

# Déclencheurs

Nicolas Travers

# Contenu du cours

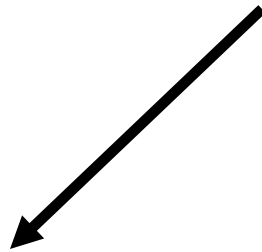
- Triggers
  - Principes
  - Déclenchement

(D'après les supports de Michel Crucianu, Cédric du Mouza et Philippe Rigaux)

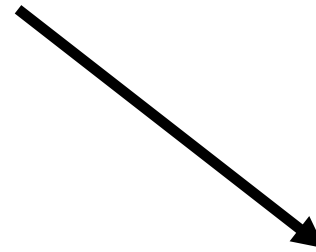
# Déclencheurs

- *Trigger*
  - Programme **déclenché** par un événement
  - **N'est pas appelé explicitement** par une application
- Événements déclencheurs :
  - Instruction LMD : **INSERT, UPDATE, DELETE**
  - Instruction LDD : **CREATE, ALTER, DROP**
  - Démarrage ou arrêt de la base
  - Connexion ou déconnexion d'utilisateur
  - Erreur d'exécution
- Usage fréquent
  - Contraintes non exprimables sur les tables
  - Modification des dépendances

- ◆ Un trigger de type Événement/Action (EA) est composé d'

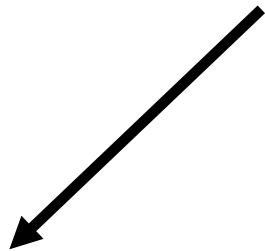


un événement qui va le  
**déclencher**

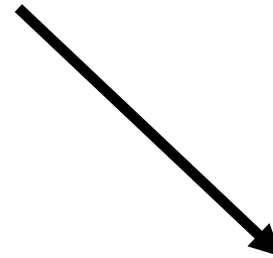


une action à **exécuter**  
au déclenchement

◆ Un trigger est associé à une **table**



L'événement qui le déclenche est une **opération** (insertion, suppression, MAJ) sur cette relation;



L'action à exécuter est spécifiée par un bloc **PL/SQL**.

# Définition d'un déclencheur

- Structure :
  - Description de l'événement
  - Condition supplémentaire à satisfaire (optionnel)
  - Traitement à réaliser
- Syntaxe pour déclenchement sur instruction LMD :

```
CREATE [OR REPLACE] TRIGGER nomDeclencheur
{BEFORE | AFTER | INSTEAD OF}
{DELETE | INSERT | UPDATE [OF colonne 1, ...] [OR ...]}
ON {nomTable | nomVue}
[REFERENCING {OLD [AS] nomAncien | NEW [AS] nomNouveau
              | PARENT [AS] nomParent } ...]
[FOR EACH ROW]
[WHEN conditionSupplementaire]
{[DECLARE ...] BEGIN ... [EXCEPTION ...] END;
 | CALL nomSousProgramme(listeParametres)}
```

# Déclencheurs sur instruction LMD

- Quand le **déclenchement** a lieu ?
  - Avant l'événement : **BEFORE**
  - Après l'événement : **AFTER**
  - À la place de l'événement : **INSTEAD OF**  
(uniquement pour vues multi-tables)
- Description de l'événement :
  - La ou les (**OR**) instructions,
  - Si l'événement concerne des colonnes spécifiques (**[OF colonne 1, ...]**) ou non,
  - Le nom de la table (ou vue) (**ON {nomTable | nomVue}**)

# Déclencheurs sur instruction LMD

- Changement des noms par défaut : **REFERENCING**
  - **:OLD** désigne un enregistrement à effacer (déclencheur sur **DELETE, UPDATE**) : **REFERENCING OLD AS nomAncien**
  - **:NEW** désigne un enregistrement à insérer (déclencheur sur **INSERT, UPDATE**) : **REFERENCING NEW AS nomNouveau**
  - **:PARENT** pour des *nested tables* : **REFERENCING PARENT AS nomParent**
- **FOR EACH ROW** :
  - Avec **FOR EACH ROW**, 1 exécution par ligne concernée par l'instruction LMD (*row trigger*)
  - Sans **FOR EACH ROW**, 1 exécution par instruction LMD (*statement trigger*)



# Base exemple

- Schéma :

Immeuble (Adr, NbEtg, DateConstr, NomGerant)

Appart (Adr, Num, Type, Superficie, Etg, NbOccup)

Personne (Nom, Age, CodeProf)

