Le langage de contrôle de données

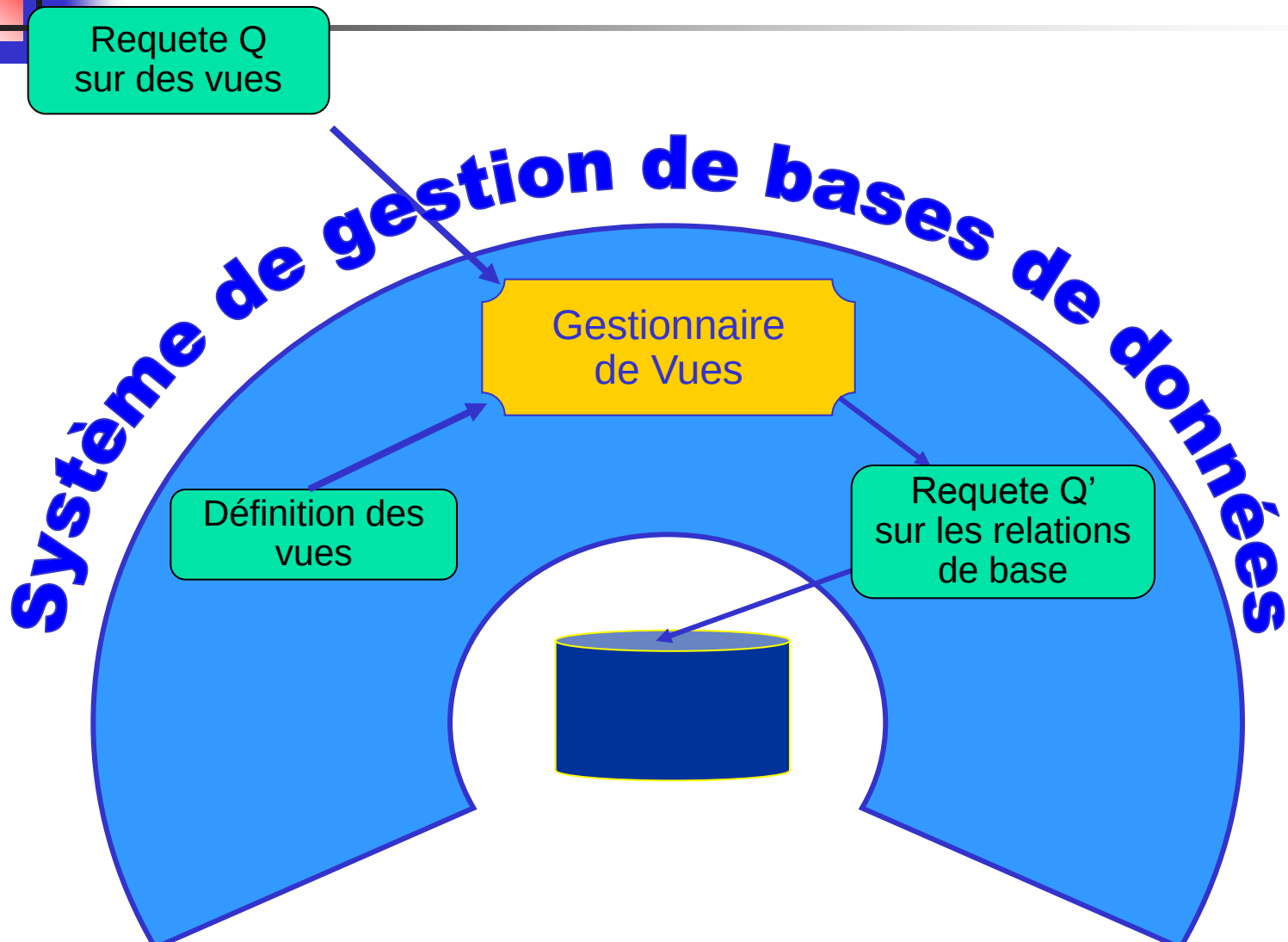
- Les Vues relationelles
- Privilèges
  - Rôles
- Déclencheurs (Triggers)

4. Gestion des vues

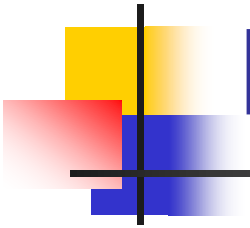
- Les vues permettent d'implémenter l'indépendance logique en permettant de créer des objets virtuels.
- Vue= Requête SQL stockée
- Le SGBD stocke la définition et non le résultat.
- Le SGBD transforme les requêtes sur les vues en requêtes sur les relations de base
- Exemple: La vue des patients toulousain

```
CREATE VIEW TOULOUSAIN AS (
        SELECT nom, prenom
        FROM Patients
        WHERE ville = 'Toulouse');
```

Gestion des vues



L'objet Vue



- Une vue est une table virtuelle
  - Aucune implémentation physique des données
- Sa définition est enregistrée dans le Dictionnaire de Données
- A chaque appel d'une vue:
  - Le SGBD réactive sa construction à partir du dictionnaire de données
- Vue mono-table
  - Créée à partir d'une table
- Vue multi-tables
  - Créée par une jointure en forme relationnelle
- Attention aux modifications sur les vues

Utilisation d'une vue

- Simplification de requêtes pour les non spécialistes
- Création de résultats intermédiaires pour des requêtes complexes
- Présentation différente de la base de données
  - Schéma externe
- Mise en place de la confidentialité
- Restructuration d'une base de données

Création et suppression d'une vue

- Création d'une vue

CREATE [OR REPLACE] VIEW <nom_vue> [liste de colonnes de la vue]
AS
SELECT …
[WITH CHECK OPTION [CONSTRAINT <nom_contrainte>]

- Suppression d'une vue

DROP VIEW <nom_vue>

Exemple Vues mono-table (1)

- Vue mono-table avec restriction horizontale

```
CREATE VIEW enseignant_info AS
SELECT * FROM Enseignant
WHERE idDip in
                (Select IdDip FROM Diplôme
                  Where UPPER(nomDip) LIKE '%INFO%');
```

- Vue mono-table avec restriction verticale

```
CREATE VIEW etudiant_scol AS
SELECT idEtu, nomEtu, adrEtu, idDip
    FRIOM Etudiant;
```

Exemple Vues mono-table (2)

- Vue mono table avec restriction mixte

```
CREATE VIEW etudiant_info (numEtudiant, nomEtdiant,
    adrEtudiant, dip) AS
    Select idEtu, nomEtu, adrEtu, idDIp
    From Etudiant
    Where idDip in
            (SELECT idDip FROM Diplôme
              Where Where UPPER(nomDip) LIKE '%INFO%');
```

