",
       SUM(QTELIV * (priuni * 1.2060)) AS "CHIFFRE D AFFAIRE"
FROM FOURNISSEUR
JOIN COMMANDE ON FOURNISSEUR.numfou = COMMANDE.numfou
JOIN LIGCOM ON COMMANDE.numcom = LIGCOM.numcom
WHERE (to_char(datecom, 'yyyy') = '2007')
GROUP BY FOURNISSEUR.numfou, nomfou;
```

 **Indice** : Utilisation de l'opérateur INTO pour mémoriser le chiffre d'affaire dans une variable déclarée au sein de la fonction. Cette variable est retournée par la fonction (RETURN).

 Exécutez cette procédure pour vérifier qu'elle fonctionne conformément à votre attente.

Test de la fonction avec un bloc anonyme : **en rouge le code à écrire.**

```
BEGIN
    -- Fournisseur existe
    -- Appel de la fonction
    -- Affichage du résultat

    -- Fournisseur n'existe pas
    -- Appel de la fonction
    -- Affichage du résultat
END;
```


III LES PROCEDURES STOCKEES


III.1 CREATION D'UNE PROCEDURE SANS PARAMETRE

Créez la procédure stockée **LST_FOURNIS** correspondant à la requête n°2 de la séance requête SQL, « Afficher le code des fournisseurs pour lesquels une commande a été passée ».

Squelette de la fonction : **en rouge le code à écrire.**

```
CREATE OR REPLACE PROCEDURE LST_FOURNIS
IS
    CODE
BEGIN
    CODE
END;
```

 Indice : Utilisation d'un curseur (CURSOR). Il est nécessaire d'utiliser un curseur car la requête SELECT utilisée retourne plusieurs enregistrements. Cf. document *CDI JEE - Langage PL-SQL.pdf*

 Exécutez cette procédure pour vérifier qu'elle fonctionne conformément à votre attente.

Test de la procédure avec un bloc anonyme : **en rouge le code à écrire.**

```
BEGIN
    -- Appel de la procédure
END;
```


III.2 CREATION D'UNE PROCEDURE AVEC UN PARAMETRE EN ENTREE

Créez la procédure stockée **LST_COMMANDES**, qui liste les commandes ayant un libellé particulier (exemple : urgent) dans le champ OBSCOM de la table COMMANDE. Le libellé est passé en paramètre de la procédure.


Cette requête sera construite à partir de la requête 11 de la séance requête SQL.

Squelette de la fonction : **en rouge le code à écrire.**

```
CREATE OR REPLACE PROCEDURE LST_COMMANDES (V_OBS VARCHAR2)
IS
    CODE
BEGIN
    CODE
END;
```

 Indice : Utilisation d'un curseur (CURSOR). Il est nécessaire d'utiliser un curseur car la requête SELECT utilisée retourne plusieurs enregistrements.

 Cf. document *CDI JEE - Langage PL-SQL.pdf*

 Exécutez cette procédure pour vérifier qu'elle fonctionne conformément à votre attente.

Test de la procédure avec un bloc anonyme.


```
BEGIN
    -- Appel de la procédure
END;
```

III.3 CREATION D'UNE PROCEDURE AVEC DES PARAMETRES EN ENTREE ET EN SORTIE

Créez une procédure nommée **GET_AMOUNT_LAST_ORDER** qui retourne le montant de la dernière commande d'un fournisseur donné.
Le montant est un paramètre de sortie de la procédure.
Le fournisseur est un paramètre en entrée de la procédure.


Squelette de la procédure : **en rouge le code à écrire.**

```
CREATE or REPLACE PROCEDURE
GET_MONTANT_LAST_COMMANDE (p_numfou IN commande.numfou%TYPE,
                           p_montant OUT NUMBER)
AS
CODE
BEGIN
CODE
END;
```

 Indice : Utilisation de **OUT** pour définir qu'un paramètre est en sortie de la procédure et de **IN** pour définir qu'un paramètre est en entrée de la procédure.

 Cf. document *CDI JEE - Langage PL-SQL.pdf*

Question : Le type utilisé pour le paramètre p_numfou est commande.numfou%TYPE. A quoi correspond ce type ?

 Exécutez cette procédure pour vérifier qu'elle fonctionne conformément à votre attente.

Bloc anonyme pour tester la procédure : **en rouge le code à écrire.**

```
DECLARE
v_montant NUMBER;
v_numfou NUMBER := 120;
BEGIN
-- Appel de la procédure
-- Affichage du résultat
END;
```


III.4 CREATION D'UNE FONCTION AVEC DES PARAMETRES EN ENTREE ET EN SORTIE ET UNE GESTION D'ERREUR UTILISATEUR

Reprenez la procédure précédente.

A partir de cette procédure, créez une fonction nommée

GET_AMOUNT_LAST_ORDER_EX.


Ajoutez un traitement d'erreur si le code fournisseur (numfou) est inconnu dans la table FOURNISSEUR.

La fonction retourne (RETURN de la fonction) :


- un code égal à 5050 en cas d'erreur (fournisseur inconnu),
- null si pas d'erreur (fournisseur est connu).

Squelette de la fonction : **en rouge le code à écrire.**

