UFR 919 Ingénierie – module 3I009
Cours 10 : SQL : contraintes/triggers/vues

- Contraintes d'intégrité (rappel)
- Triggers (rappel)
- Vues

Contraintes d'intégrité

Une *contrainte d'intégrité* est une *condition logique* qui doit être satisfaite par les données stockées dans la BD.

But : maintenir la cohérence/l'intégrité de la BD :

- Vérifier/valider *automatiquement* (en dehors de l'application) les données lors des mises-à-jour (insertion, modification, effacement)
- La cohérence est liée à la notion de transaction

cohérent (Début trans.) → Non visible de l'extérieur de la transaction → cohérent (Fin trans.)

- Déclencher *automatiquement* des mises-à-jour entre tables pour maintenir la cohérence globale.

Contraintes d'attributs

PRIMARY KEY
- désigne un *ensemble d'attributs* comme la *clé primaire* de la table

FOREIGN KEY
- désigne un *ensemble d'attributs* comme la *clé étrangère* dans une *contrainte d'intégrité référentielle*

NOT NULL
- spécifie qu'un attribut ne peut avoir de valeurs nulles

UNIQUE
- spécifie un *ensemble d'attributs* dont les valeurs doivent être distinctes pour chaque couple de n-uplets.

D'un point de vue logique, pas de différence entre PK et Unique, mais la relation peut être organisée (Arbre-B+) selon la clé primaire en utilisant la directive *organization index* en fin de *create table*.

Exemple : Clés

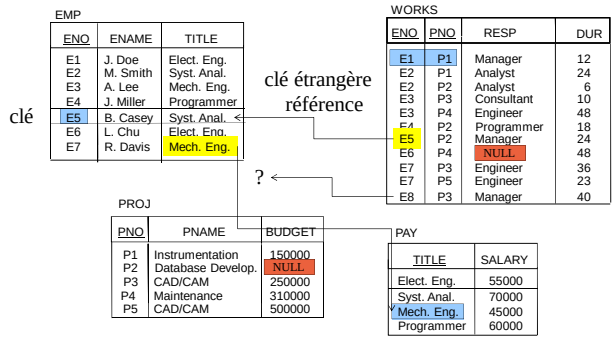
Créer la table Project(Pno, Pname, Budget, City) :

```
CREATE TABLE Project
       (Pno   CHAR(3),
        Pname VARCHAR(20)UNIQUE NOT NULL,
        BudgetDECIMAL(10,2)DEFAULT 0.00,
        City CHAR(9),
        PRIMARY KEY (Pno));
```

ou Pno **CHAR(3) PRIMARY KEY**

Remarque: plusieurs clés possibles (Pno et Pname), 1 seul est primaire
PRIMARY KEY implique UNIQUE et **NOT NULL.**

Exemple clé étrangère



Maintenance automatique de l'intégrité référentielle

La suppression (ON DELETE) ou la mise-à-jour (ON UPDATE) d'un n-uplet référencé (de clé primaire) nécessite une action sur le n-uplet avec la clé étrangère :

- RESTRICT : l'opération est *rejetée* (par défaut)
- CASCADE : supprime ou modifie tous les n-uplets avec la clé étrangère si le n-uplet référencé est supprimé ou *sa clé* est modifiée
- SET [NULL | DEFAULT] : mettre à NULL ou à la valeur par défaut quand le n-uplet référencé est effacé/sa clé est modifiée. L'attribut doit accepter la valeur NULL

Pas de *on update cascade* en Oracle => trigger

Contraintes de n-uplets

Contraintes portant ***sur une seule table***.

(d'autres tables peuvent apparaître dans des sous-requêtes)

La condition est vérifiée *chaque fois* qu'un **n-uplet** est inséré ou modifié dans la table; la mise-à jour (transaction) est rejetée si la condition est fausse.

```
CREATE TABLE Works
  (Eno      CHAR(3),
   Pno      CHAR(3),
   Resp     CHAR(15),
   Dur      INT,
PRIMARY KEY (Eno,Pno),
FOREIGN KEY (Eno) REFERENCES Emp(Eno),
FOREIGN KEY (Pno) REFERENCES Project(Pno),
CHECK (NOT(PNO<'P5') OR Dur>18));
```


