Intégrité sémantique dans les bases de données relationnelles

Bernard ESPINASSE
Professeur à Aix-Marseille Université (AMU)
Ecole Polytechnique Universitaire de Marseille

Janvier 2018

- **Intégrité sémantique et contraintes d'intégrités**
- **Prise en compte en SQL des CI inhérentes au modèle relationnel**
- **Prise en compte en SQL de CI spécifiques et Triggers**

Plan

1. Intégrité sémantique

- Intégrité dans une base de données
- Intégrité sémantique
- Types de contraintes d'intégrité
- Contrôles sur les contraintes d'intégrité

Intégrité dans les Bases de données : définition

intégrité d'une base de données : = concordance des données avec le monde réel que la base de données modélise

Divers aspects liés à l'intégrité d'une base de données :

- **intégrité sémantique * ←**
- **contrôle de concurrence ***
- **protection des données**
- **sécurité des données**

1 - Intégrité sémantique (1)

- **intégrité sémantique d'une BdD:**
  - concerne la **qualité** de l'information stockée dans la BdD,
  - consiste à s'assurer les informations stockées sont **cohérentes** par rapport à la **signification** qu'elles ont, par le respect de **règles de cohérence**
- **différents niveaux de règles de cohérence :**

Domaine d'application	**Règle de gestion (business rule)**
Modèle conceptuel (Merise, UML, …)	**Règle d'intégrité**
Base de données	**Contraintes d'intégrité**
Mécanisme de support	**Procédures stockées , index, trigger, écran ou applet de validation, …**

Intégrité sémantique (2)

- **contrainte d'intégrité (C.I.) :**

  = **assertion** qui doit être **vérifiée** par des données à des **instants déterminés.**

  Ex : la note de l'étudiant est comprise entre 0 et 20 ; le salaire du professeur ne peut jamais diminuer, …

- **base de données (sémantiquement) INTEGRE :**

  = lorsque l'ensemble des **contraintes d'intégrité** (implicites et explicites) est **respecté** par **toutes** les **données** de la base

Types de contraintes d'intégrité (1)

Contrainte		Individuelle		Ensembliste	
		statique	dynamique	statique	dynamique
Intra-table	Mono-attribut				
	Multi-attributs				
Inter-table	Multi-attributs				

Type de contraintes d'intégrité (2)

intra-table / inter-table :
- la contrainte porte sur une ou plusieurs tables

mono-attribut / multi-attribut :
- la contrainte concerne un ou plusieurs attributs

individuelle / ensembliste :
- la contrainte concerne un tuple ou une table

statique / dynamique : la contrainte s'applique à un état ou à un changement d'état :
- *statique* : condition vérifiée pour tout état
- *dynamique* : condition entre l'état avant et l'état après le changement

différées / immédiates :
- **validée au moment ou à la fin de la transaction**

Types de contraintes d'intégrités (3)

contraintes de domaine
- valeurs prises par les attributs : typage; plage de valeurs; défini en extension…
- acceptation de la valeur nulle (NOT NULL),
- valeur par défaut

contraintes d'entité
- clé primaire de table (NOT NULL)
- clé primaire candidates (UNIQUE)

contraintes référentielle
- cohérence entre les clé étrangères et les clé primaires des tables référencées

dépendences fonctionnelles (intra-table / intra-tuple) :
ex : modèle → marque, modèle → puissance,...

contraintes arithmétiques (inter-table, intra-table) :
ex : pour chaque produit p, à tout instant, quantité en stock Qs =∑Qa-∑Qv ,...

contraintes d'intégrités temporelles (dépendantes du temps)
ex: vraies en fin de mois, seule évolution croissante de valeur acceptée,...

contraintes d'intégrités générales
spécifiant la cohérence des données entre elles. ex: valeurs non-nulles, dépendances fonctionnelles diverses,...

