Chapitre 2 : Contraintes d'intégrité complexes et déclencheurs (triggers)

Objectif

- Rôle :
  - Faire un système fiable et performant
  - Assurer l'intégrité des données
- Notions de base
  - Contrainte d'intégrité :
    - assertion qui doit toujours être vérifiée par les données de la base.
  - Base de données cohérente :
    - dont l'ensemble des contraintes d'intégrité est vérifié

Contraintes d'intégrité

Plusieurs formes

- Déjà vues en S1 et S2 :
  - Définition du domaine ou du type d'un attribut (Integer, Char, ...)
  - Condition sur les valeurs des attributs d'un n-uplet (clause CHECK de SQL),
  - Définition de la clé primaire et des clés étrangères d'une relation (Primary Key, ...)
- Contraintes d'intégrité complexes
  - Assertions générales,
    - Une **assertion** est une formule logique qui doit être vraie quelle que soit l'extension de la BD.
  - Déclencheurs (Triggers)
    - Un **déclencheur** est une règle, dite **active**, de la forme :
      - « événement - condition - action »
      - L'action est déclenchée à la suite de l'événement, si la condition est vérifiée.
      - Une action peut être une vérification ou une mise à jour.

Contraintes d'intégrité

- Types SQL
  - INTEGER
  - CHAR
- ...
- Primary Key, foreign key .. References...
- NOT NULL
- Unique
- CHECK

Modification de contraintes

- Ajout de contraintes

```
ALTER TABLE <nom_table>
   ADD CONSTRAINT nom_contrainte type_contrainte;
```

- Suppression de contraintes

```
ALTER TABLE <nom_table>
   DROP CONSTRAINT nom_contrainte;
```

Modification de contraintes

```
ALTER TABLE Employes
    MODIFY IdEmploye CONSTRAINT NN_employes NOT NULL;
```

Pour les contraintes de colonne

```
ALTER TABLE Employes
    ADD CONSTRAINT CHK_salaire CHECK (salaire>3000);
```

Pour les contraintes de table

Activation/désactivation de contraintes

- Désactivation de contraintes

```
ALTER TABLE <nom_table>
    DISABLE CONSTRAINT nom_contrainte;
```

  - Les contraintes existent toujours dans le dictionnaire de données mais ne sont pas actives
  - Chargement de données volumineuses extérieures à la base

- Activation de contraintes

```
ALTER TABLE <nom_table>
    ENABLE CONSTRAINT nom_contrainte;
```

Activation/désactivation: Clause d'exception

```
ALTER TABLE <nom_table>
    ENABLE CONSTRAINT nom_contrainte
    EXCEPTIONS into <nom_table_rejet>;
```

- Structure de la table de rejet:

```
CREATE TABLE <nom_table_rejet>
(ligne ROWID,
Owner VARCHAR2(30),
Table_name VARCHAR2(30),
Constraint VARCHAR2(30));
```

- Pour sélectionner les tuples ayant transgressés la contrainte

```
SELECT <nom_table>.*
FROM <nom_table>, <nom_table_rejet>
Where <nom_table>.ROWID = <nom_table_rejet>.ligne;
```

Où le champ rowid de <nom_table> est un pointeur vers l'emplacement de chacune des lignes

Activation/désactivation: Contraintes avec contrôle différé

- Une contrainte peut être différée (déclenchée lors du commit)
  - NOT DEFERRABLE (par défaut)
  - DEFFERABLE
- Comportement par défaut
  - INITIALLY IMMEDIATE (par défaut)
  - INITIALLY DEFERRED
- ALTER TABLE Employe

Add CONSTRAINT FK_emp_ser FOREIGN KEY (idService) REFERENCES Services(IdService) deferrable)

Contraintes avec contrôle différé : exemple

```
ALTER TABLE Employe
ADD CONSTRAINT fk_emp_serv
REFERENCES IdService
DEFERRABLE INITIALLY
IMMEDIATE;
```

```
SQL> INSERT INTO
*Employe VALUES (1, 'MB',
2)
ERROR at line 1:
ORA-02291: integrity
constraint
(MB.FK_EMP_DEPT_ID)
violated - parent key not
found ;

*
```

```
ALTER TABLE Employe
ADD CONSTRAINT fk_emp_serv
REFERENCES IdService
DEFERRABLE INITIALLY
DEFFERED;
```

```
SQL> INSERT INTO Emp VALUES
(1, 'MV', 2);
1 row created.
```

```
SQL> COMMIT;
COMMIT
*
ERROR at line 1:
ORA-02091: transaction rolled back
ORA-02291: integrity constraint
(MICHEL.FK_EMP_DEPT_ID)
violated - parent key not found
```

