



**Faculté des Sciences**

**Département d'informatique**

# **BASES DE DONNEES II**

**SMI-S6**

**Pr. M.E.H CHARAF  
2021-2022**

Faculté des Sciences  
Département d'informatique  
SMI-S6



## **• AFTER ou BEFORE ? INSTEAD OF?**

En général, vous utilisez les déclencheurs BEFORE ou AFTER pour obtenir les résultats suivants :

- Utilisez les triggers **BEFORE row** pour modifier la rangée avant que les données de la rangée ne soient écrites sur le disque.
- Utilisez les triggers **AFTER row** pour obtenir, ou effectuer des opérations, en utilisant l'ID de la rangée

Pr. M.E.H CHARAF

### • AFTER ou BEFORE ? INSTEAD OF?

#### • Exemples:

- Si le trigger doit déterminer si l'instruction DML est autorisée : **utiliser BEFORE**
- Si on a besoin que l'instruction DML soit terminée pour exécuter le corps du trigger : **utiliser AFTER**
- Le trigger **INSTEAD OF** permet par exemple de contourner l'interdiction de mise à jour au travers de vues

Pr. M.E.H CHARAF

### • BEFORE DELETE TRIGGER

#### • Exemples:

1. Restreindre une opération DELETE invalide.
2. Supprimer des données d'une autre table.

Pr. M.E.H CHARAF

### • BEFORE DELETE TRIGGER

- **Articles** (ITEM\_ID number(10) primary key, NAME varchar2(30), TYPE varchar2(50), PRICE\_IN\_DOLLAR number(10));
- **Commandes** (ORDER\_ID number(10) primary key, ITEM\_ID number(10), QTY number(5), ORD\_DATE date, **STATUS** varchar2(20) );

Pr. M.E.H CHARAF

### • BEFORE DELETE TRIGGER

#### 1. Restreindre une opération DELETE invalide.

- Dans cet exemple, nous avons deux tables : Articles et Commandes.
- La table **Commandes** contient les valeurs des commandes d'achat des articles de la table **Articles**.
- Maintenant, lorsque l'utilisateur veut supprimer un article de **Articles**, nous devons vérifier si une commande **PENDING** existe pour cet article ou non.
- Si une commande « En cours » (PENDING) est trouvée, alors nous n'autoriserons pas la suppression de l'article et nous leverons une erreur d'application à partir du TRIGGER BEFORE DELETE pour restreindre l'opération de suppression sur Articles.

Pr. M.E.H CHARAF

### • BEFORE DELETE TRIGGER

CREATE OR REPLACE TRIGGER trg\_before\_item\_delete  
BEFORE DELETE on Articles FOR EACH ROW

DECLARE

Cdes\_incompletes number:=0;

BEGIN

SELECT count(\*) INTO Cdes\_incompletes FROM Commandes  
WHERE item\_id = :OLD.item\_id AND STATUS = 'PENDING';

IF (pending\_orders > 0) THEN

RAISE\_APPLICATION\_ERROR(-20000, 'Compléter la Cde avant suppression');

END IF;

END;

Pr. M.E.H CHARAF

### • BEFORE DELETE TRIGGER

## 2. Supprimer des données d'une autre table.

- Dans cet exemple, nous avons deux tables : **patient** et **patient\_details**.
- La table **patient** contient les détails de base tandis que **patient\_details** contient les valeurs d'un patient telles que la maladie, le nom du médecin, etc.
- Maintenant, chaque fois que l'utilisateur veut supprimer les données du **patient**, **nous avons besoin de supprimer les données de patient\_details aussi** comme nous n'en avons plus besoin après un patient est supprimé.
- Donc ici, nous allons supprimer les données par BEFORE DELETE TRIGGER sur la table du patient.

Pr. M.E.H CHARAF

### • BEFORE DELETE TRIGGER

- **PATIENT** ( PATIENT\_ID number(10) primary key, NAME varchar2(30), PHONE\_NO number(12));
- **PATIENT\_DETAILS** ( PD\_ID number(10) primary key, PATIENT\_ID number(10), DISEASE varchar2(50), ADMITTED\_DATE date, DOCTOR varchar2(30));

Pr. M.E.H CHARAF

### PL/SQL: Déclencheurs (Triggers) – Compléments- **AFTER ou BEFORE**

### • BEFORE DELETE TRIGGER

```
CREATE OR REPLACE TRIGGER trg_delete_from_details
    BEFORE DELETE on Patient FOR EACH ROW

BEGIN

    DELETE FROM Patient_Details PD
    WHERE PD.PATIENT_ID = :OLD.PATIENT_ID;

END;
```