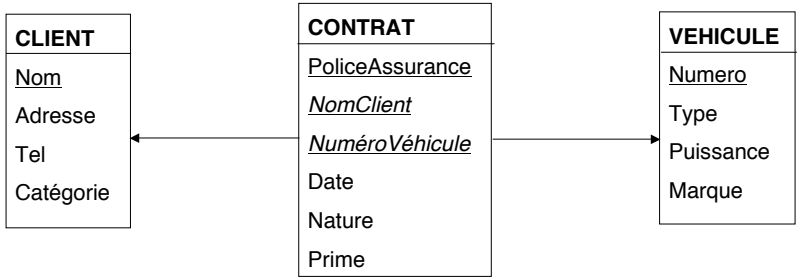
Contrôle sur les contraintes d'intégrité

- **Contrôle sémantique sur les C.I. :**
  - **conformité avec le schéma de la base**
    tout attribut ou table cité dans la C.I. appartient au schéma
  - **conformité avec les données de la base**
    toutes les données déjà existantes dans la base doivent respecter la nouvelle C.I.
  - **conformité avec les autres C.I.**
    détection de contradiction ou redondances, ...

CODD (90) propriétés « CRUDE » de l'intégrité d'une BDR :
- **C : "Colonne"** : intégrité du typage d'un attribut
- **R : "Référence"** : intégrité référentielle
- **U : "Utilisateur"** : intégrité applicative sur les règles de gestion
- **D : "Domaine"** : intégrité de domaine
- **E : "Entité"** : intégrité de clé primaire

Explicitation des CI (1)

Un exemple (Bouzeghoub, Jouve, Pucheral 1990) :



- **CLIENT** (Nom, Adresse, Tel, Catégorie, Age)
- **VEHICULE** (Numéro, Type, Puissance, Marque, Année)
- **CONTRAT** (PoliceAssurance, *NomClient*, *NuméroVéhicule*, Date, Nature, Prime, Bonus, Malus)

Explicitation des CI (2)

CLIENT (Nom, Adresse, Tel, Catégorie, Age)
VEHICULE (Numero, Type, Puissance, Marque, Année)
CONTRAT (PoliceAssurance, NomClient*, NuméroVéhicule*, Date, Nature, Prime, Bonus, Malus)

- **Contraintes d'intégrité :**

a) C.I. Clé:
Nom, Numero et PoliceAssurance sont des clés de relations

b) C.I. Dépendances fonctionnelles:
Table VEHICULE: Type → puissance, Type → marque,

c) C.I. Références:
NomClient (Table CONTRAT) réfère à Nom (Rel.CLIENT),
NuméroVéhicule (Table CONTRAT) réfère à Numéro (Rel.VEHICULE),

d) C.I. Domaine:
Table CONTRAT: Nature ∈ {TR (tous risques), RC (Resp.civile), RCBG (resp.civ. bris glace), RCIN (resp.civ.incendie), RCVO (resp.civ. vol)},

e) C.I. Générale:
Table CLIENT: Age doit être ≠ de la valeur nulle (⊥) si sa Catégorie = 'personne physique',

f) C.I. Générale:
Table CLIENT: si Age < 25 et Puissance > 10 alors Prime > 10000.

