ENICARTHAGE
المدرسة الوطنية للمهندسين بقرطاج
Ecole Nationale d'Ingénieurs de Carthage

PL-SQL
Procedural Language/SQL

Rym MGHIRBI
Ecole Nationale d'Ingénieurs de CARthage

Plan

- Introduction à PL/SQL
- Structure d'un programme PL/SQL
- Les déclarations
- Les instructions PL/SQL
  - Affectation, IF, WHILE, LOOP
- La gestion des curseurs
- Les Modules stockés
- Le traitement des erreurs
- Les déclencheurs

IX-Les déclencheurs Triggers

1. Définitions

- Un déclencheur (trigger) est une procédure stockée qui se déclenche automatiquement (Traitements implicites)sur certains événements.
- Un déclencheur est aussi une règle ECA

```
Si événement
[Condition]
Alors action
Sinon rien
Finsi
```

  - **E**vénement = LMD|LDD|INSTANCE
  - **C**ondition = optionnelle, équivaut à une clause
  - **A**ction = exécution de code spécifique (requête SQL de mise à jour, exécution d'une procédure stockée, abandon d'une transaction, ...).
  - **L'action** est déclenchée à la suite de **l'événement**, si la **condition** est vérifiée.

1. Définitions

- Un déclencheur peut être exécuté:
  1. A la création, suppression ou modification d'un objet (CREATE, DROP, ALTER), ou suite à l'attribution ou la révocation des droits (GRANT|REVOKE) sur un objet (table, index, séquence, etc.)

     → On parle de **déclencheurs LDD;**
  2. **A des instructions INSERT, DELETE, UPDATE sur une table (ou vue)**

     → **On parle de déclencheurs LMD;**
  3. Au démarrage ou à l'arrêt de la base (startup ou shutdown),
     - ou Suite à une erreur spécifique (NO_DATA_ FOUND, DUP_VAL_ON_INDEX, etc.),
     - Suite à des événements utilisateurs (Une connexion ou une déconnexion d'un utilisateur.)

     → On parle de **déclencheurs d'instances**.

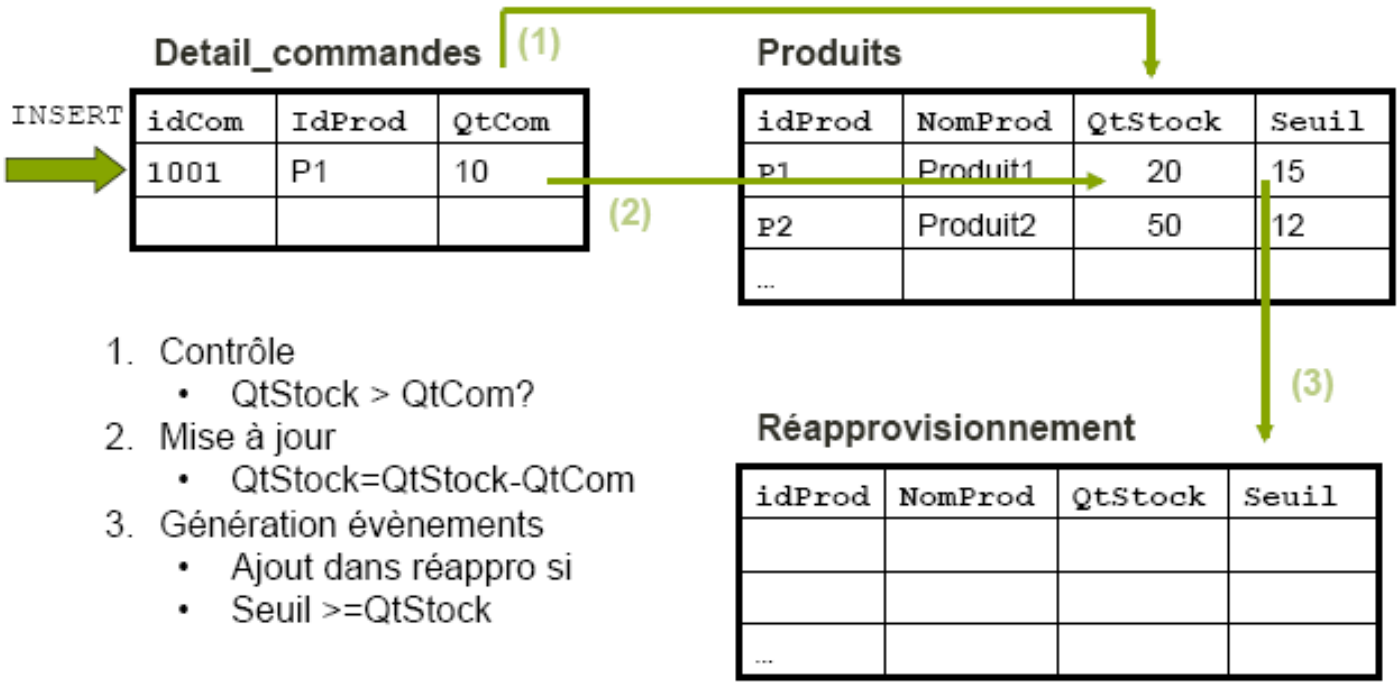
1. Définitions

- Les triggers peuvent être **applicatifs** (créés et manipulés au niveau de l'application→ invisible en dehors de l'application) ou **de base de données.**
- Nous ne nous intéresserons qu'aux déclencheurs de BD.
- Les déclencheurs de BD sont indépendants de l'environnement à partir duquel l'événement est lancé.
  - Les triggers font partie du schéma de la base.
  - Leur code compilé est conservé (comme pour les programmes stockés)

2. Utilité

- Gestion des redondances (insérer une colonne calculée au sein d'une table et contrôle de la cohérence)
- Enregistrement automatique de certains événements (auditing)
- Gestion de contraintes complexes (exemple : le salaire d'un employé ne peut qu'augmenter)
- Gestion de contraintes liées à l'environnement d'exécution (restrictions sur les horaires, les utilisateurs, etc.)
- Gestion de certains aspects transactionnels