### • BEFORE UPDATE TRIGGER

#### • Exemples:

1. Validation des données
2. Mise à jour automatique des valeurs
3. Data logging, or auditing

Pr. M.E.H CHARAF

### • BEFORE UPDATE TRIGGER

1. Validation des données

Supposons qu'une entreprise a des offres d'emploi et disposent déjà de données de candidature et que les critères sont :

- ☐ L'expérience professionnelle doit être supérieure ou égale à 3 ans et
- ☐ Une candidature précédente ne doit pas avoir été faite au cours des deux dernières années.

Pour assurer l'intégrité des données, nous allons créer un déclencheur **BEFORE UPDATE** qui limitera la mise à jour des données qui violent l'un des critères ci-dessus.

Pr. M.E.H CHARAF



### • BEFORE UPDATE TRIGGER

- **Jobs** ( **APPLICATION\_ID** number(10) primary key, **FIRST\_NAME** varchar2(50), **LAST\_NAME** varchar2(50), **JOB\_EXPERIENCE** number(2), **LAST\_APPLIED\_DATE** date );
- Ensuite nous allons créer un trigger before update sur la colonne **JOB\_EXPERIENCE** et **LAST\_APPLIED\_DATE** de la table **Jobs**.

Pr. M.E.H CHARAF

### • BEFORE UPDATE TRIGGER

```
CREATE OR REPLACE TRIGGER trg_before_emp_update
BEFORE UPDATE OF JOB_EXPERIENCE, LAST_APPLIED_DATE on Jobs FOR EACH ROW
DECLARE
years_since_last_applied number(5): = -1;
BEGIN
IF (:NEW.LAST_APPLIED_DATE IS NOT NULL) THEN
SELECT MONTHS_BETWEEN(TO_DATE(sysdate, 'DD-MON-YYYY'), TO_DATE(:NEW.LAST_APPLIED_DATE, 'DD-MON-YYYY'))/12
INTO years_since_last_applied FROM dual;
IF (years_since_last_applied <= 2) THEN
RAISE_APPLICATION_ERROR(-20000, 'Previous application attempt must not be done in last 2 years.');
```

Pr. M.E.H CHARAF

### • BEFORE INSERT TRIGGER

#### • Exemples:

1. Validation des données
2. MAJ automatique des valeurs (Exemple CREATED\_BY, CREATION\_DATE...)

Pr. M.E.H CHARAF

### • BEFORE INSERT TRIGGER

1. MAJ automatique des valeurs (Exemple CREATED\_BY, CREATION\_DATE...)

Employee (EMP\_ID number(10) primary key, FIRST\_NAME varchar2(50), LAST\_NAME varchar2(50), DATE\_OF\_BIRTH date, CREATED\_BY varchar2(20), CREATED\_DATE date);

Nous allons créer un déclencheur BEFORE INSERT pour mettre à jour automatiquement les valeurs CREATED\_BY et CREATED\_DATE.

Pr. M.E.H CHARAF



### • BEFORE INSERT TRIGGER

```
CREATE OR REPLACE TRIGGER trg_before_emp_insr_userinfo  
BEFORE INSERT ON Employee FOR EACH ROW
```

```
DECLARE
```

```
    username varchar2(20);
```

```
BEGIN
```

```
SELECT USER INTO username FROM dual;
```

```
:NEW.CREATED_BY := username;
```

```
:NEW.CREATED_DATE := sysdate;
```

```
END;
```

Pr. M.E.H CHARAF

### • AFTER INSERT TRIGGER

#### 1. Tracer des données manquantes

Dans cet exemple,

Si un nouvel utilisateur est créé dans **USER\_DETAILS**, mais que des champs comme **passport\_no** ou **Permis\_no** sont manquants

→ Un nouvel enregistrement sera inséré dans **USER\_RAPPELS** via le déclencheur '**AFTER INSERT**' sur **USER\_DETAILS**.