Occupant (Adr, NumApp, NomOccup, DateArrivee, DateDepart)

Propriete (Adr, NomProprietaire, QuotePart)

- Exemples de contraintes à satisfaire :

- `propriete` : **CONSTRAINT** `prop_pers` **FOREIGN KEY** (`NomProprietaire`) **REFERENCES** `personne` (`Nom`)
- `occupant` : **CONSTRAINT** `dates` **CHECK** (`DateArrivee` < `DateDepart`)
- Règles de gestion :
  - $\sum$  quotes-parts (propriété) = 100 ;
  - $\forall o \in \text{occupant} \mid o.\text{DateArrivee} > \text{Immeuble}.\text{dateConstr}$
  - ... *exercice* : trouver d'autres contraintes
  - → déclencheurs si erreur

# Déclencheur sur INSERT

- Pour un nouvel occupant,
  - Si `occupant.DateArrivee > immeuble.DateConstr`
  - (`FOR EACH ROW` est nécessaire pour avoir accès à `:NEW`, l'enregistrement ajouté) :

```
CREATE TRIGGER TriggerVerificationDates
    BEFORE INSERT ON occupant FOR EACH ROW
DECLARE
    Imm immeuble%ROWTYPE;
BEGIN
    SELECT * INTO Imm FROM immeuble
        WHERE immeuble.Adr = :NEW.Adr;
    IF :NEW.DateArrivee < Imm.DateConstr THEN
        RAISE_APPLICATION_ERROR(-20100, :NEW.Nom
    || ' arrivé avant construction immeuble ' ||
        Imm.Adr);
    END IF;
END;
```

# Déclencheur sur INSERT (2)

- Immeuble => au moins un appartement

```
CREATE TRIGGER TriggerAppartInitial
    AFTER INSERT ON immeuble FOR EACH ROW
BEGIN
    INSERT INTO appart (Adr, Num, NbOccup)
    VALUES (:NEW.Adr, 1, 0);
END;
```

- Exercice : faire le déclencheur pour la quote-part

# Déclencheur sur INSERT (3)

- Pour un nouvel appartement,
  - Si `appartement.nbOccupant > 0`
  - Renvoyer une erreur
  - (**WHEN** est nécessaire pour vérifier la condition)

```
CREATE TRIGGER TriggerVerificationAppartement
    AFTER INSERT ON appartement FOR EACH ROW
WHEN NEW.nbOccupant > 0
-- NEW ne prend pas de ':' dans la clause WHEN
BEGIN
    RAISE_APPLICATION_ERROR(-20101, '1
appartement ' || :NEW.Adr || ' Ne peut avoir
d'occupant non encore enregistré');
    END IF;
END;
```

# Déclencheur sur DELETE

- Suppression d'un occupant
  - Décrémenter `appart.NbOccup`
  - Après effacement
    - `FOR EACH ROW` est nécessaire (plusieurs occupants peuvent être concernés) => accès à `:OLD`

```
CREATE TRIGGER TriggerDiminutionNombreOccupants
  AFTER DELETE ON occupant FOR EACH ROW
BEGIN
  UPDATE appart SET NbOccup = NbOccup - 1
    WHERE appart.Adr = :OLD.Adr
      AND   appart.Num = :OLD.NumApp;
END;
```

# Déclencheur sur UPDATE

- Modification d'un occupant
  - Si changement d'adresse
  - Changer `appart.NbOccup` pour `:OLD` et `:NEW`

```
CREATE TRIGGER TriggerMAJNombreOccupants
  AFTER UPDATE ON occupant FOR EACH ROW
BEGIN
  IF :OLD.Adr <> :NEW.Adr OR
     :OLD.NumApp <> :NEW.NumApp
  THEN
    UPDATE appart SET NbOccup = NbOccup - 1
      WHERE appart.Adr = :OLD.Adr
      AND   appart.Num = :OLD.NumApp;
    UPDATE appart SET NbOccup = NbOccup + 1
      WHERE appart.Adr = :NEW.Adr
      AND   appart.Num = :NEW.NumApp;
  END IF;
END;
```

# Déclencheur sur conditions multiples

- **Un seul** déclencheur pour INSERT, DELETE, UPDATE

```
CREATE TRIGGER TriggerCompletMAJNombreOccupants
  AFTER INSERT OR DELETE OR UPDATE
  ON occupant FOR EACH ROW
BEGIN
  IF (INSERTING) THEN
    ...
  ELSIF (DELETING) THEN
    ...
  ELSIF (UPDATING) THEN
    ...
  END IF;
END;
```

- Exercice : Créer le déclencheur pour mettre à jour les valeurs de `appart.NbOccup` pour chaque cas.