Expression des CI en logique des prédicats

- **a) clés :**
  - ∀ T1/CLIENT,∀ T2/CLIENT; T1.Nom = T2.Nom => T1 =T2.
  - ∀ T1/VEHICULE,∀ T2/VEHICULE; T1.Numéro = T2.Numéro => T1 =T2.
  - ∀ T1/CONTRAT,∀ T2/CONTRAT; T1.PoliceAssurance = T2.PoliceAssurance => T1 =T2.
- **b) dépendances fonctionnelles :**
  - ∀ T1/VEHICULE,∀ T2/VEHICULE; T1.Type = T2.Type => T1.Puissance =T2.Puissance.
  - ∀ T1/VEHICULE,∀ T2/VEHICULE; T1.Type = T2.Type => T1.Marque =T2.Marque.
- **c) référentielles :**
  - ∀ T/CONTRAT,∃ C/CLIENT; C.Nom = T.NomClient
  - ∀ T/CONTRAT,∃ V/VEHICULE; V.Numéro = T.NuméroVéhicule
- **d) de domaine :**
  - ∀ R/CONTRAT; (R.Nature = TR)V(R.Nature = RC)V(R.Nature = RCBG)V (R.Nature = RCIN)V(R.Nature = RCVO)
- **e) générales :**
  - ∀ C/CLIENT; C.Catégorie = 'personne_physique' => C.Age ≠ ⊥ *(⊥ = valeur nulle)*
- **f) générales :**
  - ∀ C/CLIENT, ∀ V/VEHICULE, ∀ R/CONTRAT; (C.Nom = R.NomClient) ∧ (V.Numéro = R.NuméroVéhicule) ∧ (C.age<25) ∧ (V.Puissance > 10) => R.Prime > 10000.

Expression des CI en SQL étendu

a) CI lés → ***impossible en SQL de comparer des variables tuples entre elles, il faut reformuler la contrainte et l'exprimer en comptant le nombre de tuples ayant la même valeur pour l'attribut Nom:***

Assert CléClient **On** CLIENT
As Select * **From** CLIENT **Group by** Nom **Having Count** (*) >1;

si la repose à cette requête SQL est vide, alors la C.I. est vérifiée, sinon elle est violée et les tuples sélectionnés sont ceux violant la C.I.

→ **La plupart des SGBD permettent de déclarer les clés plus simplement, par exemple les options Not Null et Unique de la commande Create Table. Dans ancienne version d'ORACLE on peut spécifier aussi un index dense: Create Unique Index *indnom* On *client(nom).***

b) Dépendances fonctionnelles

Assert DF1 **On** VEHICULE
As Select All (Type, Puissance)
From VEHICULE T1, VEHICULE T2
Where T1.Type ≠T2.Type
or T1.Puissance ≠T2.Puissance;

c) CI référentielles

Assert CR1 **On** CONTRAT
As Select Distinct NomClient **From** CONTRAT R **Where**
Not Exists (**Select Distinct** Nom **From** CLIENT C **Where** (R.NomClient = C.Nom);

Expression des CI en SQL étendu (2)

- **d) CI de domaine**

Assert DomNature **On** CONTRAT
As Select Nature **From** CONTRAT
Where Nature≠'TR' **and** Nature≠'RC' **and** Nature≠'RCBG' **and** Nature≠'RCIN' **and** Nature≠'RCVO';

→ généralement les domaines sont définis plus simplement lors de la création des tables avec commande **Check**:

Create Table CONTRAT R (PoliceAssurance **Number**(10), ... Nature **Char**(10));
Check R.Nature **In** ('TR', 'RC', 'RCBG', 'RCIN', 'RCVO');

- **e) CI générales**

Assert CG1 **On** CLIENT **As Select** * **From** CLIENT **Where** Catégorie = 'personne_physique' **and** Age **Is Null**;

- **f) CI générales**

Assert CG2 **On** CONTRAT
Where (Select * **From** CLIENT C, VEHICULE V
Where C.Age<25 **and** V.Puissance>10 **and Not Exists**
(**Select** NomClient, NuméroVéhicule **From** CONTRAT R **Where** R.NomClient = C.Nom **and** R.NuméroVéhicule=V.Numéro **and** R.Prime>10000));

CI et langage de règles : le langage RAISIN (1)

Langage RAISIN (Stonebraker 1982) : cadre uniforme par langage de règles pour la sécurité et l'intégrité des données dans le SGBD Postgres.

- **Format général : ON <condition> THEN <action>**

avec

<condition> :=
ON <opération de mise à jour>
TO <table> (AFFECTING < >, QUALIFYING < >)
BY DURING <heure> FOR <jour> (WHERE < >)

<action> :=
{EXECUTE, CANCEL, UNDO, Commande SQL, MESSAGE, ADDQUAL}.