- Vue mono table avec colonne virtuelles

```
CREATE VIEW employe_salaire (ne, nome, mensuel, annuel,
    journalier) AS
            SELECT idEmp, nomEmp, sal, sal*12, sal/12
            FROM Employe;
```

Exemple Vues mono-table (3)

- Vue mono table avec groupage

```
CREATE VIEW emp_service (ns, nombreEmp, moy_sal)  AS
        Select idService, COUNT(*), AVG(salaire)
        FROM Employe
        GROUP BY idService;
```

- Utilisation de la vue → reconstruction

```
SELECT * FROM emp_service WHERE nombreEmp >5 ;
```

Création d'une vue des services comportant plus de 5 employés

```
Select idService As ns, Count(*) AS nombreEmp, AVG(salaire) AS moy_sal
FROM Employe
GROUP BY ns
HAVING COUNT(*) >5;
```

Création de vues multi-tables (1)

- Simplification de requêtes
- Tables temporaires virtuelles de travail
- Transformation de la présentation des données
  - Schéma externe

```
CREATE VIEW emp_service (nom_service,nom_employe) AS
  SELECT s.nomS, e.nomE
    FROM Employe e, Services s
    Where e.idSer=s.idSer;
```

Création de vues multi-tables (2)

- Reconstruction des clients (UNION)

```
Create View Clients (idCli,nom,secteur) AS
Select ct.*,'T' FROM clients_toulouse ct
UNION
Select cb.*,'B' FROM clients_bordeaux cb
UNION
Select cp.*,'P' FROM clients_paris cp;
```

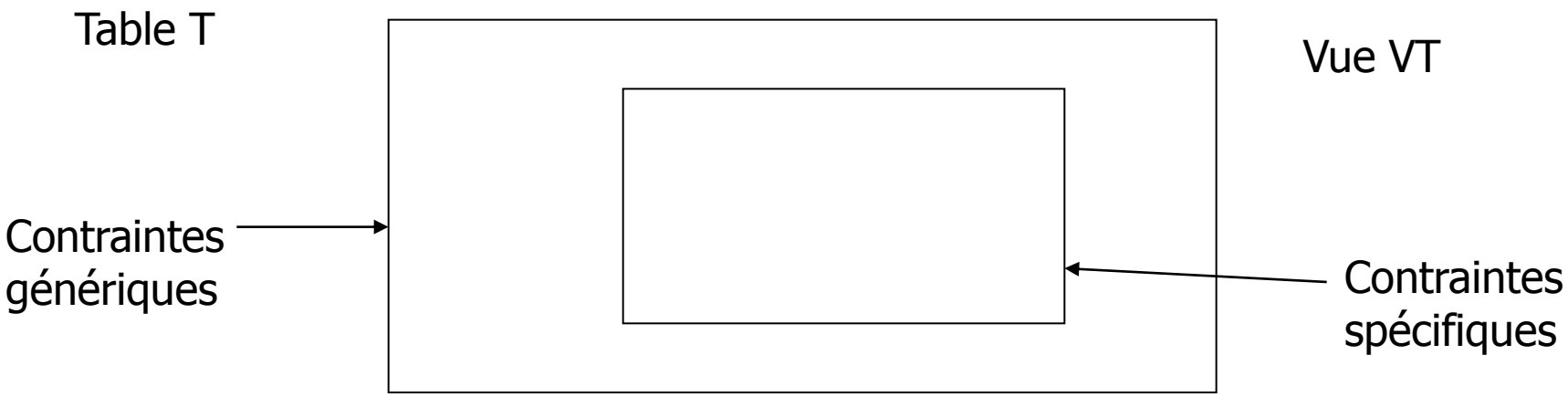
- Reconstruction des étudiants (JOINTURE)

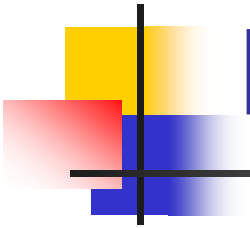
```
Create View etudiants (idEtu,nom,adresse,nomstage,entrstage) AS
        Select e.id,e.nom, e.adr, s.nomS, s.entrS
                FROM Etudiant e, Stage s
        WHERE e.id=s.id;
```

Vues avec contraintes WITH CHECK OPTION

- Principe de prédicat de sélection des lignes se transforme en contrainte
- Mise en place de contraintes spécifiques
- Permet de restreindre les insertions et mises à jour des données « à travers » la vue aux données faisant partie de la sélection de la vue
- CREATE [OR REPLACE] VIEW nom_vue [(nv_nom_col)*]
  AS subquery [WITH CHECK OPTION [CONSTRAINT nom_contrainte]]
  [WITH READ ONLY];

Vues avec contraintes WITH CHECK OPTION





Mise à jour d'une vue

Une vue ne peur être mise à jour (INSERT, UPDATE, DELETE) que si elle obéit à un certain nombre de conditions :

- ne porter que sur une table (pas de jointure)
- ne pas contenir de dédoublonnage (pas de mot clef DISTINCT) si la table n'a pas de clef
- contenir la clef de la table si la table en a une
- ne pas transformer les données (pas de concaténation, addition de colonne, calcul d'agrégat...)
- ne pas contenir de clause GROUP BY ou HAVING
- ne pas contenir de sous requête
- répondre au filtre WHERE si la clause WITH CHECK OPTIONS est spécifié lors de la création de la vue

Exemple de vue avec contrainte

```
CREATE OR REPLACE VIEW Petit_pays
    AS SELECT * FROM Pays
        WHERE surface < 100
         WITH CHECK OPTION CONSTRAINT CK_Petit_pays;
```

INSERT INTO Petit_pays values ("Chine", "Pékin", 2000, 500)

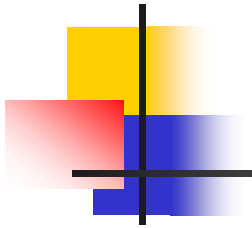
- **Impossible!!!** Car surface est égale à 500

Exemple de vue avec contrainte

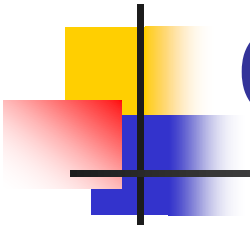
- interdit toute mise à jour en utilisant le nom de la vue dans un ordre insert, update ou delete.
- CREATE OR REPLACE VIEW Petit_pays
  AS SELECT * FROM Pays WHERE surface < 100
  WITH READ ONLY;