# Déclencheurs sur instruction LDD

- Syntaxe pour déclenchement sur instruction LDD :

```
CREATE [OR REPLACE] TRIGGER nomDeclencheur
BEFORE | AFTER <action> [OR <action> ...]
ON {[nomSchema.] SCHEMA | DATABASE}
{[DECLARE ...] BEGIN ... [EXCEPTION ...] END;
| CALL nomSousProgramme(listeParametres)}
```

- **SCHEMA** : déclencheur valable pour schéma courant
- Quelques actions :
  - **CREATE**, **RENAME**, **ALTER**, **DROP** sur un objet du dictionnaire
  - **GRANT**, **REVOKE** privilège(s) à un utilisateur



# Déclencheur sur LDD : exemple

- Enregistrement des changements de noms des objets du dictionnaire :

```
historiqueChangementNoms (Date, NomObjet, NomProprietaire)
```

```
CREATE TRIGGER TriggerHistoriqueChangementNoms
  AFTER RENAME ON DATABASE
BEGIN
  -- On se sert de 2 attributs système
  -- ora_dict_obj_name   : nom objet affecté
  -- ora_dict_obj_owner : propriétaire objet
  affecté

  INSERT INTO historiqueChangementNoms
    VALUES (SYSDATE, ora_dict_obj_name,
            ora_dict_obj_owner);
END;
```

# Déclencheurs d'instance

- Syntaxe :

```
CREATE [OR REPLACE] TRIGGER nomDeclencheur
BEFORE | AFTER <evenement> [OR <evenement> ...]
ON {[nomSchema.]SCHEMA | DATABASE}
{[DECLARE ...] BEGIN ... [EXCEPTION ...] END;
 | CALL nomSousProgramme(listeParametres) }
```

- Événements déclencheurs concernés :

- Démarrage ou arrêt de la base :
  - **SHUTDOWN** ou **STARTUP**
- Connexion ou déconnexion d'utilisateur :
  - **LOGON** ou **LOGOFF**
- Erreurs :
  - **SERVERERROR, NO\_DATA\_FOUND, ...**

# Déclencheur d'instance : exemple

- Afficher l'identité de l'objet ayant provoqué un débordement :

```
CREATE TRIGGER TriggerDebordement
  AFTER SERVERERROR ON DATABASE
DECLARE
  eno NUMBER;
  typ VARCHAR2; owner VARCHAR2; ts VARCHAR2;
  obj VARCHAR2; subobj VARCHAR2;
BEGIN
  IF (space_error_info(eno, typ, owner, ts, obj, subobj)
  =
    TRUE) THEN
    DBMS_OUTPUT.PUT_LINE('L'objet' || obj ||
    ' de ' || owner || ' a débordé !');
  END IF;
END;
```

# Manipulation d'un déclencheur

- Tout déclencheur est actif dès sa compilation !
- Re-compilation d'un déclencheur après modification :

```
ALTER TRIGGER nomDeclencheur COMPILE;
```

- Désactivation de déclencheurs :

```
ALTER TRIGGER nomDeclencheur DISABLE;
```

```
ALTER TABLE nomTable DISABLE ALL TRIGGERS;
```

- Réactivation de déclencheurs :

```
ALTER TRIGGER nomDeclencheur ENABLE;
```

```
ALTER TABLE nomTable ENABLE ALL TRIGGERS;
```

- Suppression d'un déclencheur :

```
DROP TRIGGER nomDeclencheur;
```

# Déclencheurs et Concurrence

- Pas d'instruction
  - COMMIT
  - ROLLBACK
  - SAVEPOINT

# Conclusion

Un déclencheur permet de

- Contrôler l'intégrité,
- Maintenir des renseignements sur l'activité de la base,
- Rendre le SGBD "actif"
  - En calculant automatiquement certains attributs,
  - En insérant, modifiant ou supprimant des tuples,
- Contrôler dynamiquement certaines manipulations de la base,
- Rafraîchir des données dupliquées.

# Droits de création et manipulation

- Déclencheurs d'instance : privilège

**ADMINISTER DATABASE TRIGGER**

- Autres déclencheurs :

- Dans tout schéma : privilège

**CREATE ANY TRIGGER**

- Dans votre schéma : privilège

**CREATE TRIGGER (rôle RESOURCE)**