- **Contrainte de confidentialité en RAISIN :**
  - pré-condition concernant un utilisateur et mettant un jeu un ensemble de valeurs, via une requête
  - ADDQUAL : modification automatique d'une requête utilisateur par rajout de contraintes de confidentialité associées aux données concernées

ON * TO pilote BY bernard THEN ADDQUAL pilote_adresse ="Aix";
→ ***bernard ne peut accéder qu'aux pilotes habitant à Aix***

CI et langage de règles : le langage RAISIN (2)

- **Contrainte d'intégrité :** autorise une mise à jour de la BD si une pré-condition est satisfaite :

  ON UPDATE, INSERT DELETE TO avion WHERE AVG (avion_capacité) <= 150 THEN UNDO;

  → la moyenne des capacités des avions doit être supérieure à 150

- **Déclencheur (ou démon ou trigger)**
  - entraîne une activation de procédure si une pré-condition est satisfaite
  - pré-condition associée à des cdes de mises à jour et aux tuples modifiés)
  - la procédure peut comporter une requête SQL

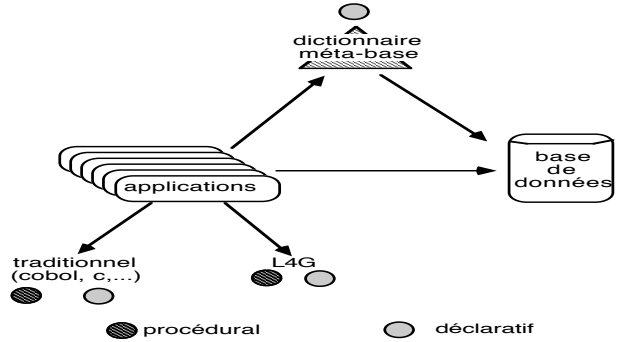
  ON UPDATE, INSERT TO vol AFFECTING ndisp <=0 THEN EXECUTE SURBOOKING;

  → activer la transaction SURBOOKING lorsque ndisp=0

- **Alerteur**
  - entraîne l'envoi d'un message si une pré-condition est satisfaite
  - gérés par serveur, permettent d'informer automatiquement les clients des changements critiques survenant sur la BD

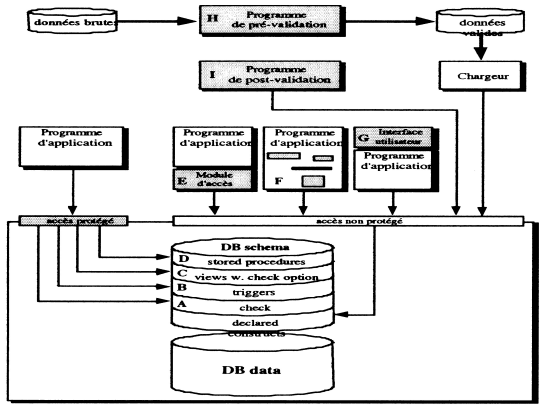
  ON UPDATE TO vol AFFECTING ville_arrivée THEN MESSAGE attention : vous modifier la ville_arrivée d'un vol

Prise en compte des CI dans les SGBDR (1)



- **traitement des contraintes d'intégrités:**
  - → **procédural** au niveau des applicatifs (3LG, 4LG)
  - → **déclaratif** au niveau des applicatifs (4LG)
  - → **déclaratif** au niveau du dictionnaire

Prise en compte des CI dans les SGBDR (2)



Prise en compte des CI dans les SGBDR (3)

En SQL :

- **A. Prédicats de contrainte SQL** **(check, assertions)**
- **B. Procédures déclenchées SQL** **(triggers)**
- **C. Vues filtrantes SQL** **(views with check option)**
- **D. Procédures SQL associées au schéma** **(stored procedures)**

Hors SQL :

- **E. Modules d'accès**
- **F. Sections de code distribuées dans les programmes d'application**
- **G. Procédure de validation attachées aux écrans de saisies**
- **H. Programme de validation avant chargement**
- **I. Programme de validation après chargement …**