Contraintes de tables : Assertions

Contraintes ***globales sur plusieurs relations***.

```
CREATE ASSERTION name CHECK (condition)
```

Exemple: le salaire total des employés du projet P5 ne peut dépasser 500

```
CREATE ASSERTION salary-control CHECK
  (500 >= (SELECT    SUM(Salary)
           FROM      Emp E, Pay P, Works W
           WHERE     W.Pno = 'P5'
           AND       W.Eno = E.Eno
           AND       E.Title = P.Title));
```

Remarque : pas disponible dans les SGBD actuels => triggers
Remarque : vérification des contraintes = problème d'efficacité

SQL : contraintes

- Contraintes d'intégrité
- Triggers
- vues

Triggers : utilisation

« Règles actives » (ECA) *généralisant* les contraintes d'intégrité **:**

- génération automatique de valeurs manquantes (ex. valeur dérivée, par défaut)
- éviter des modifications invalides (C: test, A: abort)
- implantation de règles applicatives (« business rules »)
- génération de traces d'exécution, statistiques, ...
- maintenance de répliques
- propagation de mises-à-jour sur des vues vers les tables
- intégrité référentielle entre des données distribuées
- interception d'événements utilisateur / système (LOGIN, STARTUP, …)

Trigger ou règle active ou règle ECA

Définition ECA :

Événement (E) :

- une mise-à-jour de la BD qui active le trigger, ou d'autres (opérations DDL, logon, serverror, ..)

ex.: réservation de place

Condition (C):

- un test ou une requête devant être vérifié lorsque le trigger est activé (une requêtes est *vraie* si sa réponse n'est pas vide)

ex.: nombre de places disponibles ?

Action (A):

- une procédure exécutée lorsque le trigger est activé et la condition est *vraie* : E,C → A

ex.: annulation de réservation

Exécution des triggers (1)

Moment de déclenchement du trigger par rapport à l'événement *E* (maj. activante) :

- avant (before) E
- après (after) *E*
- à la place de (instead of) de *E* (spécifique aux vues => màj des données de la base)

Nombre d'exécutions de l'action A par déclenchement :

- une exécution de l'action *A* par n-uplet modifié (ROW TRIGGER)
- une exécution de l'action *A* par événement (STATEMENT TRIGGER)

Exécution des triggers (2)

Delta structure : « les données considérées par le trigger »

- *:old* avant l'événement, *:new* après l'événement (peuvent être renommés)
- *for each row* : un n-uplet, *for each statement*: un ensemble de n-uplets
- *:new* peut être modifié par l'action, mais effet seulement si *before*
- Pour agir avec un trigger *after*, il faut modifier directement la base
- *:old* (resp. *:new*) n'a pas de sens pour *insert* (resp. *delete*)

Syntaxe (Oracle)

```
{ CREATE | REPLACE } TRIGGER <nom>
// Événement
{BEFORE | AFTER | INSTEAD OF}
{INSERT | DELETE | UPDATE [OF <attribut, …>]}
    ON <table>
[REFERENCING [NEW AS <nouv>] [OLD AS <anc>]]
[FOR EACH ROW] // ROW TRIGGER
// Condition
[WHEN  (<condition SQL>) THEN]
// Action
<Procédure SQL>
```


Contrôle d'intégrité

Emp (Eno, Ename, Title, City)

Vérification de la contrainte de clé à l'insertion d'un nouvel employé :

```
CREATE TRIGGER InsertEmp
BEFORE INSERT ON Emp
REFERENCING NEW AS N
FOR EACH ROW
WHEN EXISTS
     (SELECT * FROM Emp WHERE Eno=N.Eno)
THEN
    ABORT;
```


Contrôle d'intégrité

Emp (Eno, Ename, Title, City) **Pay**(Title, Salary)

Suppression d'un titre et des employés correspondants (« ON DELETE CASCADE ») :

```
CREATE TRIGGER DeleteTitle
BEFORE DELETE ON Pay
REFERENCING OLD AS O
FOR EACH ROW
BEGIN
    DELETE FROM Emp WHERE Title=O.Title
END;
```

Mise-à-jour automatique

Emp (Eno, Ename, Title, City)