- INSERT INTO Petit_pays values ("Chine", "Pékin", 2000, 500):
- **Impossible !!! Pas de mise à jour de la table autorisée à travers la vue**

Le langage de contrôle de données



- GRANT et REVOKE
- BEGIN et END TRANSACTION
- COMMIT et ROLLBACK



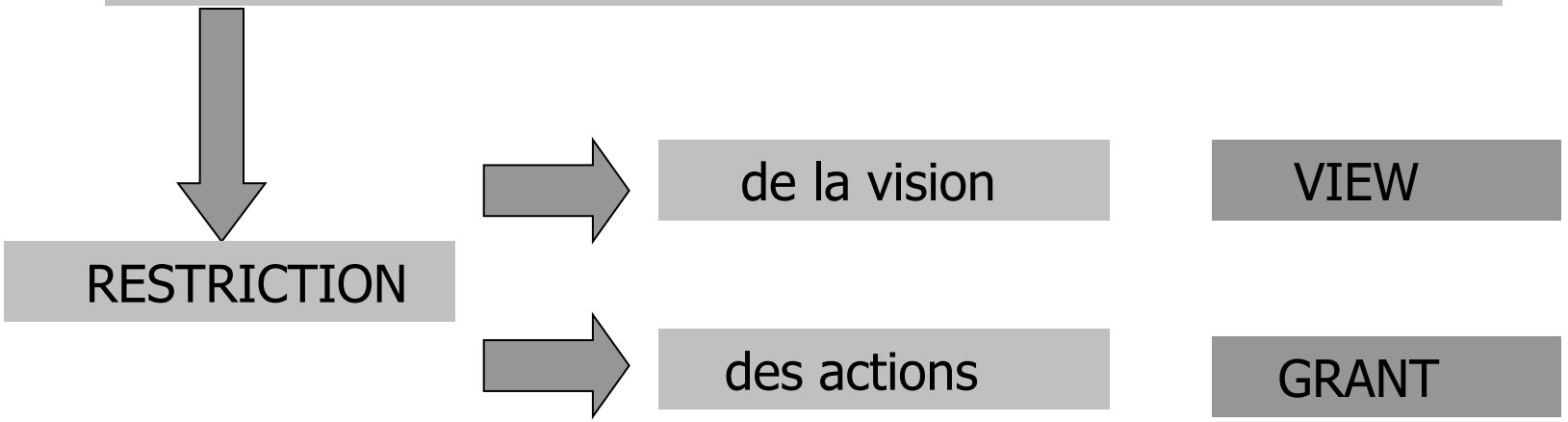
Contrôle des données

Il s'agit de définir des permissions au niveau des utilisateurs d'une base de données. On parle de DCL (Data Control Language)

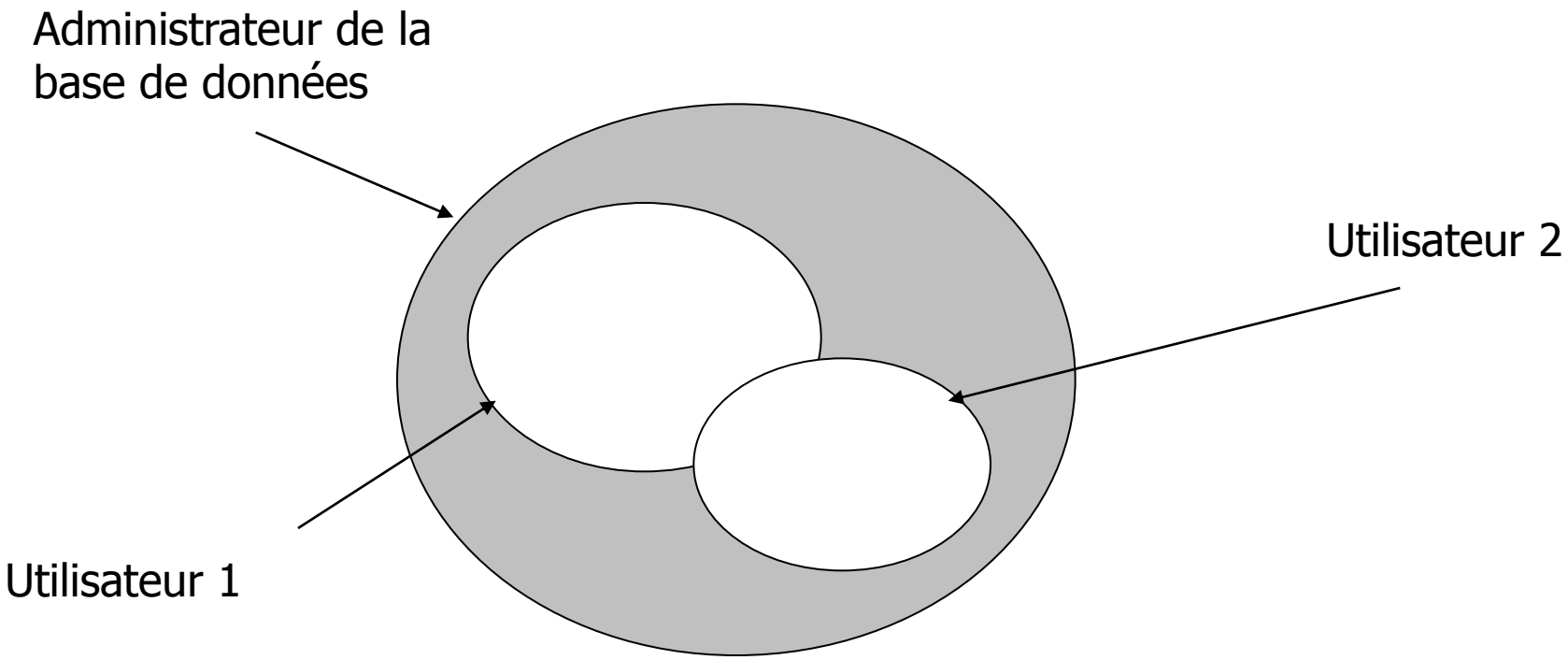
- Notion de sous schémas
  - Restriction de la vision
  - Restriction des actions
- Privilèges
  - Systèmes
  - Objets
- Contraintes événementielles: triggers
  - Contrôle avant une modification
  - Mises à jour automatiques