2. Utilité- Situation



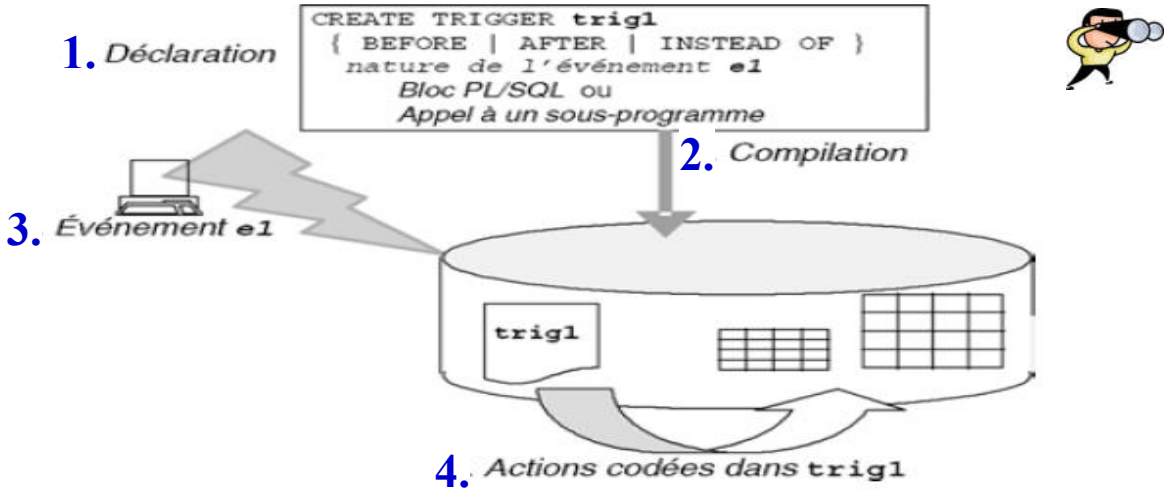
1. Contrôle
   - QtStock > QtCom?
2. Mise à jour
   - QtStock=QtStock-QtCom
3. Génération évènements
   - Ajout dans réappro si
   - Seuil >=QtStock

3. Privilèges et conseils

- Privilèges requis:
  - Pour pouvoir créer un déclencheur dans votre schéma, vous devez disposer du privilège CREATE TRIGGER (qui est inclus dans le rôle RESOURCE mais pas dans CONNECT).
  - Pour créer un déclencheur dans un autre schéma, le privilège CREATE ANY TRIGGER est requis.
  - En plus de ces conditions, pour créer un déclencheur d'instances, il faut détenir le privilège ADMINISTER DATABASE TRIGGER.

4. Mécanisme général

- Mise en œuvre d'un déclencheur :
  - Il faut d'abord le coder (comme un sous-programme) **(1)**, puis le compiler (il sera stocké ainsi en base) **(2)**.
  - Par la suite, au cours du temps, et si le déclencheur est actif (il est possible de désactiver un déclencheur même s'il est compilé), chaque événement **(3)** (qui caractérise le déclencheur) aura pour conséquence son exécution **(4)**.



ALTER TRIGGER nom_trigger ENABLE | DISABLE

5. Conseils sur l'implémentation des Triggers

- Les triggers sont des mécanismes puissants
  - Pour contrôler ce qui se passe dans une base de données
  - Pour mémoriser les actions sensibles (destruction)
  - Mais dangereux, à utiliser avec précaution
  - L'exécution d'un trigger est invisible : l'équivalent d'un effet de bord en programmation:
    - Ne pas créer de déclencheurs récursifs (exemple d'un déclencheur qui exécute une instruction lançant elle-même un déclencheur ou deux déclencheurs s'appelant en cascade jusqu'à l'occupation de toute la mémoire réservée).
    - Un trigger peut être source (invisible) de lenteur
  - La composition de plusieurs triggers peut être est incontrôlable
- Autres:
  - Il est conseillé de limiter la taille (partie instructions) d'un déclencheur à soixante lignes de code PL/SQL (la taille d'un déclencheur ne peut excéder 32 ko). ➔ Pour contourner cette limitation, appeler des sous-programmes dans le code du déclencheur.
  - Un déclencheur ne peut valider aucune transaction, ainsi les instructions suivantes sont interdites : COMMIT, ROLLBACK, SAVEPOINT, et SET CONSTRAINT.

6. Déclencheurs LMD

6.1 Introduction

- **Un déclencheur LMD est associé à une et une seule table;**
- **il est opérationnel jusqu'à la suppression de la table à laquelle il est lié.**
- **Il peut être désactivé (**ALTER TRIGGER nom_trigger ENABLE | DISABLE)

- Sous Oracle, aussi triggers INSTEAD OF sur les vues

6.2 Syntaxe de création

```
CREATE [OR REPLACE] TRIGGER <nom_trigger>
{BEFORE | AFTER| INSTEAD OF}          --moment de déclenchement ou séquencement
                                       -- instead of : spécifique aux vues
INSERT |DELETE | UPDATE [ OF colonnes]      -- type d'évenement LMD
ON <nom_table>                              -- un déclencheur par table
[FOR EACH ROW]– Niveau de déclenchement-- le déclencheur est de niveau ligne : row trigger
    /* si For each row n'est pas indiqué: c'est un déclencheur niveau table : statement trigger */
[WHEN condition]
            /*specifique au row trigger  pour chaque ligne, trigger déclenché si vrai.*/

[DECLARE]
-- déclaration de variables, exceptions,
-- curseurs
BEGIN
-- corps du trigger
        <Action(s)>
END;
/
```

6.3 Corps du Trigger

- Le bloc d'instructions PL/SQL peut contenir:
  - des blocs spécifiant des actions différentes en fonction de l'événement déclencheur
  - des Instructions SQL
    - SELECT, INSERT, UPDATE, DELETE, ... Mais pas de COMMIT et ROLLBACK
    - En particulier la commande SELECT ... INTO ou définir un curseur.
  - Instructions de contrôle de flux (IF, LOOP, WHILE, FOR)
  - Générer des exceptions
    - Raise_application_error(code_erreur,message)
    - code_erreur compris entre -20000 et -20999 (sinon code d'erreur oracle)
    - Si on fait une section Exception: la procédure RAISE_APPLICATION_ERROR passe par la section EXCEPTION (s'il en existe une) avant de terminer le déclencheur. En conséquence, si vous utilisez aussi une section exception dans le même bloc, il faut forcer la sortie du déclencheur par la directive RAISE pour ne pas perdre le message d'erreur et surtout ne pas réaliser la mise à jour de la base.
  - Faire appel à des procédures et fonctions PL/SQL

6.4 Row Level trigger

- Un déclencheur de lignes est déclaré avec la directive FOR EACH ROW.
  - Ce n'est que dans ce type de déclencheur qu'on a accès aux **anciennes valeurs (OLD)** et **aux nouvelles valeurs (NEW)** des colonnes de la ligne affectée par la mise à jour prévue par l'événement.
  - La valeur prise en compte dépend de l'ordre SQL :

	:old	:new
INSERT	null	valeur insérée
DELETE	valeur supprimée	null
UPDATE	valeur avant modif	valeur après modif

La clause WHEN

- On peut définir une condition pour un trigger de niveau ligne : le trigger se déclenchera pour chaque ligne vérifiant la condition.
- La condition contenue dans la clause WHEN doit être une expression SQL, et ne peut inclure de requêtes ni de fonctions PL/SQL.
- Dans la clause **WHEN** ou dans le corps, on peut se référer à la valeur d'un attribut avant ou après que soit effectuée la modification déclenchant le trigger :
  - **OLD**.*nomAttribut* : la valeur avant la transaction UPDATE ou DELETE
  - **NEW**.*nomAttribut* : la valeur après la transaction UPDATE ou INSERT.

Exemple 1

Code PL/SQL	Commentaires

`CREATE OR REPLACE TRIGGER TrigInsQualif`
`BEFORE INSERT ON Qualifications`
`FOR EACH ROW`
	Déclaration de l'événement déclencheur.