```
CREATE or REPLACE FUNCTION
GET_AMOUNT_LAST_ORDER_EX (p_numfou IN commande.numfou%TYPE,
                           p_montant OUT NUMBER)
AS
CODE
BEGIN
CODE
RETURN NULL;
EXCEPTION
CODE
END;
```

 Indices : Utilisation du bloc EXCEPTION. Utilisation de l'instruction RAISE pour lever une exception.

 Cf. document *CDI JEE - Langage PL-SQL.pdf*

 Exécutez la fonction pour vérifier qu'elle fonctionne conformément à votre attente.

Bloc anonyme pour tester la fonction : **en rouge le code à écrire.**

```
DECLARE
v_montant NUMBER;
v_numfou NUMBER;
v_return VARCHAR2(20);
BEGIN
-- Fournisseur existe
v_numfou := 2;
-- Appel de la fonction
CODE
-- Affichage du résultat
CODE
-- Fournisseur n'existe pas
v_numfou := 212;
-- Appel de la fonction
CODE
-- Affichage du résultat
CODE
END;
```

IV LES DECLENCHEURS

Ci-dessous, un rappel des principales actions que vous pouvez effectuer sur les triggers.

Création du trigger *nom_trigger*

```
CREATE [OR REPLACE] TRIGGER [schéma.]nom_trigger
{BEFORE | AFTER | INSTEAD OF | FOR }
{DELETE | INSERT | UPDATE [OF colonne [,colonne]...]}
[OR {DELETE | INSERT | UPDATE [OF colonne [,colonne]...]}...
ON [schéma.]table
[[REFERENCING {OLD [AS] ancien NEW [AS] nouveau }
| NEW [AS] nouveau OLD [AS] ancien }]]
[FOR EACH ROW]
[FOLLOWS nom_autre_trigger]
[COMPOUND TRIGGER]
[ENABLE|DISABLE]
[WHEN (condition )]
BEGIN
Bloc PL/SQL
END;
```

Suppression du trigger *nom_trigger*

```
DROP TRIGGER nom_trigger;
```

Désactivation du trigger *nom_trigger*

```
ALTER TRIGGER nom_trigger DISABLE;
```

Activation du trigger *nom_trigger*

```
ALTER TRIGGER nom_trigger ENABLE;
```

IV.1 CREATION D'UN DECLENCHEUR AFTER INSERT, UPDATE ET DELETE

Créez un trigger nommé **TR_MAJSTOCK** qui remet à jour le stock de l'article (champs stkphy de la table PRODUIT) après chaque suppression, insertion ou modification d'une ligne de commande (table LIGCOM).

Squelette du trigger : **en rouge le code à écrire.**


```
CREATE OR REPLACE TRIGGER TR_MAJSTOCK
-- Le bloc est exécuté après la mise à jour des données dans la table LIGCOM.

-- Trigger ligne : le corps du trigger sera exécuté pour chaque ligne touchée par
-- l'instruction qui a déclenché le trigger.
BEGIN
-- SI insertion d'une nouvelle ligne de commande
-- Mettre à jour le stock
-- FINSI insertion d'une nouvelle ligne de commande


-- SI modification d'une ligne de commande
-- Mettre à jour le stock
-- FINSI modification d'une ligne de commande

-- SI suppression d'une ligne de commande
-- Mettre à jour le stock
-- FINSI suppression d'une ligne de commande

END;
```

 Indices : Utilisation des booléens INSERTING, UPDATING et DELETING pour connaître le type de la commande SQL exécutée (insert, update ou delete). Utilisation de :new et :old. Utilisation de FOR EACH ROW pour déclarer un trigger ligne.

 Cf. document *CDI JEE - Langage PL-SQL.pdf*

 Exécutez des ordres de modification, d'insertion et de suppression sur la table LIGCOM pour vérifier que le trigger fonctionne conformément à votre attente.

Test du trigger

```
-- Sur une modification d'une ligne de commande
UPDATE LIGCOM SET QTECDE=? WHERE NUMCOM=? AND CODART=?;

-- Sur une insertion d'une ligne de commande
INSERT INTO LIGCOM (NUMCOM, CODART, NUMLIG, QTECDE, PRIUNI, QTELIV,
DERLIV)
VALUES (?, ?, ?, ?, ?, ?, ?);

-- Sur une suppression d'une ligne de commande
DELETE FROM LIGCOM WHERE NUMCOM=? AND NUMLIG = ?;
```

IV.2 MODIFICATION D'UN DECLENCHEUR AFTER UPDATE

Modifiez le trigger précédent pour qu'il détecte si les stocks sont insuffisants. Si le stock est insuffisant, le trigger doit générer une exception. Pour ce faire, utilisez l'instruction RAISE_APPLICATION_ERROR décrite dans votre support de cours.

Squelette du trigger : **en rouge le code à écrire.**

```
CREATE OR REPLACE TRIGGER TR_MAJSTOCK
-- Le bloc est exécuté après la mise à jour des données dans la table.