Contraintes à contrôle différé

```
SET {CONSTRAINT I  CONSTRAINTS }
   {nomcontrainte1, nomcontrainte2, … IALL }
   {DEFERRED I IMMEDIATE};
```

```
SET CONSTRAINTS ALL DEFERRED;
```

Exemple

- Soit la BD contenant les deux relations suivantes :
  - employé(nom_emp, nom_dept, salaire)
  - département(nom_dept, directeur, nb_emp)
  - La contrainte d'intégrité :
    - « Tout employé du département '*Recherche*' doit avoir un salaire supérieur à 3000 € »,
  - Comment mettre en place cette contrainte d'intégrité ?

Assertion

- Une **assertion** est une formule logique qui doit être vraie quelle que soit l'extension de la BD.

- CREATE ASSERTION <name>
  CHECK (<condition>);

Exemple

- Retour sur l'exemple :
  - employé(nom_emp, nom_dept, salaire)
  - département(nom_dept, directeur, nb_emp)
  - La contrainte d'intégrité : « Tout employé du département 'Recherche' doit avoir un salaire supérieur à 3000 € », s'exprime par l'assertion suivante :

```
CREATE ASSERTION CHECK NOT EXISTS
(SELECT *
FROM employe
WHERE salaire <= 3000 AND
nom_dept =
(SELECT nom_dept
FROM departement
WHERE nom_dept = 'Recherche')))
```

- Concept de bases de données actives
- Programme évènementiel
  - Bloc évènement
    - UPDATE, DELETE, INSERT
  - Bloc action
    - Bloc PL/SQL
- Trois fonctions assurées
  - Mise en place de contraintes complexes
  - Mise à jour de colonnes dérivées
  - Génération d'évènements
- **Associé à une table**
  - Suppression de la table -> suppression des triggers

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

12 type de triggers

- « Row» Trigger ou « Statement » Trigger
  - Row: le trigger est exécuté pour chaque ligne touchée
  - Statement: le trigger est exécuté une fois

- Exécution avant ou après l'évènement
  - Before: le bloc action est levé avant que l'évènement soit exécuté
  - After: le bloc action est levé après l'exécution du bloc évènement

- Trois évènements possibles
  - UPDATE: certaines colonnes
  - INSERT
  - DELETE

Syntaxe de création

```
CREATE [OR REPLACE] TRIGGER <nom_trigger>
{BEFORE | AFTER}
{INSERT | DELETE | UPDATE [ OF colonnes]}
ON <nom_table>
[FOR EACH ROW]
[When condition ]
[DECLARE]
-- déclaration de variables, exceptions,
-- curseurs
BEGIN
-- bloc action
-- ordres SQL et PL/SQL
END;
/
```

Déclencheurs

Syntaxe

- INSERT OR UPDATE OR DELETE ON nom-table
  - Précise le ou les événements provoquant l'exécution du trigger
- BEFORE | AFTER
  - Spécifie l'exécution de la procédure avant(BEFORE) ou après (AFTER) l'exécution de l'événement déclencheur
  - BEFORE : vérification de l'insertion
  - AFTER : copie ou archivage des opérations
- [FOR EACH ROW]
  - Indique que le trigger est exécuté pour chaque ligne modifiée/insérée/supprimée
  - Si cette clause est omise, le trigger est exécuté une seul fois pour chaque commande UPDATE, INSERT, DELETE, quelque soit le nombre de lignes modifiées, insérées ou supprimées.
- [WHEN (condition)]
  - conditions optionnelles permettant de restreindre le déclenchement du trigger
  - !!! Ne peut contenir de requêtes

- Pour les row trigger (triggers lignes)

(et seulement eux!!!)

Accès aux valeurs des colonnes pour chaque ligne modifiée

- Deux variables: : NEW.colonne et OLD.colonne

	Ancienne valeur :OLD.colonne	Nouvelle valeur :NEW.colonne
INSERT	NULL	Nouvelle valeur
DELETE	Ancienne valeur	NULL
UPDATE	Ancienne valeur	Nouvelle valeur

Possibilité d'utiliser d'autres noms
(clause REFERENCING
NEW AS nouveauNom
OLD AS ancienNom)