`WHEN (NEW.typa = 'A320' OR NEW.typa = 'A340'` `OR NEW.typa = 'A330')`	Condition de déclenchement.

`DECLARE`
`…`
`BEGIN`
`…`
`END;`
`/`
	Corps du déclencheur.

Événement déclencheur	Événement non déclencheur
`INSERT INTO Qualifications` `VALUES ('PL-2','A340','20-06-2006');`	`INSERT INTO Qualifications` `VALUES ('PL-2','A380','20-06-2006');`

Exemple 2

- Déterminer que fait ce trigger:

```
CREATE OR REPLACE TRIGGER journal_emp
AFTER UPDATE OF salary ON EMP
FOR EACH ROW
WHEN (new.salary < old.salary)
BEGIN
INSERT INTO EMP_LOG(emp_id, date_evt, msg)
VALUES (:new.empno, sysdate,  'salaire diminué' );
end ;
```


Exemple 2

EMP

Empno	…	Salary
1		10000
2		4000
3		6000
4		8000

update Emp

set salary = 5000

where salary < 10000

;

EMP

Empno	…	Salary
1		10000
2		5000
3		5000
4		5000

3 lignes modifiées

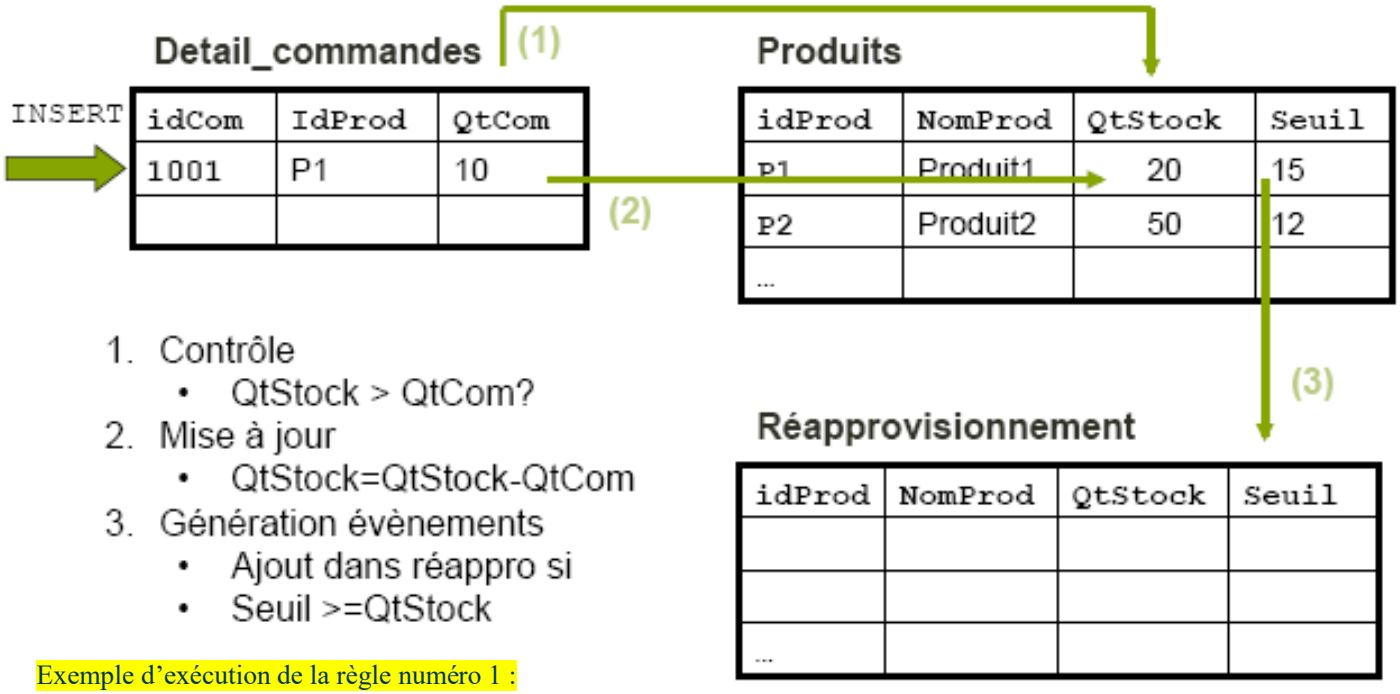
EMP_LOG

emp_id	date_evt	msg
3	5/03/16	salaire diminué
4	5/03/16	salaire diminué

Le trigger se déclenche 2 fois

Exercice 1

- Créer les programmes permettant de résoudre les règles de gestion ci-contre:



1. Contrôle
   - QtStock > QtCom?
2. Mise à jour
   - QtStock=QtStock-QtCom
3. Génération évènements
   - Ajout dans réappro si
   - Seuil >=QtStock

Exemple d'exécution de la règle numéro 1 :

SQL> INSERT INTO Detail_commandes values (1002,'P1', 30)

ORA_20001 : Stock insuffisant!!

Solution (1/3)

```
CREATE OR REPLACE TRIGGER t_b_i_detail_commandes
BEFORE INSERT ON detail_commandes
FOR EACH ROW
DECLARE
vqtstock NUMBER;
BEGIN
SELECT qtstock INTO vqtstock FROM produits
WHERE idprod = :NEW.idprod;
IF vqtstock < :NEW.qtcom THEN
RAISE_APPLICATION_ERROR(-20001, 'stock insuffisant');
END IF;
END;
/
```

Solution (2/3)

```
CREATE OR REPLACE TRIGGER t_a_i_detail_commandes
AFTER INSERT ON detail_commandes

FOR EACH ROW
BEGIN
UPDATE Produits p
SET p.qtstock = p.qtstock - :NEW.qtcom
WHERE idprod= :NEW.idprod;
END;
/
```

--qd lancer le déclencheur: le séquecement,
-- quel évènement
--Niveau de déclenchement : Trigger niveau ligne

Actions

Solution (3/3)

```
CREATE TRIGGER t_a_u_produits

    AFTER UPDATE OF qtstock ON produits
    FOR EACH ROW

BEGIN

    IF :NEW.qtstock <= : NEW.seuil THEN
        INSERT INTO reapprovisionnement VALUES
        (:NEW.idprod, :NEW.nomprod, :NEW.qtstock, :NEW.seuil);
    END IF;

END;
/
```

Solution (3/3) ou

```
CREATE TRIGGER t_a_u_produits

    AFTER UPDATE OF qtstock ON produits
    FOR EACH ROW
     WHEN NEW.qtstock <= NEW.seuil
BEGIN

    IF :NEW.qtstock <= : NEW.seuil THEN
          INSERT INTO reapprovisionnement VALUES
          (:NEW.idprod, :NEW.nomprod, :NEW.qtstock, :NEW.seuil);
    END IF;

END;
/
```


Exercice 2

- Schéma relationnel suivant:
  - Client(numClient, nom, prénom, ca)
  - Article(numArticle, description, prixUnitaire, quantitéEnStock)
  - Commande(numCommande, #numClient, dateCommande, montant)
  - LigneCommande(#numCommande, #numArticle, quantité)