-- Trigger ligne : le corps du trigger est exécuté pour chaque ligne touchée par
-- l'instruction qui a déclenché le trigger

DECLARE
    CODE
BEGIN


-- SI insertion ou modification d'un enregistrement dans LIGCOM ALORS
-- Mémoriser dans une variable le stock physique actuel du produit
-- FINSI

-- SI insertion d'une ligne de commande
-- SI stock physique du produit >= quantité commandée
-- Mise à jour du stock
-- SINON stock physique du produit < quantité commandée
-- Lever une exception
-- FINSI
-- FINSI insertion d'une ligne de commande

-- SI Modification d'une ligne de commande
-- SI stock physique du produit >= quantité commandée
-- Mise à jour du stock
-- SINON stock physique du produit < quantité commandée
-- Lever une exception
-- FINSI
-- FINSI modification d'une ligne de commande

-- SI suppression d'une ligne de commande
-- Modification du stock physique du produit
-- FINSI suppression d'une ligne de commande

EXCEPTION
-- Traitement de l'exception stock insuffisant
-- Traitement des autres exceptions
END;
```

 Exécutez des ordres de modification, d'insertion et de suppression sur la table LIGCOM pour vérifier que le trigger fonctionne conformément à votre attente.

Test du trigger

```
-- Sur une modification d'une ligne de commande
```

```
UPDATE LIGCOM SET QTECDE =? WHERE NUMCOM=? AND CODART=?;
```

```
-- Sur une insertion d'une ligne de commande
```

```
INSERT INTO LIGCOM (NUMCOM, CODART, NUMLIG, QTECDE, PRIUNI, QTELIV,  
DERLIV)
```

```
VALUES (?, ?, ?, ?, ?, ?, ?);
```

```
-- Sur une suppression d'une ligne de commande
```

```
DELETE FROM LIGCOM WHERE NUMCOM=? AND NUMLIG = ?;
```

IV.3 CREATION D'UN DECLENCHEUR INSERTION SUR LA TABLE PRODUIT


Créez un trigger nommé **TR_VERIF_ARTICLE** qui empêche l'insertion d'une désignation d'article (LIBART de la table PRODUIT) déjà présente dans la table en générant une exception.


Squelette du trigger : **en rouge le code à écrire.**

```
CREATE OR REPLACE TRIGGER TR_VERIF_ARTICLE
-- Trigger se déclenche avant l'insertion de l'enregistrement dans la table PRODUIT

-- Trigger de ligne : le corps du trigger est exécuté pour chaque ligne touchée par
-- l'instruction qui a déclenché le trigger

DECLARE
    CODE
BEGIN
    -- Rechercher dans la table PRODUIT si un article existe déjà avec la
    -- désignation du nouvel article à insérer
    -- SI un article existe
    -- Lever une exception
    -- FINSI
END;
```

 Indice : Utilisation de l'instruction RAISE_APPLICATION_ERROR() pour lever une exception.

 Exécutez des commandes d'insertion sur la table PRODUIT pour vérifier que le trigger fonctionne conformément à votre attente.

Test du trigger

```
-- Insertion avec un libelle qui n'existe pas
INSERT INTO PRODUIT (CODART, LIBART, STKALE, STKPHY, QTEANN, UNIMES)
VALUES (?,?,?, ?, ?, ?);
--> insertion ok

-- Insertion avec un libelle qui existe déjà
INSERT INTO PRODUIT (CODART, LIBART, STKALE, STKPHY, QTEANN, UNIMES)
VALUES (?,?,?, ?, ?, ?);
--> insertion Nok
```

IV.4 CREATION D'UN DECLENCHEUR LDD (DDL)

RAPPEL : Le Langage de Définition des Données – LDD (en anglais DDL - Data Definition Language) regroupe l'ensemble des possibilités permettant de modifier la structure de la base de données.

Les principales instructions du LDD sont : CREATE, ALTER, DROP, RENAME, TRUNCATE TABLE, PURGE, COMMENT, GRANT (accorder des droits), DENY, REVOKE (retirer des droits) et AUDIT (activer un audit sur les objets Oracle).

Les déclencheurs LDD s'exécutent donc en réponse aux instructions LDD. Il est possible d'utiliser ces déclencheurs pour suivre les changements de la structure de la base de données, tels que :

- Création, modification ou suppression d'un objet au sein de la base de données ou d'un schéma.
- Mise en place ou la suppression d'un audit sur un objet de la base de données ou d'un schéma.
- Modification des privilèges et des rôles.


Créez un déclencheur qui interdit toutes opérations de création, modification, suppression sur les triggers sur le schéma PAPYRUS.
Affichez un message qui indique le type d'événement (CREATE, DROP,...).

Squelette du trigger : **en rouge le code à écrire.**