Pr. M.E.H CHARAF

\*

### • AFTER INSERT TRIGGER

- **USERS** (USER\_ID number(10) primary key, USER\_NAME varchar2(15), EMAIL varchar2(20), PHONE number(12), **PASSPORT\_NO** varchar2(10), **Permis\_NO** varchar2(10));
- **RAPPELS** ( USER\_ID number(10), REMINDER\_TEXT varchar2(200), REMINDER\_DATE date, STATUS varchar2(10));
- Ensuite nous allons créer un **trigger AFTER INSERT** sur la table **USERS**

Pr. M.E.H CHARAF

```
CREATE OR REPLACE TRIGGER Trig AFTER INSERT on USERS FOR EACH ROW
DECLARE
    counter number(2):=0;
    rappel_text varchar2(200):= '';

BEGIN
    IF (:NEW.PASSPORT_NO = '' OR :NEW.PASSPORT_NO is null) THEN
        rappel_text := 'Merci de renseigner le numero de passeport';
        counter := counter+1;
    END IF;
    IF (:NEW.Permis_NO = '' OR :NEW.Permis_NO is null) THEN
        rappel_text := rappel_text || 'Merci de renseigner le numero de permis.';
        counter := counter+1;
    END IF;
    IF (counter > 0) THEN
        INSERT INTO REMINDERS VALUES (:NEW.USER_ID, rappel_text, sysdate+3, 'Manquant');
    END IF;
END;
```

Pr. M.E.H CHARAF

## • AFTER UPDATE TRIGGER

### 1. Tracer des changements (Logs)

Dans cet exemple,

Après chaque mise à jour de la colonne 'SALARY' de **EMPLOYEES**, un déclencheur '**AFTER UPDATE**' sera activé

→ Les nouvelles données mises à jour seront insérées dans une table **EMPLOYEES\_LOG**, à des fins d'audit.

Pr. M.E.H CHARAF

## • AFTER UPDATE TRIGGER

- **EMPLOYEES** ( EMP\_ID number(10), SALARY number(10), EMP\_NAME varchar2(50));
- **EMPLOYEES\_LOG** (EMP\_ID number(10), **NEW\_SALARY** number(10), **UPDATED\_DATE** date, **UPDATED\_BY** varchar2(20));
- Ensuite nous allons créer un **tigger AFTER UPDATE** sur la table **EMPLOYEES**

Pr. M.E.H CHARAF

### • AFTER UPDATE TRIGGER

CREATE OR REPLACE TRIGGER trig\_log AFTER UPDATE OF **SALARY** ON  
**EMPLOYEES** FOR EACH ROW

DECLARE

username varchar2(20);

BEGIN

SELECT USER INTO username FROM dual;

INSERT INTO **EMPLOYEES\_LOG** VALUES (  
:NEW.EMP\_ID, :NEW.SALARY, sysdate, username);

END;

Pr. M.E.H CHARAF

### • AFTER DELETE TRIGGER

#### 1. Tracer des suppressions (Audit Suppressions d'une Table)

Dans cet exemple,

Si l'utilisateur a supprimé une ligne de la Table **MEDICAL\_FACTURES**:

→ La ligne supprimée sera insérée dans **MEDICAL\_FACT\_HISTORY** par le déclencheur '**AFTER DELETE**' sur la table **MEDICAL\_FACTURES**

Pr. M.E.H CHARAF

### • AFTER DELETE TRIGGER

- **MEDICAL\_FACTURES** (Fact\_ID number(10) primary key, Fact\_NUMBER varchar2(20), PARTY\_NAME varchar2(50), Fact\_DATE date, CREATED\_BY varchar2(20), CREATED\_DATE date);
- **MEDICAL\_FACT\_HISTORY** (Fact\_ID number(10), Fact\_NUMBER varchar2(20), PARTY\_NAME varchar2(50), Fact\_DATE date, **DELETED\_BY** varchar2(20), **DELETED\_DATE** date);
- Ensuite nous allons créer un déclencheur **AFTER DELETE** sur la table **MEDICAL\_FACTURES**

Pr. M.E.H CHARAF

### • AFTER DELETE TRIGGER

```
CREATE OR REPLACE TRIGGER trig_Fact AFTER DELETE ON
MEDICAL_FACTURES FOR EACH ROW

DECLARE
    username varchar2(10);

BEGIN

    SELECT user INTO username FROM dual;
    INSERT INTO MEDICAL_FACT_HISTORY VALUES
        (:OLD.FACT_ID, :OLD.FACT_NUMBER, :OLD.PARTY_NAME,
        :OLD.FACT_DATE, USERNAME, SYSDATE);

END;
```

Pr. M.E.H CHARAF