1. Créer un déclencheur « Pas_baisse_prix » qui empêche qu'un prix unitaire d'un article de baisser.

Exemple d'exécution:

```
SQL> UPDATE Article SET prixUnitaire=100 WHERE numArticle=1;
ORA_20100 : le prix d'un article ne peut pas diminuer!!!
```

```
CREATE OR REPLACE TRIGGER pas_baisse_prix
        BEFORE UPDATE OF prixUnitaire ON Article
        FOR EACH ROW
        WHEN (OLD.prixUnitaire > NEW.prixUnitaire)
BEGIN
        RAISE_APPLICATION_ERROR(-20100, 'le prix d''un
        article ne peut pas diminuer!!!') ;
END;
/
```

Exercice 2
(Before ou After)

- On termine avec l'extrait suivant du même schéma relationnel
  - Client(numClient, nom, prénom, CA),
  - Commande(numCommande, #numClient, dateCommande, montant)

2. Ecrire un programme qui, au passage d'une commande client, augmente le chiffre d'affaire de ce dernier par le montant de sa commande. Si le client est nouveau, il doit l'ajouter à la base . Son chiffre d'affaire sera initialisé au montant de la commande.

```
CREATE OR REPLACE TRIGGER maj_client
    BEFORE INSERT ON Commande
    FOR EACH ROW
DECLARE
        Vclient Client.numClient%TYPE;
BEGIN
   SELECT numClient INTO Vclient FROM Client
   WHERE numClient = :new.numClient ;
   UPDATE Client SET ca = ca + :new.montant
   WHERE numClient = :new.numClient ;
EXCEPTION
   WHEN NO_DATA_FOUND THEN
   INSERT INTO Client(numClient, CA) VALUES ( :new.numClient, :new.montant) ;
END;
```


6.5 Regroupement d'événements

Des événements (INSERT, UPDATE ou DELETE) peuvent être **regroupés** au sein d'un même déclencheur s'ils sont de même type (BEFORE ou AFTER).

```
CREATE TRIGGER <nom_trigger>
{BEFORE | AFTER}
INSERT [OR] DELETE [OR] UPDATE [ OF colonnes]
ON <nom_table>
[FOR EACH ROW]
[DECLARE]
-- déclaration de variables, exceptions,
-- curseurs
BEGIN
    IF UPDATING('colonne') THEN (actions en cas de mise à jour) END IF;
    IF DELETING THEN (actions en cas de suppression) END IF;
    IF INSERTING THEN (actions en cas d'insertion) END IF;
END;
/
```

Exemple 3

Code PL/SQL	Commentaires

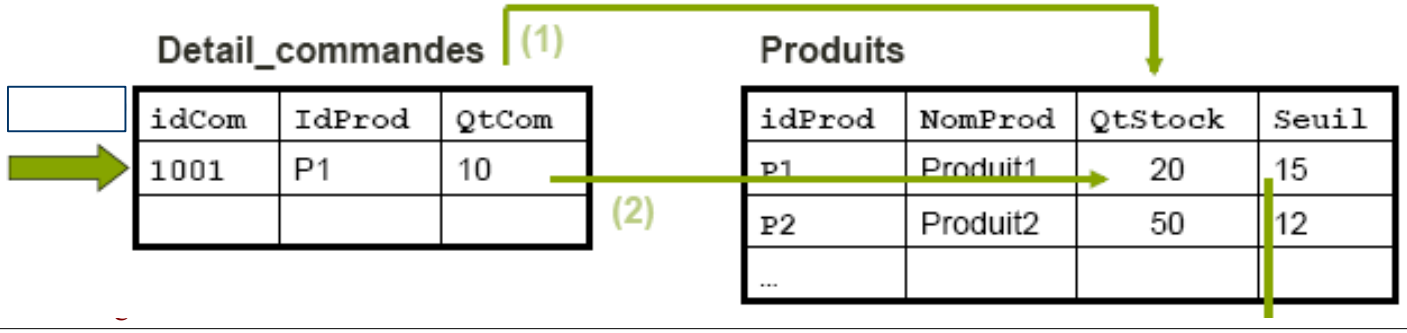
`CREATE OR REPLACE TRIGGER TrigDelUpdQualif`
`AFTER DELETE OR UPDATE OF brevet ON Qualifications`
`FOR EACH ROW`
	Regroupement de deux événements déclencheurs.

`DECLARE`
`...`
`BEGIN`
`IF (DELETING) THEN`
`...`
	Bloc exécuté en cas de DELETE.

`ELSIF (UPDATING('brevet')) THEN`
`...`
`END IF;`
`END;`
	Bloc exécuté en cas de UPDATE de la colonne brevet.

Exercice 4

- Proposer un déclencheur qui permet de mettre à jour la quantité en stock d'un produit dans le cadre des différentes manipulations des commandes [détail_commandes]
  - Passage d'une nouvelle commande de produit,
  - Modification de la quantité commandée d'un produit ou
  - Annulation d'une commande de produit



Exercice 4

```
CREATE OR REPLACE TRIGGER tr_MAJStockClt
AFTER INSERT OR UPDATE OF qtcom OR DELETE
ON detail_commandes
FOR EACH ROW
DECLARE
vdiffqte detail_commandes.qtcom%TYPE;
BEGIN
IF INSERTING THEN
UPDATE Produits SET qtstock= qtstock - :NEW.qtcom
WHERE idProd=:NEW. idProd;
END IF;
IF DELETING THEN
UPDATE Produits SET qtstock= qtstock + :OLD.qtcom
WHERE idProd =:OLD. idProd;
END IF;
IF UPDATING THEN
/* le client augmente la quantité commandé du produit */
IF :OLD.qtcom <:NEW.qtcom THEN
vdiffqte :=:NEW.qtcom - :OLD.qtcom;
UPDATE Produits SET qtstock= qtstock -vdiffqte
WHERE idProd =:OLD. idProd;
END IF;

/* le client retourne une partie du produit ou la totalité*/
IF :OLD.qtcom >:NEW.qtcom THEN
vdiffqte := :OLD.qtcom- :NEW.qtcom;
UPDATE Produits SET qtstock= qtstock +vdiffqte
WHERE idProd =:OLD. idProd;
END IF;
END IF;
END;
```


6.6 Déclencheur d'état *(statement trigger)*

- Un déclencheur d'état est déclaré **sans** la directive FOR EACH ROW.
  - Il n'est pas possible d'avoir accès aux valeurs des lignes mises à jour par l'événement.
  - Le raisonnement de tels déclencheurs porte donc sur la globalité de la table et non sur chaque enregistrement particulier.