Création automatique d'une valeur de clé (autoincrément) :

```
CREATE TRIGGER SetEmpKey
BEFORE INSERT ON Emp
REFERENCING NEW AS N
FOR EACH ROW
BEGIN
  N.Eno := SELECT COUNT(*) FROM Emp
END;

/* le premier Eno sera 0 */
```


Mise-à-jour automatique

Pay(Title, Salary, Raise)

Maintenance des augmentations (raise) de salaire :

```
CREATE TRIGGER UpdateRaise
AFTER UPDATE OF Salary ON Pay
REFERENCING OLD AS O, NEW AS N
FOR EACH ROW
BEGIN
     UPDATE Pay
     SET Raise = N.Salary - O.Salary
     WHERE Title = N.Title;
END
```


Analyse des triggers

Plusieurs triggers de type différent peuvent être affectés au même événement :

- Ordre par défaut : BEFORE STATEMENT → BEFORE ROW → AFTER ROW → AFTER STATEMENT

Un trigger activé peut en activer un autre :

- longues chaînes d'activation => problème de performances
- boucles d'activation => problème de terminaison

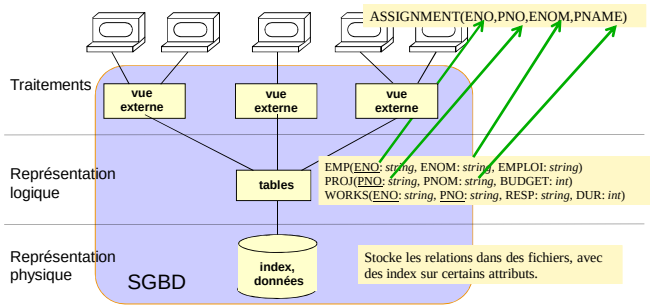
Recommandations :

- pour l'intégrité, utiliser si possible le mécanisme des contraintes plus facile à optimiser par le système.
- associer les triggers à des règles de gestion.

SQL : contraintes

- Contraintes d'intégrité
- Triggers
- Vues

Architecture ANSI/SPARC



Pourquoi définir des vues

Une BD peut contenir des *centaines de tables avec des milliers d'attributs* :

1.Les requêtes sont complexes :
- difficiles à formuler
- Ne portent que sur un sous-ensemble des attributs
- source d'erreurs

2.Une modification du schéma nécessite la modification de beaucoup de programmes.

Solution : *Adapter* le schéma et les données à des applications spécifiques → vues

Définition d'une vue

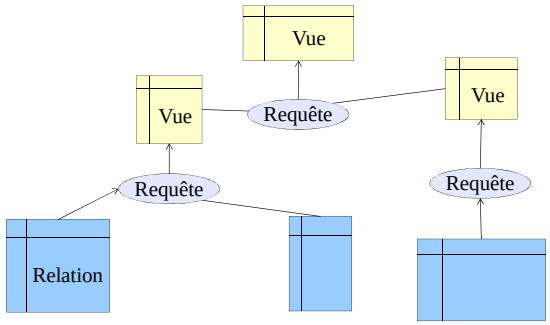
Définition : Une vue **V(a1, a2, … an)** est une relation avec n attributs qui contient le résultat d'une requête Q(x1, x2, …xn) évaluée sur une base de données BD :

V(a1, a2, …an) :- Q(x1, x2, …,xn, BD)

Remarques :

- V possède un schéma relationnel avec des attributs a1,…an.
- V *reflète* l'état actuel d'une base de données BD
- V peut être interrogée et il est possible de définir des vues à partir d'autres vues.
- On distingue les relations « matérialisées » (tables) et les relations « virtuelles » (vues)

Relations et Vues



Définition d'une vue dans SQL

CREATE VIEW *nom_vue* [(*att1*, *att2*...)]
AS *requête_SQL* [**WITH CHECK OPTION**]

- *nom_vue* désigne le nom de la relation
- *att1*, ... (optionnel) permet de nommer les attributs de la vue (attributs de la requête par défaut)
- *requête_SQL* désigne une requête SQL standard qui définit le « contenu » (instance) de la vue
- **WITH CHECK OPTION** (voir mises-à-jour de vues)

Exemple

Emp (Eno, Ename, Title, City) **Project**(Pno, Pname, Budget, City)
Pay(Title, Salary) **Works**(Eno, Pno, Resp, Dur)