```
CREATE OR REPLACE TRIGGER TR_SECURITE_TRIGGER
-- Trigger qui se déclenche avant la création, la modification ou la suppression d'un objet
-- du schéma
DECLARE
CODE
BEGIN
-- Récupérer le type d'événement (CREATE, DROP,...)

-- Récupérer le type d'objet (TRIGGER, ....)


-- SI l'objet est un trigger
-- SI création
-- Afficher message création impossible
-- FINSI
-- SI modification
-- Afficher message modification impossible
-- FINSI
-- SI suppression
-- Afficher message suppression impossible
-- FINSI
Lever une exception
-- FINSI
END;
```

 Indices : Utilisation de la fonction ora_sysevent pour connaître le type d'événement (DROP, CREATE, ...) et de la fonction ora_dict_obj_type pour connaître le type de l'objet (TRIGGER, SEQUENCE, ...). Utilisation de l'instruction RAISE_APPLICATION_ERROR() pour lever une exception.

 Cf. document *CDI JEE - Langage PL-SQL.pdf*

 Exécutez des ordres de suppression, de modification et de création d'objets du schéma pour vérifier que le trigger fonctionne conformément à votre attente.

```
DROP TABLE nom_table;  
CREATE SEQUENCE nom_sequence;  
DROP TRIGGER nom_trigger;  
COMMENT ON TABLE nom_table IS 'Commentaire sur la table';
```

 Note : pour supprimer ce trigger, vous devez vous connecter avec le compte sys et exécuter une commande DROP sur le trigger.

Suppression du trigger (en tant que sys)

```
DROP TRIGGER papyrus.TR_SECURITE_TRIGGER;
```


IV.5 CREATION D'UN DECLENCHEUR POUR GERER LES CLES PRIMAIRES

Créez un déclencheur qui permet d'insérer automatiquement sur une requête insert la clé primaire d'une table.

La table ci-dessous est utilisée pour tester le déclencheur.

```
CREATE TABLE "VILLE"  
(  
    ID_VILLE NUMBER,  
    NOM VARCHAR2(25 BYTE) NOT NULL ENABLE  
);  
  
ALTER TABLE "VILLE"  
    ADD CONSTRAINT pk_ville PRIMARY KEY (ID_VILLE);
```

Avant de créer le déclencheur, vous devez créer une séquence nommée **SQ_VILLE** qui sera utilisée par le déclencheur. Cette séquence commence à 1 et s'incrémente de 1 en 1.


Squelette de la séquence : **en rouge le code à écrire.**

```
CREATE SEQUENCE Code
```

Squelette du trigger : **en rouge le code à écrire.**

```
CREATE OR REPLACE trigger TR_SQ_VILLE  
    -- Déclencheur de type BEFORE INSERT sur la table VILLE  
    -- Déclencheur ligne  
BEGIN  
    -- Affecter à ID_VILLE (clé primaire) la prochaine valeur de la séquence SQ_VILLE  
END;
```

 Indice : Utilisation de la fonction NEXTVAL pour obtenir la prochaine valeur de la séquence.

 Exécutez des commandes d'insertion sur la table VILLE pour vérifier que le trigger fonctionne conformément à votre attente.


1^{ère} syntaxe

```
INSERT INTO VILLE (NOM) VALUES ('ROME');
```

2^{ème} syntaxe

```
INSERT INTO VILLE (ID_VILLE, NOM) VALUES ('', 'LYON');
```

Vérifiez le contenu de la table VILLE.

 Note : pour insérer automatiquement une clé primaire AVEC une séquence mais **SANS** trigger, vous devez utiliser la syntaxe ci-dessous.

```
INSERT INTO VILLE (ID_VILLE, NOM) VALUES (SQ_VILLE.nextval, 'LYON');
```

Etablissement référent
Direction de l'ingénierie Neuilly

Equipe de conception
Pascal DANGU

Remerciements :
A Bob, Keith, Miles,

Reproduction interdite

Article L 122-4 du code de la propriété intellectuelle.
« Toute représentation ou reproduction intégrale ou partielle faite sans le consentement de l'auteur ou de ses ayants droits ou ayants cause est illicite. Il en est de même pour la traduction, l'adaptation ou la reproduction par un art ou un procédé quelconque. »

Date de mise à jour 10/12/2014
afpa © Date de dépôt légal décembre 14