Exemple 4

Row-level Trigger

```
CREATE OR REPLACE TRIGGER journal_emp
AFTER UPDATE OF salary ON EMP
FOR EACH ROW
WHEN (new.salary < old.salary)
BEGIN
INSERT INTO EMP_LOG(emp_id, date_evt, msg)
VALUES (:new.empno, sysdate, 'salaire diminué' );
end ;
```

Exemple d'exécution

```
UPDATE Emp SET salary = 5000 WHERE SLARY < 10000
```

EMP

Empno	…	Salary
1		10000
2		4000
3		6000
4		8000

⇨

EMP

Empno	…	Salary
1		10000
2		5000
3		5000
4		5000

3 lignes modifiées

Statement Level Trigger

```
CREATE OR REPLACE TRIGGER historique_evt
AFTER UPDATE OF salary ON EMP
BEGIN
INSERT INTO edit_history (history_id, username, modification, edit_date) VALUES (history_id_sequence.nextVal, USER,'Update', SYSDATE)end ;
```

EMP_LOG

Emp_id	Date_evt	msg
3	07/04/2020	salaire diminué
4	07/04/2020	salaire diminué

Edit_history

history_id	Username	Modification	Edit_date
1	toto	Update	07/04/2020

Exercice 5

- Créer un déclencheur 'Alerte_MAJ_CMD' qui empêche tout passage, annulation ou modification d'une commande à partir de 20h.00

Exemple d'exécution:

- SQL> DELETE FROM Commande WHERE montant>500 ;
- **ORA-20102: Désolé pas de manipulation des commandes à partir de 20h.00**

Exercice 5

- Créer un déclencheur 'Alerte_MAJ_CMD' qui empêche tout **passage, annulation** ou **modification** d'une **commande** à partir de **20h.00**

Exemple d'exécution:

- SQL> DELETE FROM Commande WHERE montant>500 ;
- **ORA-20102: Désolé pas de manipulation des commandes à partir de 20h.00**

```
CREATE OR REPLACE TRIGGER Alerte_maj_client
    BEFORE INSERT OR UPDATE OR DELETE ON Commande
BEGIN
   IF TO_CHAR(SYSDATE,'HH24:MI') >='20:00'then
Raise_application-error(-20102,'Désolé pas de manipulation des commandes à partir de 20h.00');
END if;
End;
/
```


6.7 Trigger INSTEAD OF

- Trigger faisant le travail ' à la place de '…
- Posé sur une vue multi-tables pour autoriser les modifications sur des objets virtuels (car mise à jour impossible sur des vues multi-tables)
- Quand utiliser les triggers INSTEAD OF
  - Quand la vue ne peut pas être modifiée directement.
  - C'est-à-dire quand elle contient :
    - Un opérateur ensembliste
    - Un opérateur DISTINCT
    - Une fonction d'agrégat
    - Un GROUP BY, un ORDER BY
    - Une sous-requête dans la liste du SELECT
    - Une jointure (mais ça dépend des cas)

Exemple 5

- Soit la vue étudiant résultant de 4 tables :
  - Etudiant_licence, Etudiant_Master, Etudiant_doctorat, Stage

```
Create View Etudiant (num, Nom, Adresse, cycle, nomstage, adstage)
AS
SELECT el.num, el.Nom, el.adr, 'L', s.noms, s.ads FROM etudiant_licence el, stage s
Where el.num=s.num
UNION
SELECT em.num, em.Nom, em.adr, 'M', s.noms, s.ads FROM etudiant_Master em, stage s
Where em.num=s.num
UNION
SELECT ed.num, ed.Nom, ed.adr, 'D', s.noms, s.ads FROM etudiant_doctorat ed, stage s
Where ed.num=s.num
```

- Si on fait un:
  - INSERT INTO ETUDIANT (100, 'Claude', 'Toulouse', 'M', 'Oracle', 'CICT')
  - **Pas possible le SGBD ne sait pas dans quelle table** insérer les données (etudiant_licence?, etudiant_master ? Stage ? , etudiant_doctorat ?

Exemple 5 (suite)

- Solution créer un trigger 'INSTEAD of'

```
CREATE Trigger insert_etudiant
INSTEAD OF INSERT ON Etudiant
FOR EACH ROW
BEGIN
IF :NEW.cycle='L' THEN INSERT INTO etudiant_licence VALUES (:new.num,
:new.nom, :new.adresse)
INSERT INTO Stage VALUES (:new.num, :new.nomstage, :new.adstage)
ELSIF : NEW.cycle='M' THEN
--……..Idem pour M et D
ELSE RAISE_APPLICATION_ERROR(-20455, 'Enter M, L ou D'):
END IF;
END;
```

Exemple 6

Movie (title,year,length,studioname,producer); Soit la vue ParamountMovie définie comme suit.

```
Create view ParamountMovie As
select title, year from movie
where studioName = 'Paramount'
WITH CHECK OPTIONS;
```

On veut insérer le tuple 'Star Treck', 1979, à travers la vue ParamountMovie.

➔ Insert into **ParamountMovie values** ('Star Treck', 1979)

Quel est le pbm qui va suivre cette action si on fait **select * from ParamountMovie?**

➔ Le tuple inséré n'est pas retourné par la vue

Y- Donner une solution?

Exemple 6

Movie (title,year,length,studioname,producer); Soit la vue ParamountMovie définie comme suit.

```
Create view ParamountMovie As
select title, year from movie
where studioName = 'Paramount'
WITH CHECK OPTIONS;
```

On veut insérer le tuple 'Star Treck', 1979, à travers la vue ParamountMovie.

➔ Insert into **ParamountMovie values** ('Star Treck', 1979)

Quel est le pbm qui va suivre cette action si on fait **select * from ParamountMovie?**

➔ Le tuple inséré n'est pas retourné par la vue

Y- Donner une solution?

```
create trigger ParamountInsert
instead of insert on ParamountMovie
for each row
begin
insert into Movie(title,year,studioName) values (:new.title, :new.year,
'Paramount')
end ;
```

6.8 Tables mutantes

- Une table mutante est une table qui est en cours de modification par une opération déclenchante (UPDATE, DELETE, ou INSERT) ou par l'effet de DELETE CASCADE provenant de cette opération.
  - Il est en principe, interdit de manipuler une table mutante dans le corps du déclencheur lui-même.
    - Lecture possible de la table dans le cas d'un déclenchement par un insert de type **BEFORE**.
  - Les vues modifiables par des déclencheurs INSTEAD OF ne sont pas considérées comme des tables mutantes.

Table mutante

```
CREATE OR REPLACE TRIGGER TrigMutant1
 AFTER INSERT ON Trace FOR EACH ROW
DECLARE
 v_nombre NUMBER;
BEGIN
   SELECT COUNT(*) INTO v_nombre
          FROM Trace;
   DBMS_OUTPUT.PUT_LINE
     ('Nombre de traces : '
                || v_nombre);
END;
/
```

```
INSERT INTO Trace VALUES ('Insertion
le ' || TO_CHAR(SYSDATE,'DD-MM-YYYY
HH24:MI:SS'));
ERREUR à la ligne 1 :
ORA-04091: table SOUTOU.TRACE en
mutation, déclencheur/fonction ne
peut la voir
ORA-06512: à "SOUTOU.TRIGMUTANT1",
ligne 4
ORA-04088: erreur lors d'exécution du
déclencheur 'SOUTOU.TRIGMUTANT1'
```


6.9 Ordre d'exécution

- La séquence d'exécution des déclencheurs est théoriquement la suivante. En pratique certaines exécutions peuvent ne pas suivre cet ordre !
  - Tous les déclencheurs d'état BEFORE ;
  - Analyse de toutes les lignes affectées par l'instruction SQL;
  - Tous les déclencheurs de lignes BEFORE ;
  - Verrouillage, modification et vérification des contraintes d'intégrité ;
  - Tous les déclencheurs de lignes AFTER ;
  - Vérification des contraintes ;
  - Tous les déclencheurs d'état AFTER.