Bloc d'instruction

- Bloc PL/SQL standard
  - DECLARE
    - Déclaration de variables et constantes avec leur type
  - BEGIN
    - Bloc d'instructionsPL/SQL
  - END
- Le bloc d'instructions PL/SQL peut contenir:
  - des blocs spécifiant des actions différentes fonction de l'événement déclencheur
    - IF INSERTING THEN bloc d'instructions pl/sql END IF
    - IF UPDATING THEN bloc d'instructions pl/sql END IF
    - IF DELETING THEN bloc d'instructions pl/sql END IF
  - des Instructions SQL
    - SELECT, INSERT, UPDATE, DELETE, ... Mais pas de COMMIT et ROLLBACK
  - 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)
- Faire appel à des procédures et fonctions PL/SQL

Trigger de contraintes: lever une erreur

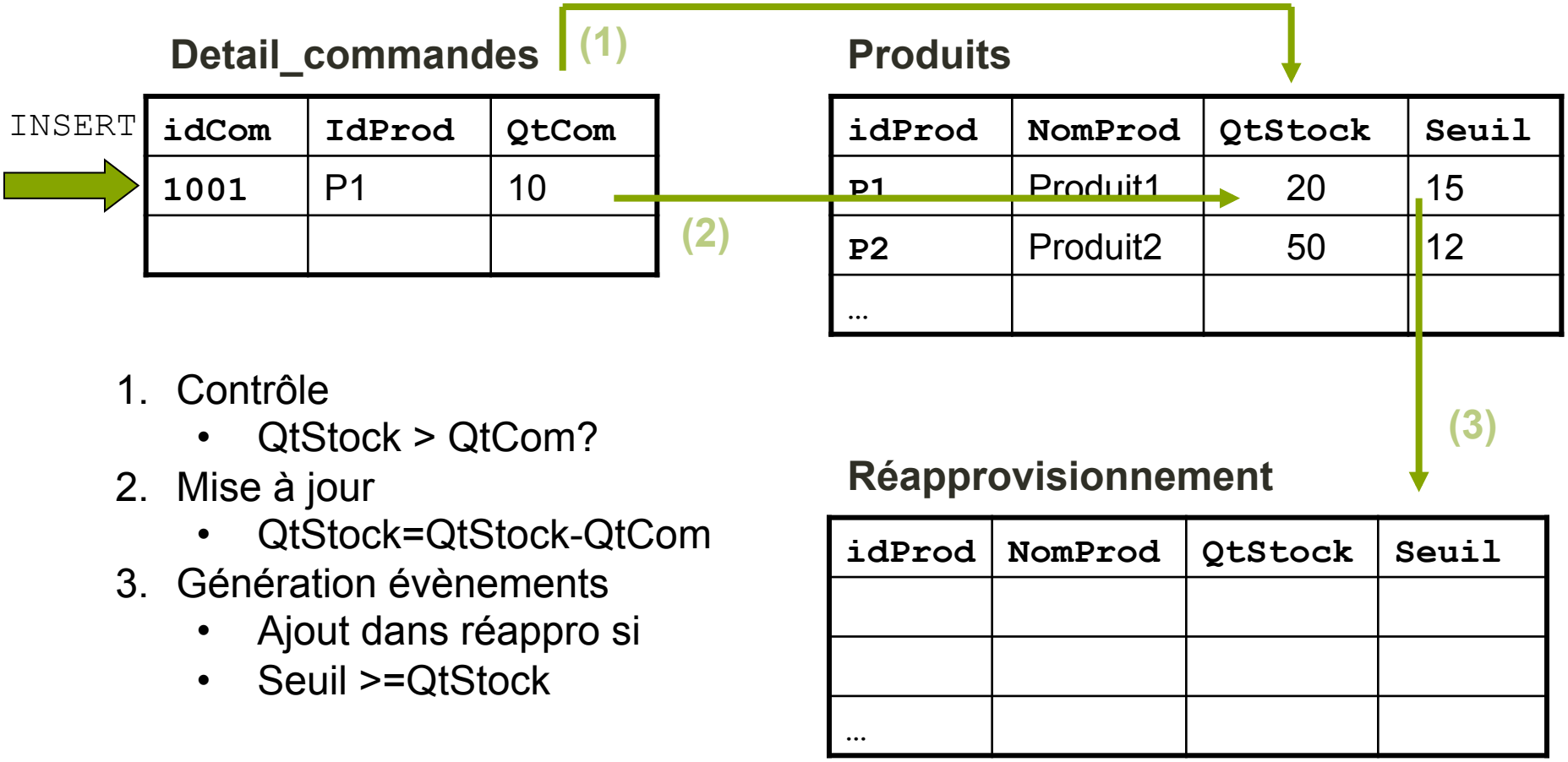
- Ordre: RAISE_APPLICATION_ERROR

```
RAISE_APPLICATION_ERROR(n°erreur, 'texte erreur')
```