Restreindre les accès à une BD

Tous le monde ne peut pas voir et faire n'importe quoi



Restriction des accès
Sous schéma ou schéma externe



FIN

Restriction des actions: les privilèges

- Contrôler l'accès à la base de données
- Sécurité système
  - Couvre l'accès à la BD et son utilisation au niveau du système
    - Nom de l'utilisateur et mot de passe, espace disque alloué aux utilisateurs et opérations systèmes autorisées par l'utilisateur
- Sécurité données
  - Couvre l'accès aux objets de la base de données et leur utilisation, ainsi que les actions exécutées sur ces objets par les utilisateurs

Exemples de privilèges systèmes

Objets	Opération
SYSTEM	ALTER, AUDIT
SESSION	CREATE, ALTER, RESTRICTED
TABLESPACE	CREATE, ALTER, MANAGE, DROP, UNLIMITED
USER	CREATE, DECOME, ALTER, DROP
ROLLBACK SEGMENT	CREATE, ALTER, DROP
TABLE	CREATE
ANY TABLE	CREATE, ALTER, BACKUP, DROP, LOCK, COMMENT, SELECT, INSERT, UPDATE, DELETE
CLUSTER	CREATE
ANY CLUSTER	CREATE, ALTER, DROP
PROCEDURE	CREATE
ANY PROCEDURE	CREATE, ALTER, EXECUTE, DROP
TRIGGER	CREATE
ANY TRIGGER	CREATE, ALTER, DROP
PROFILE	CREATE, ALTER, DROP
ANY INDEX	CREATE, ALTER, DROP
.....	

Délégation de privilèges systèmes

Délégation: **GRANT**

GRANT Privilège / rôle TO Utilisateur / Rôle / PUBLIC [WITH ADMIN OPTION] ;	**With admin option** donne le droit de redistribuer les privilèges reçus. PUBLIC représente tous les utilisateurs

EXEMPLE :

```
GRANT ALTER ANY TABLE TO SCOTT WITH ADMIN OPTION;
GRANT CREATE USER, ALTER USER, DROP USER TO SCOTT;
```

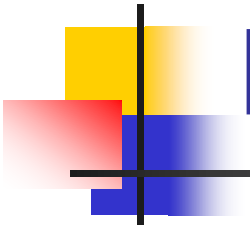
Suppression de privilèges systèmes

Suppression: **REVOKE**

```
REVOKE Privilège/rôle
FROM utilisateur / rôle /  PUBLIC  ;
```

EXEMPLE :

```
REVOKE ALTER USER, DROP USER FROM SCOTT;
```



Privilèges Objet

- Contrôles les actions sur les objets
  - objets;: tables, vues…
  - Actions: update, alter, delete, select, insert, execute…
- Le propriétaire (owner) peut donner des privilèges sur ses propres objets
- Les privilèges peuvent être donnés avec l'option de délégation

Privilèges Objet

Object Privilege	Tables	Views	Sequences	Procedure	Snapshots
ALTER	√		√		
DELETE	√	√			
EXECUTE				√	
INDEX	√				
INSERT	√	√			
REFERENCES	√				
SELECT	√	√	√		√
UPDATE	√	√			

Délégation de privilèges objet

GRANT privilège_objet ON objet TO PUBLIC / UTILISATEUR / ROLE [WITH GRANT OPTION] ;] ;	☞ **With grant option** donne le droit de redistribuer les privilèges reçus. ☞ Pour les privilèges Insert, Update et References il est possible de préciser le nom de la colonne.

EXEMPLE : Depuis le compte **SCOTT** où l'on trouve ces différents objets

GRANT SELECT ON EMP TO SYSTEM;

GRANT ALL ON DEPT TO SYSTEM;

⇨ L'utilisateur SYSTEM peut tout faire sur la table DEPT

GRANT INSERT, UPDATE (adr, tel) ON etud_info

To Martine, Nicole;

Suppression de privilèges objet

```
REVOKE privilège_objet ON objet
 FROM   PUBLIC / UTILISATEUR / ROLE;
```

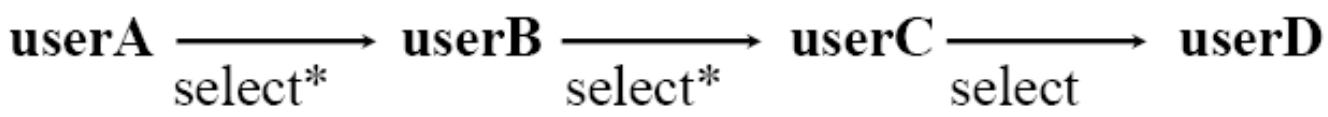
EXEMPLE **:**

```
REVOKE ALL ON EMP FROM SCOTT;
```

```
REVOKE UPDATE (tel) On etud_info
        FROM Nicole;
```

Suppression en cascade

table T :



* : abréviation pour WITH GRANT OPTION

userA exécute : revoke select on T from userB
→ suppression du privilège pour userB, userC et userD

Mot réservé : PUBLIC

- PUBLIC est un mot réservé pour désigner l'ensemble des utilisateurs de la base de données.
- Donner des droits a PUBLIC, correspond a donner les droits a tous les utilisateurs de la base.
  - Ainsi que les futurs utilisateurs !
- Exemple

```
GRANT SELECT ON inscrit TO PUBLIC;
```

Les rôles

- Un rôle est un ensemble nommé de privilèges qui peut être attribué à de utilisateurs et/ou à d'autres rôles.
- Les rôles facilitent la gestion des privilèges d'accès aux données.
- Les rôles ne font pas partie du schéma.