CI prises en compte en SQL

En SQL, 2 classes de contraintes :

1. Contraintes inhérentes au modèle relationnel :

- intégrité des **domaines**
- intégrité de **relations** (clés **identifiantes**)
- intégrité de **références** (clés **étrangères**)

Requêtes SQL, CREATE TABLE et ALTER TABLE …

2. Contraintes spécifiques :

- **contraintes générales** (évaluées pour toute modification de la base de données)
- **contraintes attachées** aux tables de base
- **3 types de requêtes SQL :**
  - **CREATE ASSERTION ... CHECK**
  - **CHECK**
  - **TRIGGER**

2. Prise en compte en SQL des CI inhérentes au modèle relationnel

- CI de domaines,
- Clés primaires
- Clés uniques
- Clés étrangères

CI inhérentes au modèle relationnel en SQL

- **Contraintes de domaines :**
  - **Types, NOT NULL, CHECK, CREATE DOMAIN, DEFAULT**
- **Contraintes de clé primaire :**
  - **PRIMARY KEY**
- **Contraintes d'unicité :**
  - **UNIQUE**
- **Contraintes d'intégrité référentielle :**
  - **FOREIGN KEY**
- **Désactivation et réactivation de contraintes**

Contraintes de domaines : Types SQL

```
INTEGER, CHAR, DATE, …
```

Contraintes de domaines : NOT NULL, CHECK

• Contrainte NOT NULL :

```
CREATE TABLE circuits
(nocir char (6)          NOT NULL,
designcir char (20)      NOT NULL,
distancecir char (6)     NOT NULL)
```

• Contrainte CHECK sur une colonne : le nocir est supérieur à 0 et inférieur à 10000 :

```
CREATE TABLE circuits
(nocir char (6)          NOT NULL
designcir char (20)      NOT NULL,
distancecir char (6)     NOT NULL
CHECK (distancecir > 0 AND distancecir < 200))
```

Contraintes de domaines : CREATE DOMAIN DEFAULT

• **CREATE DOMAIN - Création d'un domaine « sexe » :**

```
CREATE DOMAIN domaineSexe
 CHAR (1)  CHECK (VALUE IN ('M', 'F'))


CREATE TABLE Employé
(…
sexe   domaineSexe,
…
)
```

• **DEFAULT - Valeur par défaut :**

```
CREATE TABLE client
(…
noTelephone VARCHAR(15) DEFAULT 'Confidentiel' NOT NULL
…
)
```


Contraintes de domaines : CREATE DOMAIN

• **Création de DOMAINE : CREATE DOMAIN**

```
CREATE SCHEMA
<création de 2 domaines ville et heure>
    CREATE DOMAIN ville AS CHAR(12)
    DEFAULT 'Paris'
    CHECK (VALUE IN ('Paris', 'Marseille', 'Nice', 'Aix'))

    CREATE DOMAIN heure AS HOUR
    CHECK (VALUE >7 AND VALUE < 22)
<création de la table pilote>
    CREATE TABLE pilote
    (no_pilote DECIMAL (4),
    nom_pilote CHAR(12),
    Adresse_pilote Ville CHECK (VALUE IN ('Aix', 'Marseille')),
    salaire_pilote DECIMAL (5)
    PRIMARY KEY (no_pilote))
```


SQL 2 : définition de données

• **Contrainte sur DOMAINE :**

Une contrainte sur un domaine peut être nommée et ultérieurement supprimée :

```
CREATE DOMAIN ville AS CHAR(12)
   CONSTRAINT C_Ville CHECK (VALUE IN ( 'Paris', ...))
```

• **Modification et suppression de DOMAINE :**

```
ALTER DOMAIN
DROP DOMAIN
DROP CONSTRAINT
```

• **Contrainte générale :**

```
CREATE (DROP) ASSERTION airbusloc CHECK
(NOT EXIST (SELECT * FROM avion
 WHERE nom_avion = 'AIRBUS' AND localisation ≠ 'Paris'));
```

→ *création d'une contrainte noté "airbusloc" satisfaite s'il n'existe pas d'avion de marque Airbus non localisé à Paris*

Remarque : la clé étrangère FOREIGN KEY X REFERENCES T (Y) correspond à la vérification de contrainte : CHECK (X MATCH (SELECT Y FROM T)).