6.9 Ordre d'exécution L'option FOLLOWS

- Bien qu'Oracle permette que plusieurs déclencheurs soient programmés pour le même événement, il n'était pas possible de connaître l'ordre dans lequel les déclencheurs s'exécutaient.
- Depuis la version 11*g*, la directive FOLLOWS précise cet ordre.
- La syntaxe simplifiée qui permet la déclaration d'un tel déclencheur est la suivante :
  - CREATE [OR REPLACE] TRIGGER [*schéma.*] *nomTrigger*
    ... **FOLLOWS** [*schéma.*] *nomTriggerQuiSexecuteAvant* ...
    BEGIN
    ...
    END;
    /

Exemple 8

- CREATE OR REPLACE TRIGGER Trig_follows_1
  BEFORE INSERT ON TypeAvion FOR EACH ROW
  BEGIN
  DBMS_OUTPUT.put_line('Trig_follows_1 en exécution');
  END;
  /
- CREATE OR REPLACE TRIGGER Trig_follows_2
  BEFORE INSERT ON TypeAvion FOR EACH ROW
  BEGIN
  DBMS_OUTPUT.put_line('Trig_follows_2 en exécution');
  END;
  /

Si on désire que le premier déclencheur se lance toujours après le deuxième, il faut recompiler ce dernier de la manière suivante (il n'est pas possible de faire une référence avant, à savoir déclarer un déclencheur référençant un déclencheur inexistant) :

- CREATE OR REPLACE TRIGGER Trig_follows_1
  BEFORE INSERT ON TypeAvion FOR EACH ROW
  FOLLOWS Trig_follows_2
  BEGIN
  DBMS_OUTPUT.put_line('Trig_follows_1 en exécution');
  END;
  /

TD3.3 Les triggers
Ex1

Exercice 1 :

1. Soit l'extrait suivant du schema relationnel Gestpersonnel :
   - personne (id_pers, nompers, fonction, date_debFonction, numchef, salaire, #dept) – *table de description d'un employé*
   - projet (id_projet,titre, resp_proj) – *tablede description des projets*
   - persprojet (#id_proj, #id_pers, quota) *table de partcipation d'un employé donné à un projet*
   - job_history(#id_pers, Date_deb, Date_fin, fonction, # dept) ;

Nous présentons le jeu de données suivant :

persprojet :

id_proj	id_pers	quota
Proj10	100	10
Proj10	101	
Proj10	102	100
Proj13	103	
Proj13	104	
Proj13	101	50

Projet

Id_projet	titre	resp_proj
Proj10	projetAA	101
Proj13	projetBB	103

1. Créer un programme permettant d'automatiser la règle de gestion suivante :
   Refuser de nommer responsable d'un projet quelqu'un qui n'y participe pas.
   - Faites appel à une fonction f_verifparticipation qui vérifie la participation d'un employé donné à un projet,

Exemples d'exécution

```
Insert into projet (id_projet, resp_proj) values ('proj10', 103) ;
Update projet set resp_proj=103 where upper(id_projet)=upper('proj10')
                  *
ERROR at line 1:
ORA-20100: cet employé ne participe pas au projet !
```


function f_verifParticipation

```
Create or replace function f_verifParticipation
( emp personne.id_pers%type, proj
projet.id_projet%type) return Boolean is
   participe boolean :=false ;
   vnb number;
   Begin
   Select count(*) into vnb from persprojet
   where id_pers = emp and upper(id_proj)= upper(proj);
   if vnb>0 then participe:= true; end if;
   return participe ;
End f_ verifParticipation ;
```

Trigger Resp

```
Create or replace trigger trigResp
   BEFORE insert or update of resp_proj on projet
   For each row
   When (new.resp_proj is not null)
   Declare
   participe boolean ;
   Begin
   Participe:=f_verifparticipation(:new.resp_proj,
   :new.id_projet);
   If (participe =false) then
   raise_application_error(-20100,'cet employé ne participe
   pas au projet !') ;
   End if ;
End ;
```


Question 2

2. Pour interdire que la somme des quotas de participation d'un employé aux projets ne dépasse les 100h , nous avons dévéloppé le trigger suivant :

persprojet :

id_proj	id_pers	quota
Proj10	100	10
Proj10	101	
Proj10	102	100
Proj13	103	
Proj13	104	
Proj13	101	50

```
Create or replace trigger trgheursProjet
before insert or update on  persprojet
For each row when (new.quota is not null)
Declare
somme integer;
Begin
Select sum(quota) into somme from persprojet where id_pers=:new.id_pers ;
If somme+:new.quota > 100 then raise_application_error(-20200,'attention au surmenage');
End if;
End;
```

Qu'en pensez vous ? si on tente d'exécuter la ligne suivante ?

SQL> Update persprojet set quota=55 where id_pers=101 and upper(id_proj)= upper('proj10') ;

ERROR at line 1:
ORA-04091: table PERSPROJET is mutating, trigger/function may not see it
ORA-06512: at "TP2SVI.TRGHEURSPROJET", line 4
ORA-04088: error during execution of trigger 'TP2SVI.TRGHEURSPROJET'

3.

- personne (id_pers, nompers, fonction, date_debFonction, numchef, salaire, #dept) – *table de description d'un employé*
- projet (id_projet,titre, resp_proj) – *tablede description des projets*
- persprojet (#id_proj, #id_pers, quota) *table de partcipation d'un employé donné à un projet*
- job_history(#id_pers, Date_deb, Date_fin, fonction, # dept) ;

3. Un employé peut changer de poste ou de département au sein de la même entreprise durant sa carrière professionnelle. Toute modification doit être accompagnée par un enregistrement de la trace de son emploi passé dans la table job_history. Programmer cette règle de gestion.

```
CREATE OR REPLACE TRIGGER TrigMAJEmploie
AFTER UPDATE OF job_id, department_id ON employees
FOR EACH ROW

BEGIN
Insert into job_history VALUES (:old.id_pers,:old.date_debFonction,
    sysdate, :old.fonction, :old.department_id);
END TrigMAJEmploie;
   /
```