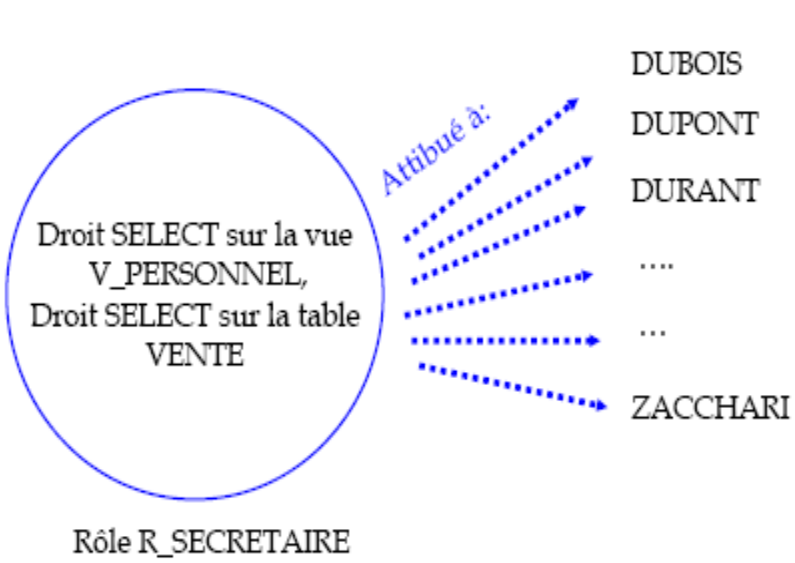
Regroupement de privilèges pour des familles d'utilisateurs

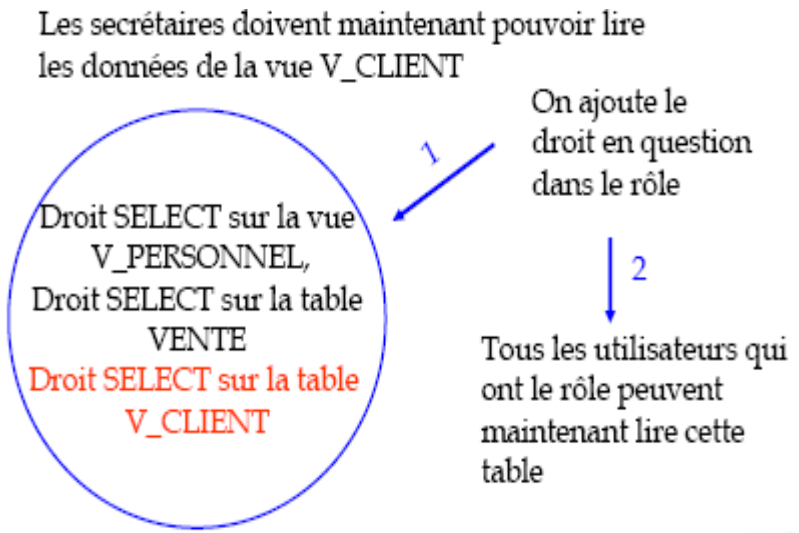
Evitent la multiplication des GRANT

Un utilisateur peut posséder plusieurs roles mais n'est connecté qu'avec un seul à la fois

On peut donner un mot de passe pour certains roles

Les rôles


Droit SELECT sur la vue V_PERSONNEL,
Droit SELECT sur la table VENTE
Attribué à:
DUBOIS
DUPONT
DURANT
....
...
ZACCHARI
Rôle R_SECRETAIRE


Les secrétaires doivent maintenant pouvoir lire les données de la vue V_CLIENT
Droit SELECT sur la vue V_PERSONNEL,
Droit SELECT sur la table VENTE
Droit SELECT sur la table V_CLIENT
1
On ajoute le droit en question dans le rôle
2
Tous les utilisateurs qui ont le rôle peuvent maintenant lire cette table

Manipulation des rôles: ordres

Création, modification d'un rôle

[CREATE |ALTER] ROLE <nom_role> {BOT IDENTIFIED |IDENTIFIED {BY mot de passe |EXTERNALLY}};

Remplissage et attribution d'un role

GRANT {privilège1 |role1} TO <nom_role>;

GRANT ROLE <nom_role> TO user;

Rôle par défaut ou activation

SET ROLE <nom_role> [IDENTIFIED BY <mot de passe>];

Suppression /révocation d'un role

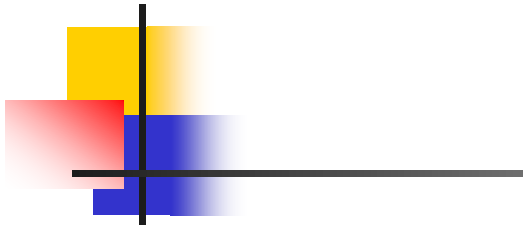
DROP <nom_role>; REVOKE ROLE <nom_role> FROM user;

Exemples de manipulation de rôles

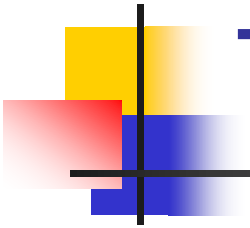
- CREATE ROLE secretariat info;
- GRANT SELECT, UPDATE(adr,tel) On ens_info TO secretariat_info;
- GRANT secretariat_info TO laurent, caroline;

Rôles prédéfinis

- DBA
  - Tous les privilèges systèmes et objets
- RESOURCE
  - Création de tous les objets classiques
  - Propriétaire des données (owner)
- CONNECT
  - Connexion à la base
  - Attente des privilèges objets
- EXP_FULL_DATABASE
  - Exportation de tous les objets
- IMP_FULL_DATABASE
  - Importation d'objets.



Les Déclencheurs (Triggers)



Trigger (Introduction)

Base de Données Active

- réagit aux changements d 'état de la base de données

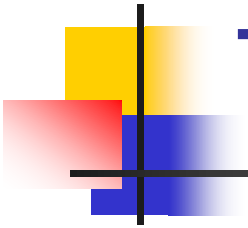
Déclencheur = Evénement-Condition-Action

- Evénement dans la base
- Condition
- Déclenchement d 'une action

Trigger (Introduction)

Les déclencheurs (triggers) introduisent un aspect dynamique dans la base de données car ils lancent l'exécution d'un module de programme automatiquement quand quelque chose survient.

Un déclencheur est donc une procédure stockée dans la base de données et exécutée (ou déclenchée) en réponse à des activités particulières comme un INSERT, UPDATE ou DELETE à effectuer sur une table.

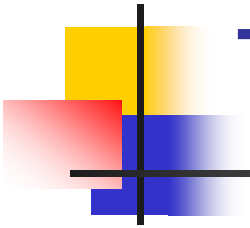


Trigger (Introduction)

- Traitement implicite déclenché par un événement
- Utilisé pour implémenter des règles de gestion complexes et pour étendre les règles d'intégrité référentiel associées à table lors de leur création de cette dernière
- Un déclencheur peut être associé soit à un outil client, soit à une table
- Un trigger est défini au moyen de PL/SQL.