Contraintes de clé primaire : PRIMARY KEY

• **déclaration d'une clé primaire au moment de la création d'une table :**

```
CREATE TABLE circuits
(nocir char(6)    NOT NULL,
designcir char(20)    NOT NULL,
distancecir char(6)   NOT NULL,
PRIMARY KEY (nocir)
)
```

Remarque : un attribut déclaré clé primaire doit être défini avec l'option NOT NULL

• **ajouter une clé primaire à une table existante :**

```
CREATE TABLE circuits –
(nocir char (6)    NOT NULL, –
designcir (20)    NOT NULL, –
distancecir (6)   NOT NULL))
```

déclaration d'une clé primaire sur la table circuits :

```
ALTER TABLE circuits
  ADD PRIMARY KEY (nocir)
```

Remarque : s'il existe des tuples de la table circuits ayant la même valeur pour l'attribut nocir, la commande alter n'aboutit pas (violation de la contrainte)

Clé primaire composée

• **déclaration d'une clé primaire composée :**

```
CREATE TABLE ligneCommande
(noCommande    INTEGER   NOT NULL,
noArticle      INTEGER   NOT NULL,
quantité       INTEGER   NOT NULL,
PRIMARY KEY (noCommande, noArticle))
```

Contrainte d'unicité : UNIQUE

• un ou plusieurs attributs d'une table peuvent identifier de façon unique chaque tuple de la table
• similaire à une clé primaire, **mais ne peut être référencée par une clé étrangère**
• **déclaration d'une contrainte d'unicité au moment de la création d'une table :**

```
CREATE TABLE circuits
(nocir      char (6)      NOT NULL,
designcir   char(20)      NOT NULL, UNIQUE
distancecir char(6)       NOT NULL)
```

Remarques :

1- un attribut déclaré en contrainte d'unicité doit être défini avec l'option NOT NULL
2- une table peut avoir plusieurs contraintes d'unicité

Contraintes d'intégrité référentielle : FOREIGN KEY

• **Déclaration d'une clé étrangère au moment de la création d'une table :**

```
CREATE TABLE voyages
(
novoy char (6)     NOT NULL,
nocir char (6)     NOT NULL,
datedeb date    NOT NULL,
PRIMARY KEY (novoy)
FOREIGN KEY r_nocir (nocir) REFERENCES circuits ON DELETE
  CASCADE
)
```

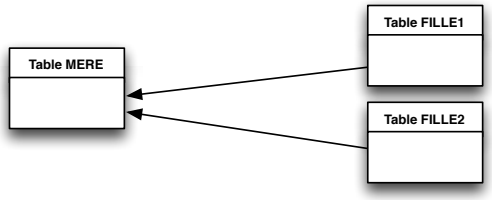
Remarques :

1- la table **parente** doit déjà **exister**
2- l'attribut de la table parente doit avoir été définie en **clé primaire**
3- le nom de cette contrainte d'intégrité référentielle est **r_nocir**, il sera utilisé pour désactiver, activer et supprimer la contrainte.
4- **Option de suppression** : **ON DELETE CASCADE** signifie que si un tuple de la table circuits est détruit, tous les tuples de la table voyages s'y référant sont détruits automatiquement

Options de suppression (ON DELETE) sur clé étrangère (1)