7. Déclencheurs LDD & d'instance

7.1. Déclencheurs LDD

- Ils réagissent aux modifications de la structure de la base de données et non plus à la modification des données de la base.
- Ils sont sensibles aux options **BEFORE** et **AFTER**
- La directive **DATABASE** précise que le déclencheur peut s'exécuter à partir d'un événement provoqué par n'importe quel schéma
- La directive **SCHEMA** précise que le déclencheur peut s'exécuter à partir d'un événement provoqué par le schéma lui même
- Les ordres LDD pouvant provoquer l'exécution du déclencheur sont : **ALTER**,, **CREATE**, **DROP**, **GRANT**, **RENAME**, **REVOKE**

7.2. Syntaxe de création des triggers LDD

CREATE [OR REPLACE] TRIGGER <nom_trigger>
BEFORE | AFTER

{{ ***évenement LDD***[OR *événementLDD* } (regroupe non seulement des ORDRES LDD
ON { [*schéma.*] SCHEMA | DATABASE } }

[DECLARE]
-- déclaration de variables, exceptions,
-- curseurs
BEGIN
-- corps du trigger
<Action(s)>
<Appel Pocédures|fct PLSQL ou externes
END;/

CALL Procedures|fonctions PLSQL ou externes (exp JAVA)

7.3. Principaux évènements déclencheurs des triggers LDD

- Les principaux évènements sur la structure de la base prise en compte sont :
  - ALTER pour déclencher en cas de modification d'un objet du dictionnaire (table, index, séquence, etc.).
  - COMMENI' pour déclencher en cas d'ajout d'un commentaire.
  - CREATE pour déclencher en cas d'ajout d'un objet du dictionnaire.
  - DROP pour déclencher en cas de suppression d'un objet du dictionnaire.
  - GRANT pour déclencher en cas d'affectation de privilège à un autre utilisateur ou rôle.
  - RENAME pour déclencher en cas de changement de nom d'un objet du dictionnaire.
  - REVOKE pour déclencher en cas de révocation de privilège d'un autre utilisateur ou rôle

Exemple

- Exemple 9 : Le déclencheur suivant interdit toute suppression d'objet, dans le schéma « soutou », se produisant un lundi ou un vendredi.

Code PL/SQL	Commentaires
`CREATE TRIGGER surveilleDROPSoutou` `  BEFORE DROP ON soutou.SCHEMA`	Événement déclencheur LDD.

`BEGIN`
`  IF TO_CHAR(SYSDATE,'DAY') IN`
`  ('LUNDI  ','VENDREDI') THEN`
`  RAISE_APPLICATION_ERROR(-20104,'Désolé pas`
`  de destruction ce jour..') ;`
`  END IF ;`
`END;`
`/`
	Corps du déclencheur.
	
	Retour d'une erreur.

7.4. Déclencheurs d'instances

- Ils sont déclenchés suite à des événements systèmes comme :
  - **LOGON, STARTUP, SERVERRROR, SUSPEND** (utilisés avec l'option **AFTER**)
  - **LOGOFF, SHUTDOWN** (utilisés avec t l option **BEFORE**)
- Des événements comme **AFTER STARUP** et **BEFORE SHUTDOWN** s'appliquent avec des déclencheurs de type **DATABASE.**

7.5. Syntaxe de création des triggers d'instance

CREATE [OR REPLACE] TRIGGER <nom_trigger>
BEFORE | AFTER

STRATUP| SUHTDOWN| SUSPEND| SERVERERROR| LOGON| LOGOFF
ON { [schéma.] SCHEMA | **DATABASE** } }

[DECLARE] -- déclaration de variables, exceptions, -- curseurs	CALL Procedures|fonctions PLSQL ou externes (exp JAVA
BEGIN -- corps du trigger <Action(s)> [Appel Pocédures| fonctions PLSQL| Externes] END;/	

Exemples

- Exemple 1: Le déclencheur suivant insère une ligne dans une table qui indique l'utilisateur et l'heure de déconnexion (sous SQLtiPlus, via un programme d'application, etc.). On suppose la table Trace(événement VARCHAR2(100)) créée.

Code PL/SQL	Commentaires
`CREATE TRIGGER espionDéconnexion` `BEFORE LOGOFF ON DATABASE`	Événement déclencheur.

`BEGIN`
`INSERT INTO Trace VALUES (USER ||`
`' déconnexion le ' ||`
`TO_CHAR(SYSDATE,'DD-MM-YYYY HH24:MI:SS'));`
`END;`
`/`
	Corps du déclencheur exécuté à chaque déconnexion.

- Exemple2:

```
CREATE TRIGGER espionconnexion
AFTER LOGON ON DATABASE
CALL user1.sousProgDeclenr (SYSDATE)
/
```

Appel direct d'une procédure PUSQL

8. Maintenance des déclencheurs

- Un trigger est complètement compilé lorsque l'instruction CREATE TRIGGER est appelée.
- Recompiler un trigger:
  - ALTER TRIGGER COMPILE
- Remplacer et recompiler un trigger défini
  - CREATE OR REPLACE TRIGGER Supprimer un trigger
- Supprimer un trigger:
  - DROP TRIGGER
- Activation/désactivation d'un déclencheur.
  - Il est possible de désactiver un déclencheur avec la commande suivante
    ALTER TRIGGER nom_déclencheur DISABLE.
  - et de l'activer avec la commande suivante
    ALTER TRIGGER nom_déclencheur ENABLE
- De la même façon, on peut désactiver tous les déclencheurs définis sur une table :
  - ALTER TABLE nom_table DISABLE ALL TRIGGERS
  - et de les activer avec la commande suivante
    ALTER TABLE nom_table ENABLE ALL TRIGGERS

9. Dictionnaire de données

- Les informations sur les déclencheurs sont visibles à travers les vues du dictionnaire de données.
  - **USER_TRIGGERS** pour les déclencheurs appartenant au schéma
  - **ALL_TRIGGERS** pour les déclencheurs appartenant aux schémas accessibles
  - **DBA_TRIGGERS** pour les déclencheurs appartenant à tous les schémas

```
Exp: Select base_object_type, trigger_type, triggering_event, trigger_body
FROM DBA_TRIGGERS where upper(trigger_name)=upper('Alerte_maj_client');
```