Trigger (Introduction)

- Le traitement est exprimé en PL/SQL. Il peut lui-même faire appel à des procédures et des fonctions écrites en PL ou Java.
- Son code est stocké dans la base de données (développement plus rapide)
- Un déclencheur peut être actif ou non
- Si un déclencheur aboutit, la transaction qui l'a appelé peut se poursuivre
- Le déclenchement peut se propager en cascade. (tout en respectant le principe d'atomicité d'une transaction)



Trigger (Introduction)

- Utilisés pour spécifier des contraintes plus complexes tels que:
  - Contraintes portant sur plusieurs tables
  - Contraintes nécessitant l'utilisation de requêtes
- Offrent une technique procédurale
  - Procédures à exécuter programmés en PL/SQL

Structure d'un trigger

- CREATE [OR REPLACE] TRIGGER nom-trigger
  BEFORE | AFTER
  INSERT OR UPDATE [OF column] OR DELETE ON nom-table
  [FOR EACH ROW [WHEN (condition)]]
  bloc d'instructions pl/sql

  - BEFORE | AFTER
    - spécifie le point d'execution de la procérure comme avant ou après l'éxecution de l'événement (NB qui pourrait être annulée par le trigger)
  - INSERT OR UPDATE [OF column] OR DELETE ON nom-table
    - spécifie les événements qui feront déclencher le trigger (c'est à dire ceux qui pourranient violer la contrainte d'integrité que le trigger doit implémenter)
  - [FOR EACH ROW [WHEN (condition)]]
    - spécifie le type de trigger comme étant un trigger de ligne (si omis c'est un trigger d'instruction) et des conditions optionnelles pour son déclenchement
  - bloc d'instructions pl/sql
    - spécifie les actions à executer, programmées en pl/sql

Le nom d'un Trigger

doit être unique dans un même schéma

peut être le nom d'un autre objet (table, vue, procédure) mais à éviter

Triggers (Creation, suppression,…)

Un déclencheur peut être créé et remplacé (avec CREATE OR REPLACE TRIGGER), supprimé (avec DROP TRIGGER), activé ou désactivé (avec ALTER TRIGGER et les directives ENABLE ou DISABLE) habituellement dans le contexte d'un outil interactif (SQL*Plus, par exemple).

Dans la définition d'un déclencheur, il faut préciser l'énoncé déclencheur, le moment d'exécution (BEFORE, AFTER, INSTEAD OF), le mode d'exécution (FOR EACH ROW), la condition d'exécution (WHEN), etc. ainsi que le traitement à effectuer (constituant le corps sous forme d'un bloc PL/SQL).

Réalisation d'un Trigger

Le corps peut être exécuté avant (BEFORE), après (AFTER) ou à la place de (INSTEAD OF) l'énoncé déclencheur.

Il peut être exécuté une fois pour chaque ligne (avec FOR EACH ROW) de table affectée par l'énoncé déclencheur ou une seule fois pour toutes les lignes affectées par ce dernier (sans FOR EACH ROW).

On distingue donc les déclencheurs de type ligne (avec FOR EACH ROW) des déclencheurs de type énoncé (sans FOR EACH ROW).

Le traitement associé à une ligne affectée peut nécessiter les versions ancienne et nouvelle (désignées par OLD et NEW) de celle-ci.

L’option Before/ After

les triggers AFTER row sont plus efficaces que les BEFORE row parce qu'ils ne nécessitent pas une double lecture des données.

Fonctionnement d'un Triggers

Le traitement associé à toutes les lignes affectées peut nécessiter les versions ancienne et nouvelle (désignées par OLD_TABLE et NEW_TABLE) de celles-ci.

:NEW et :OLD doivent être utilisés à l'intérieur du corps du déclencheur.

NEW et OLD doivent être utilisés dans la condition d'une clause WHEN et l'option REFERENCING NEW AS nom_1 OLD AS nom_2 (pour résoudre un conflit de nom).

Les mêmes règles s'appliquent aussi aux deux symboles OLD_TABLE et NEW_TABLE.

Il est possible de regrouper la programmation de différents déclencheurs associés à la même table.

Fonctionnement d'un Triggers

- ✓ La nouvelle valeur est appelée
  :new.colonne
- ✓ L'ancienne valeur est appelée
  :old.colonne
- ✓ *Exemple :* IF :new.salaire < :old.salaire .....
- ✓ Si un trigger ligne BEFORE modifie la nouvelle valeur d'une colonne, un éventuel trigger ligne AFTER déclenché par la même instruction voit le changement effectué par le trigger BEFORE.

Fonctionnement d'un Triggers

La définition du trigger précise la table associée au trigger :

- ✓ une et une seule table
- ✓ pas une vue.

Triggers sur les vues

L'option INSTEAD OF ne peut être utilisée que dans les déclencheurs créés sur une vue.

Les options BEFORE et AFTER ne peuvent pas être utilisées dans les déclencheurs créés sur une vue.

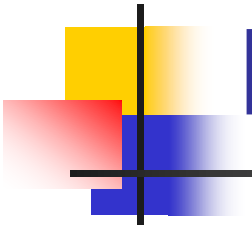
Le corps d'un déclencheur INSTEAD OF doit assurer le renforcement de la contrainte de l'option CHECK déclarée dans le CREATE VIEW associé.

Applications de déclencheurs : audit évolué, mise en œuvre de règles d'affaires, autorisation de sécurité, intégrité référentielle, validation de données et de transactions, calcul de valeurs dérivées, mise à jour de vues complexes, suivi et journaux d'opérations, etc.

Exemple (Trigger)

```
1 CREATE OR REPLACE TRIGGER myFirstTrigger
2                                   BEFORE UPDATE OF ename ON emp
3   FOR EACH ROW
4     BEGIN
5     DBMS_OUTPUT.ENABLE(20000);
6     DBMS_OUTPUT.PUT_LINE(:NEW.ENAME|| ` ` ||:OLD.ENAME);
7     EXCEPTION
8     WHEN OTHERS THEN       DBMS_OUTPUT.PUT_LINE(SQLERRM);
9    END;
```

```
SQL> UPDATE emp
       SET ename = 'Durand'
     WHERE empno = 7839;
SQL>
```

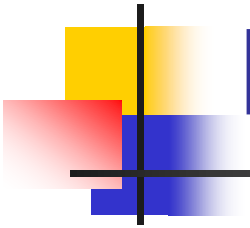


Résultats d'un trigger

Le trigger se termine favorablement s'il ne soulève pas d'exceptions ou ne viole pas d'autres contraintes (contraintes déclarées ou check).

Dans ce cas, les actions effectuées par l'événement et le trigger sont acceptées (commit implicite). Sinon toutes les actions du trigger sont annulées (rollback implicite).