Définition de la vue **EmpProjetsParis** des employés travaillant dans des projets à Paris :

```
CREATE VIEW EmpProjetsParis(NumE, NomE, NumP, NomP, Dur)
AS SELECT Emp.Eno, Ename, Works.Pno, Pname, Dur
   FROM Emp, Works, Project
   WHERE Emp.Eno=Works.Eno
   AND Works.Pno = Project.Pno
   AND Project.City = 'Paris'
```


Interrogation de vues

Emp (Eno, Ename, Title, City) **Project**(Pno, Pname, Budget, City)
Pay(Title, Salary) **Works**(Eno, Pno, Resp, Dur)

Les noms des employés de projets Parisiens :

Requête sans vue:

```
SELECT Ename
FROM Emp, Works, Project
WHERE Emp.Eno=Works.Eno
AND Works.Pno = Project.Pno
AND Project.City = 'Paris'
```

Requête avec vue:

```
SELECT NomE
FROM EmpProjetsParis
```

On obtient le même résultat

Évaluation de requêtes sur des vues

Vue :

```
CREATE VIEW EmpProjetsParis
AS SELECT Emp.Eno, Ename, Project.Pno,
Pname, Dur
FROM Emp, Works, Project
WHERE Emp.Eno=Works.Eno
AND Works.Pno = Project.Pno
AND Project.City = 'Paris'
```

Requête :

```
SELECT Emp.Eno FROM EmpProjetsParis
WHERE Dur > 3
```

Mise-à-jour de vues

Problème de mise-à-jour : une vue est une relation virtuelle et toutes les modifications de cette relation doivent être "transmises" aux relations (tables) utilisées dans sa définition.

La plupart du temps il n'est pas possible de mettre à jour une vue (insérer un n-uplet, …).

Exemple:

CREATE VIEW V AS SELECT A,C
FROM R,S WHERE R.B = S.B

- Insertion d'un n-uplet [A:1,C:3] dans la vue V
- Quelle est la modification à faire dans R et S (valeur de B) ?

Vues modifiables (dépend beaucoup du sgbd)

Une vue *n'est pas modifiable* en général:

- quand elle ne contient pas tous les attributs définis comme NON NULL dans la table interrogée (cas de l'insert)
- quand elle contient une jointure
- quand elle contient une fonction agrégat

Règle : Une vue *est modifiable* quand elle est définie comme une sélection/projection sur une relation R (qui peut aussi être une vue modifiable) sans utilisation de SELECT DISTINCT ni de group by.

Lorsqu'une vue n'est pas modifiable, on peut créer un trigger *instead of*, qui va exécuter les répercussions sur les tables de la base de la mise à jour sur la vue au lieu de le faire sur la vue (attention, dans un trigger *instead of*, on ne peut pas préciser l'attribut d'un update, donc *update on* et non pas *update of Att on*)

Mises-à-jour

Emp (Eno, Ename, Title, City) **Project**(Pno, Pname, Budget, City)
Pay(Title, Salary) **Works**(Eno, Pno, Resp, Dur)

```
CREATE VIEW ProjetParis
AS SELECT Pno,Pname,Budget
      FROM Project
      WHERE City='Paris';

UPDATE ProjetParis
SET Budget = Budget*1.2;
```


WITH CHECK OPTION

WITH CHECK OPTION protège contre les « disparitions de n-uplets » causées par des mise-à-jour :

```
CREATE VIEW ProjetParis
WITH CHECK OPTION
AS SELECT Pno,Pname,Budget,City
      FROM Project
WHERE City='Paris';
```

```
UPDATE ProjetParis
SET City = 'Lyon'
WHERE Pno=142;
```

← Mise-à-jour rejetée

Vues et tables

Similitudes :
- Interrogation SQL
- UPDATE, INSERT et DELETE sur vues modifiables
- Autorisations d'accès
- Evaluation et optimisation

Différences:
- On ne peut pas créer des index sur les vues
- On ne peut pas définir des contraintes (clés)
- Une vue est recalculée à chaque fois qu'on l'interroge
- *Vue matérialisée* : stocker temporairement la *vue* pour améliorer les performances. => pb de performance si les tables sont mises à jour
  - rafraichissement des vues incrémental : détecter quelles vues matérialisées doivent être rafraichies après une mise à jour