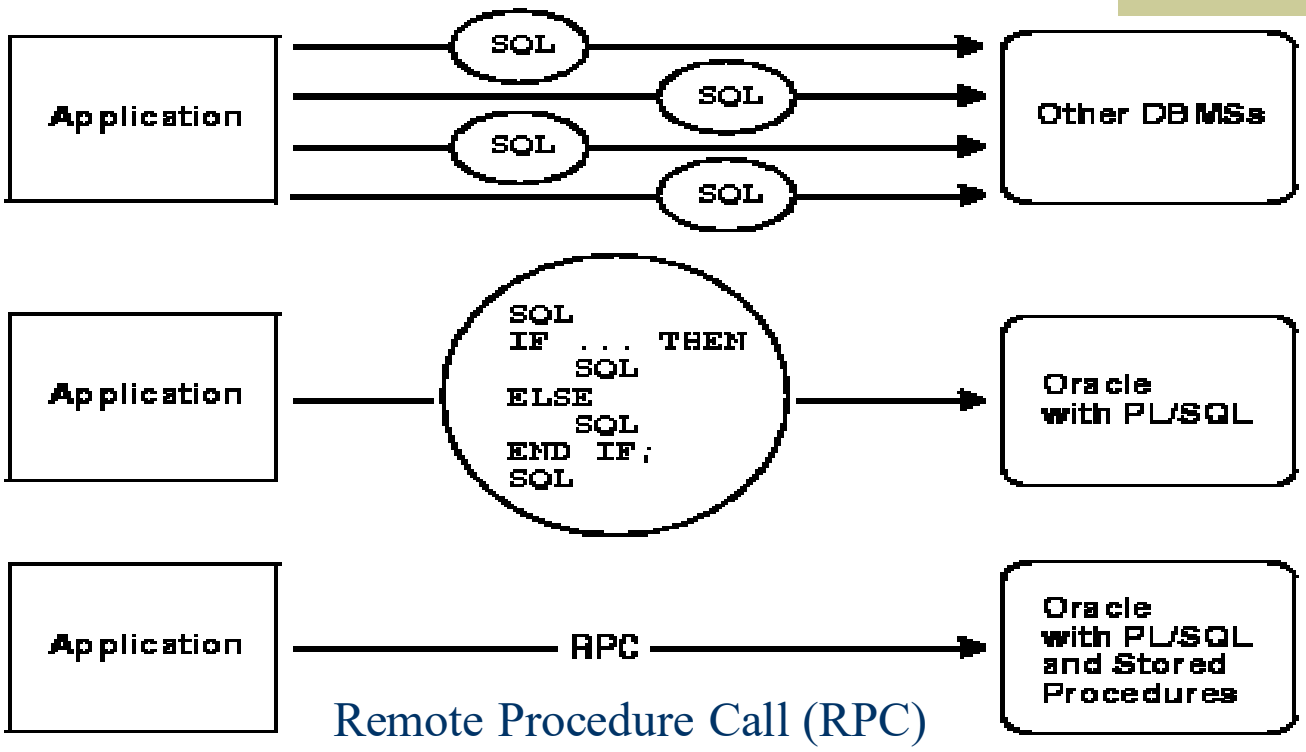
BASE_OBJECT_TYPE	TRIGGER_TYPE	TRIGGERING_EVENT	TRIGGER_BODY
TABLE	BEFORE STATEMENT	INSERT OR UPDATE OR DELETE	BEGIN IF TO_CHAR(SYSDATE,'HH24:MI') >='20:00'then Raise_application_error(-20

- La colonne BASE_OBJECT_TYPE permet de savoir si le déclencheur est basé sur une table, une vue, un schéma ou la totalité de la base .
- La colonne TRIGGER_TYPE permet de savoir s'il s'agit d'un déclencheur
  - BEFORE, AFTER ou INSTEAD OF
  - si son mode est FOR EACH ROW ou non
- La colonne TRIGGERING_EVENT permet de connaître l'évènement concerné par le déclencheur
- La colonne TRIGGER_BODY contient le code du bloc PL/SQL

Conclusion: Avantages de PL/SQL

- PL/SQL est un langage portable et performant dans le traitement de transactions.
- Il offre les avantages suivants :
  1. Support de SQL
  2. Support de la programmation orientée objet
  3. Meilleure performance
  4. Productivité plus élevée
  5. Meilleure portabilité
  6. Grande interaction avec Oracle
  7. Sécurité accrue
  8. Gestion des erreurs

Meilleure performance



Remote Procedure Call (RPC)

Références

Livres:

1. J.GABILLAUD: Oracle 12c: SQL, PLSQL, SQL plus, edition ENI,: mars 2015
2. C.SOUTOU: SQL pour Oracle, 7ème édition, Eyrolles 02/04/2015, Paris.

Supports de cours

1. Anne VILNAT, Cours 4 : PL/SQL : ou comment faire plus avec ORACLE, 2eme partie, Limsi
2. M.BOUGHANEM, Contraintes d'intégrité complexes et déclencheurs (triggers), Université Paul Sabatier.

Exercices LDD & Instance

- **Exercice 2**

  Règles de Gestion :

  1. Toute manipulation de la structure de base de données doit être effectuée durant les heures de travail (de lundi à vendredi entre
  08 et 18 heures) sinon la manipulation doit être refusée.

```
CREATE OR REPLACE trigger trigctrModificationControlee
BEFORE CREATE or Drop OR ALTER on database
BEGIN
IF TO_CHAR (SYSDATE, 'HH24:MI') <'08:00' OR TO_CHAR (SYSDATE,
'HH24:MI')> '18:00'
OR TO_CHAR (SYSDATE, 'DAY') IN ('SAMEDI', 'DIMANCHE') THEN
RAISE_APPLICATION_ERROR (-20205, ' La modification dans les jours fériés et
hors des heures de travail est interdite ') ;
END IF;
End trigctrModificationControlee;
```


Résultat lors d'une manipulation après 18h

- Un evenement de creation d'un nouvel utilisateur de n'importe quel schéma y compris le schéma système:
  - Erreur commençant à la ligne: 1 de la commande -
  - CREATE user user1 identified by user1
  - Rapport d'erreur -
  - Erreur SQL : ORA-00604: error occurred at recursive SQL level 1
  - ORA-20205: La modification dans les jours fériés et hors des heures de travail est interdite
  - ORA-06512: at line 4
  - 00604. 00000 - "error occurred at recursive SQL level %s"
  - *Cause: An error occurred while processing a recursive SQL statement
  - (a statement applying to internal dictionary tables).
  - *Action: If the situation described in the next error on the stack
  - can be corrected, do so; otherwise contact Oracle Support.

Exercices (suite)

- **2.** Soit la table pointeuse définie comme suit :
  Create table pointeuse(utilisateur varchar(50), msg varchar2(100), dateheure varchar2(100) ;
  Créer un programme permettant de retracer toute les connexions à la base en particulier à chaque connexion, insérer dans la table
  de pointage l'utilisateur connecté, un message 'connecté le : date de connexion (jour mois annee heure, minutes et secondes)

- Correction
- Create or replace trigger trigconnexion
- After logon on database
- Begin
- Insert into pointeuse values (USER, 'connecté le : '| |,to_char(sysdate, 'DD-MM-YYYY HH :MI :SS')) ;
- End trigconnexion ;

Résultats

- Résultat juste après le programme:
- conn user1/user1
  Connected.
  SQL> select * from pointeuse

UTILISATEUR

MSG

DATEHEURE
---
user1
connecté le :
15-04-2020 20 :07 :03

Références

Livres:

1. J.GABILLAUD: Oracle 12c: SQL, PLSQL, SQL plus, edition ENI,: mars 2015
2. C.SOUTOU: SQL pour Oracle, 7ème édition, Eyrolles 02/04/2015, Paris.

Supports de cours

1. Anne VILNAT, Cours 4 : PL/SQL : ou comment faire plus avec ORACLE, 2eme partie, Limsi
2. M.BOUGHANEM, Contraintes d'intégrité complexes et déclencheurs (triggers), Université Paul Sabatier.