- N° erreur utilisateur:[-20000, -20999] → SQLCODE
- Texte erreur: message envoyé → SQLERRM

- Utilisation obligatoire de RAISE_APPLICATION_ERROR si on veut empêcher une insertion, une suppression ou une mise à jour
  - Utilisation directe
  - Utilisation lors de la récupération d'une exception

Exemple de row trigger: prise de commande



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

Déclencheurs

Prise de commande:(1) contrôle quantité en stock ?

```
CREATE 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;
/
```

Prise de commande: (2) mise à jour quantité en stock

```
CREATE 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;
/
```

Prise de commande: (3) génération d'un réapprovisionnement

```
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;
/
```

Prédicats dans un trigger

```
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 ... END IF;
IF DELETING THEN ... END IF;
IF INSERTING THEN ... END IF;
END;
/
```

- Raisonnement sur la globalité de la table et non sur un enregistrement particulier
- TRIGGER BEFORE
  - 1 action avant un ordre UPDATE de plusieurs lignes
- TRIGGER AFTER
  - 1 action après un ordre UPDATE de plusieurs lignes

Exemple de Statement Trigger

```
CREATE TRIGGER contrôle_date_emp
BEFORE UPDATE ON emprunt
BEGIN
IF TO_CHAR(SYSDATE, 'DY') IN ('SAT', 'SUN') THEN
RAISE_APPLICATION_ERROR (-20102, 'Desole les emprunts sont interdits le
We...');
END IF;
END;
/
```

- Trigger faisant le travail 'à la place de' ...
- Posé sur une vue multitables pour autoriser les modifications sur des objets virtuels (car mise à jour impossibles sur des vues multitables)
- Principalement utilisé dans les bases de données réparties pour permettre les modifications sur les objets virtuels fragmentés

- Vue étudiant résultat de 4 tables : Etudiant_licence, Etudiant_Master, Etudiant_doctorat, Stage

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

Exemple de Trigger « Instread of »utilisation de la vue : INSERT

- 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 ?

- Solution créer un trigger ‘INSTEAD of’

Exemple de Trigger « Instread of »utilisation de la vue : INSERT

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

- Lors du lancement d'une requête SQL, oracle
  - Execute tous les before statement triggers declenchés par la requête
  - Pour chaque ligne affectée par la requête (each row)
    - Exécute tous les before row triggers déclenchée par la requête
    - Vérouille la ligne, effectue la maj et vérifie les contraintes d'intégrité
    - Exécute tous les after row triggers déclenchés par la requête
  - Vérifie les contraintes d'intégrité différées
  - Execute toues les after statement triggers déclenchés par la requête

- Suppression d'un trigger

```
DROP TRIGGER <nom_trigger>;
```

- Désactivation d'un trigger

```
ALTER TRIGGER <nom_trigger> DISABLE;
```

- Réactivation d'un trigger

```
ALTER TRIGGER <nom_trigger> ENABLE;
```

- Tous les triggers d'une table

```
ALTER TABLE <nom_table>
{ENABLE|DISABLE} ALL TRIGGERS;
```

- USER_TRIGGERS

```
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
```

- USER_TRIGGER_COLS

TRIGGER_OWNER	VARCHAR2(30)
TRIGGER_NAME	VARCHAR2(30)
TABLE_OWNER	VARCHAR2(30)
TABLE_NAME	VARCHAR2(30)
COLUMN_NAME	VARCHAR2(4000)
COLUMN_LIST	VARCHAR2(3)
COLUMN_USAGE	VARCHAR2(17)

IdService	NomService	Etage	NbEmployés
1	Comptabilité	2	1
2	RH	1	1
3	Informatique	3	2

IdEmploye	NomEmploye	Salaire	DateEmbauche	#IdService
18	Martin	30000	01/02/1999	2
12	Cartaux	25500	30/03/2000	3
15	Jean	45000	01/01/2001	1
23	Parot	23000	23/06/2001	3

Exemple de row trigger: prise de commande

INSERT

Detail_commandes

idCom	IdProd	QtCom
1001	P1	10
1002	P1	30

Produits

idProd	NomProd	QtStock	Seuil
P1	Produit1	20	15
P2	Produit2	50	12
…			

Réapprovisionnement

idProd	NomProd	QtStock	Seuil
…			