Résultats d'un Trigger



- Un Trigger peut répondre à plusieurs événements. Dans ce cas, il est possible d'utiliser les prédicats intégrés **INSERTING**, **UPDATING** ou **DELETING** pour exécuter une séquence particulière du traitement en fonction du type d'événement (voir slide suivant).
- NB: un déclencheur peut s'appliquer au niveau de la base (démarrage d'instance par exemple). Ce point n'est pas traité ici.

Les prédicats conditionnels INSERTING,DELETING et UPDATING

- ✓ Quand un trigger comporte plusieurs instructions de déclenchement (par exemple INSERT OR DELETE OR UPDATE), on peut utiliser des prédicats conditionnels (INSERTING, DELETING et UPDATING) pour exécuter des blocs de code spécifiques pour chaque instruction de déclenchement.
- ✓ *Exemple :*

```
CREATE TRIGGER ...
BEFORE INSERT OR UPDATE ON employe
.....
BEGIN
.....
IF INSERTING THEN ..... END IF;
IF UPDATING THEN ..... END IF;
.....
END;
```

Exemple

UPDATING peut être suivi d'un nom de colonne :

```
CREATE TRIGGER ...
BEFORE UPDATE OF salaire, commission ON employe
.......
BEGIN
......
IF UPDATING ('salaire') THEN ..... END IF;
......
END;
```

Exemple (Trigger)

```
CREATE OR REPLACE TRIGGER myFirstTrigger
                                    AFTER UPDATE OR INSERT ON emp
        FOR EACH ROW
        BEGIN
    DBMS_OUTPUT.ENABLE(20000);
                IF INSERTING THEN  DBMS_OUTPUT.PUT_LINE(' INSERT ');
                END IF;
                IF UPDATING('ENAME') THEN
                  DBMS_OUTPUT.PUT_LINE(' UPDATED ' || :NEW.ENAME);
                END IF;
        END;
/
SQL> UPDATE emp SET ename = 'TOTO' WHERE empno = 7839;
UPDATED TOTO
1 row updated.
```

Mise hors-service d'un trigger

- ALTER TRIGGER myTrigger DISABLE;

- ALTER TABLE maTable
- DISABLE ALL TRIGGERS;

- ALTER TRIGGER tonTrigger ENABLE
- ALTER TRIGGER nosTrigger
- ENABLE ALL TRIGGERS;

- DROP TRIGGER ceTrigger;

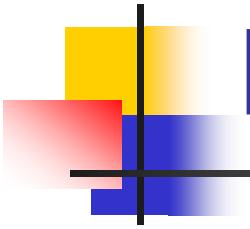
USER_TRIGGERS

- SQL> desc user_triggers

```
Name                                      Null?    Type
----------------------------------------- -------- ----------------
TRIGGER_NAME                                       VARCHAR2(30)
TRIGGER_TYPE                                       VARCHAR2(16)
TRIGGERING_EVENT                                   VARCHAR2(227)
TABLE_OWNER                                        VARCHAR2(30)
BASE_OBJECT_TYPE                                   VARCHAR2(16)
TABLE_NAME                                         VARCHAR2(30)
COLUMN_NAME                                        VARCHAR2(4000)
REFERENCING_NAMES                                  VARCHAR2(128)
WHEN_CLAUSE                                        VARCHAR2(4000)
STATUS                                             VARCHAR2(8)
DESCRIPTION                                        VARCHAR2(4000)
ACTION_TYPE                                        VARCHAR2(11)
TRIGGER_BODY                                       LONG
```

Proprietés de triggers

- SUR UNE MÊME TABLE ON NE PEUT AVOIR 2 TRIGGERS DIFFERENTS AYANT LES MÊMES (TYPE, CONDITION DE DECLENCHEMENT, TIMING)
- 12 TRIGGERS DIFFERENTS AU MAXIMUM SUR UNE TABLE
- PAS DE CONTROLE DE TRANSACTION COMMIT ROLLBACK INTERDITS



Exemple

```
CREATE OR REPLACE TRIGGER Emp_count
AFTER DELETE ON Emp_tab FOR EACH ROW
DECLARE
n INTEGER;
 BEGIN
 SELECT COUNT(*) INTO n FROM Emp_tab;
DBMS_OUTPUT.PUT_LINE(' There are now ' || n || '
employees.');
 END;
```

```
DELETE FROM Emp_tab WHERE Empno = 7499;
```

Nombre de Triggers par table

- ✓ On peut avoir au maximum un trigger de chacun des types suivants pour chaque table :
  - *BEFORE UPDATE row*
  - *BEFORE DELETE row*
  - *BEFORE INSERT statement*
  - *BEFORE INSERT row*
  - *BEFORE UPDATE statement*
  - *BEFORE DELETE statement*
  - *AFTER UPDATE row*
  - *AFTER DELETE row*
  - *AFTER INSERT statement*
  - *AFTER INSERT row*
  - *AFTER UPDATE statement*
  - *AFTER DELETE statement.*
- ✓ Même pour UPDATE, on ne peut pas en avoir plusieurs avec des noms de colonnes différents.

Instructions autorisées

- ✓ les instructions du LMD sont autorisées
- ✓ les instructions du LDD ne sont pas autorisées
- ✓ les instructions de contrôle de transaction (ROLLBACK, COMMIT) ne sont pas autorisées.

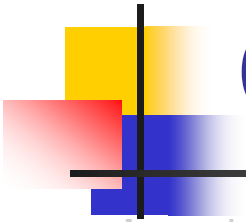
Ordre de traitement

Ordre de traitement des lignes

- ✓ On ne peut pas gérer l'ordre des lignes traitées par une instruction SQL.
- ✓ On ne peut donc pas créer un trigger qui dépende de l'ordre dans lequel les lignes sont traitées.

Triggers en cascade

- ✓ Un trigger peut provoquer le déclenchement d'un autre trigger.
- ✓ ORACLE autorise jusqu'à 32 triggers en cascade à un moment donné.



Conditions

Conditions nécessaires pour créer un trigger

- ✓ il faut avoir le privilège CREATE TRIGGER
- ✓ il faut soit posséder la table sur laquelle on veut définir un trigger, soit posséder le privilège ALTER sur la table sur laquelle on veut définir le trigger, soit posséder le privilège ALTER ANY TABLE

Modification de triggers

- ✓ Pour modifier un trigger, on refait une instruction CREATE TRIGGER suivie de OR REPLACE ou bien on supprime le trigger (DROP TRIGGER nomtrigger) et on le crée à nouveau.

Activation d'un trigger

- ✓ Un trigger peut être activé ou désactivé.
- ✓ S'il est désactivé, ORACLE le stocke mais l'ignore.
- ✓ On peut désactiver un trigger si :
  - ❊ il référence un objet non disponible
  - ❊ on veut charger rapidement un volume de données important ou recharger des données déjà contrôlées.
- ✓ Par défaut, un trigger est activé dès sa création.
- ✓ Pour désactiver un trigger, on utilise l'instruction *ALTER TRIGGER avec l'option DISABLE :*

  ALTER TRIGGER nomtrigger DISABLE;
- ✓ On peut désactiver tous les triggers associés à une table avec la commande :

  ALTER TABLE nomtable DISABLE ALL TRIGGERS;
- ✓ A l'inverse on peut réactiver un trigger :

  ALTER TRIGGER nomtrigger ENABLE;

  ou tous les triggers associés à une table :

  ALTER TABLE nomtable ENABLE ALL TRIGGERS;

Informations sur les triggers

Recherche d'information sur les triggers

- ✓ Les définitions des triggers sont stockées dans les tables de la métabase, notamment dans les tables USER_TRIGGERS, ALL_TRIGGERS et DBA_TRIGGERS

La procédure

raise_application_error (error_number,error_message)

- ✓ error_number doit être un entier compris entre -20000 et -20999
- ✓ error_message doit être une chaîne de 500 caractères maximum.
- ✓ Quand cette procédure est appelée, elle termine le trigger, défait la transaction (ROLLBACK), renvoie un numéro d'erreur défini par l'utilisateur et un message à l'application.

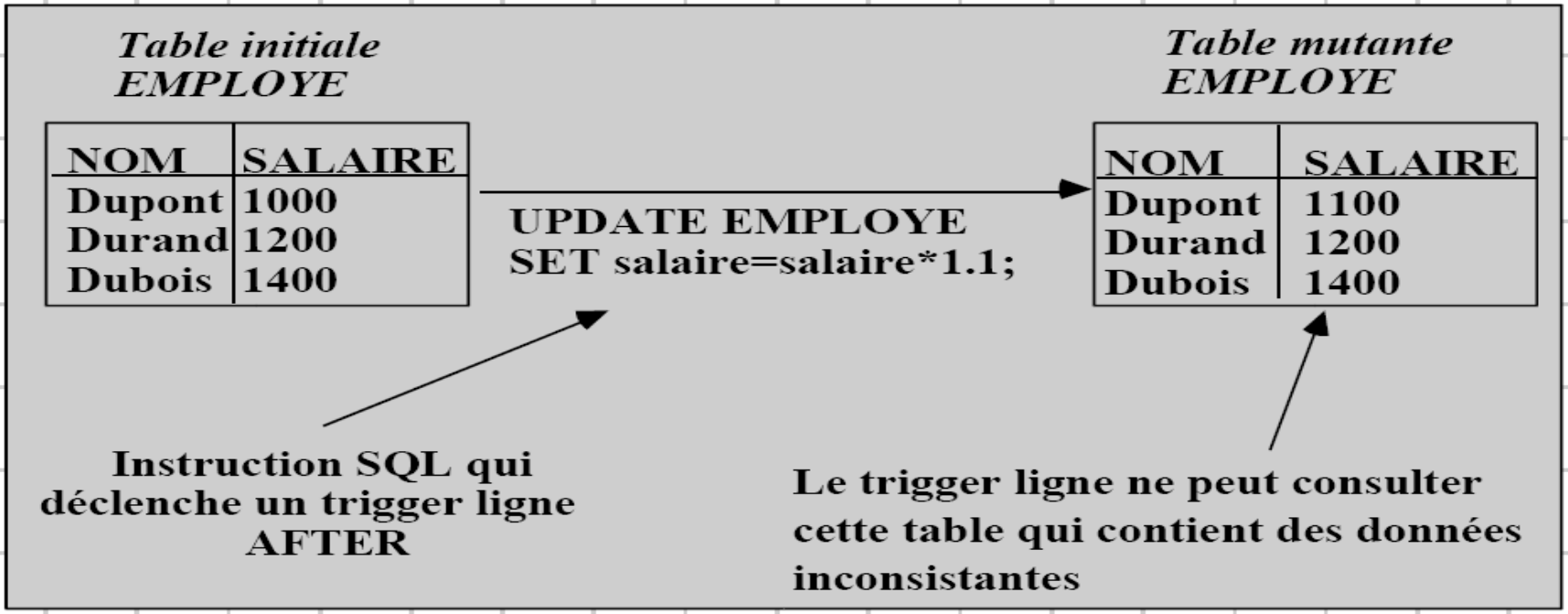
Gestion des exceptions

Gestion d'exceptions

- ✓ Si une erreur se produit pendant l'exécution d'un trigger, toutes les mises à jour produites par le trigger ainsi que par l'instruction qui l'a déclenché sont défaites.
- ✓ On peut introduire des exceptions en provoquant des erreurs.
  - Une exception est une erreur générée dans une procédure PL/SQL.
  - Elle peut être prédéfinie ou définie par l'utilisateur.
  - Un bloc PL/SQL peut contenir un bloc EXCEPTION gérant les différentes erreurs possibles avec des clauses WHEN.
  - Une clause WHEN OTHERS THEN ROLLBACK; gère le cas des erreurs non prévues.

Limites

- ✓ **Un trigger ligne ne peut pas lire et/ou modifier la table concernée (appelée table mutante) par l'instruction (INSERT, UPDATE ou DELETE) qui a déclenché ce trigger. *Exemple :***



Conseils

Utiliser des triggers pour garantir que, lorsqu'une opération spécifique est effectuée, les actions liées ont aussi réalisées.

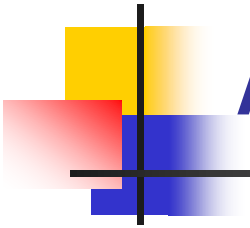
N'utiliser les triggers que pour les opérations globales, centralisées qui doivent être déclenchées indifféremment de l'utilisateur ou de l'application.

N'utiliser des triggers que s'il n'y a pas d'autre possibilité intégrée dans Oracle (par exemple des contraintes déclaratives définies avec la table).

Ne pas créer des triggers récursifs :

- ✓ *par exemple, un trigger AFTER UPDATE sur une table qui ferait un UPDATE sur la même table.*

Limiter la taille des triggers : à la première utilisation, le trigger est compilé (préférer une procédure stockée, déjà compilée).



A suivre

- Quelques exemples

FIN