Soit une table MERE reliée à une ou plusieurs tables FILLES par des clés étrangères et considérant la suppression d'un tuple de la table MERE :

```
CREATE TABLE mere (
   Id_mere INT NOT NULL,
   …
   PRIMARY KEY (id)
);
CREATE TABLE fille (
   Id_fille INT NOT NULL,
   mere_id INT,
   …
   FOREIGN KEY (mere_id) REFERENCES mere(id) ON DELETE « OPTION »
);
```



avec OPTION : = CASCADE | NO ACTION | RESTRICT | SET NULL | SET DEFAULT

Options de suppression (ON DELETE) sur clé étrangère (1)

- **CASCADE :** le tuple de la table MERE est supprimé ainsi que tous les tuples dépendants des tables FILLES
- **NO ACTION** : si des tuples référencées des tables FILLE existent encore quand la contrainte est vérifiée, une erreur est générée (***option par défaut***)
- **RESTRICT** : le tuple de la table MERE est supprimé SSI aucun autre tuple des tables FILLE ne dépend de lui (les tuples dépendants des tables FILLES doivent être supprimés ou modifiés AVANT !)
- **SET NULL** (pour clés étrangères autorisant le NULL) : le tuple de la table MERE est supprimé, toutes les clés étrangères des FILLES de valeur égale à la clé primaire du tuple MERE sont automatiquement mises à NULL
- **SET DEFAULT** (pour clés étrangères ayant une valeur par défaut) : le tuple de la table MERE est supprimé, et toutes les clés étrangères des FILLES de valeur égale à la clé primaire du tuple MERE sont automatiquement mises à leur valeur par défaut

***Remarque** : NO ACTION permet de différer le contrôle à la fin de la transaction, alors que RESTRICT ne le permet pas.*

Options (ON DELETE) sur clé étrangère : exemple 1

Exemple :

- CIRCUITS (nocir, designcir, distancecir)
- VOYAGES (**novoy**, ***nocir***, datedeb,...)

```
CREATE TABLE voyages
...
FOREIGN KEY r_nocir (nocir) REFERENCES circuits ON DELETE
  CASCADE
```

On a : le tuple de la table mère CIRCUITS est supprimé ainsi que tous les tuples dépendants de la table fille VOYAGES

Options (ON DELETE) sur clé étrangère : exemple 2

Soit les tables suivantes :

```
CREATE TABLE produits (
    produit_no integer PRIMARY KEY,
    …
);
CREATE TABLE commandes (
    commande_id integer PRIMARY KEY,
    …
);
CREATE TABLE Ligne_commande (
    commande_id integer REFERENCES commandes ON DELETE CASCADE,
    produit_no integer REFERENCES produits ON DELETE RESTRICT,
quantite integer,
    PRIMARY KEY (produit_no, commande_id)
);
```

On a :

- Si le tuple de la table mère COMMANDE est supprimé, alors tous les tuples dépendants de la table fille LIGNE_COMMANDE sont supprimés
- Le tuple de la table mère PRODUITS ne peut être supprimé que si tous les tuples dépendants de la table fille LIGNE_COMMANDE l'ont été avant.

Options de modification (ON UPDATE) sur clé étrangère (1)

Soit les 2 tables suivantes :

```
CREATE TABLE parent (
    idpar INT NOT NULL,
    …
    PRIMARY KEY (id)
);
CREATE TABLE enfant (
    idenf INT NOT NULL,
    parent_id INT,
    …
   FOREIGN KEY (parent_id) REFERENCES parent(id) ON UPDATE CASCADE
);
```

Signifie que si par exemple est faite la mise à jour :

```
UPDATE parent SET idpar = 20 WHERE idpar = 10
```

Alors tous les enfants de parent_id de valeur 10, verront la valeur de parent_id également mis à jour à 20

Remarque : si le champ auquel la clé étrangère fait référence n'est jamais mise à jour, cette spécification ON UPDATE CASCADE n'est pas nécessaire

Ajout d'une clé étrangère

```
CREATE TABLE voyages –
(novoy char (6)    NOT NULL, –
(nocirc char (6)    NOT NULL, –
(datedeb date   NOT NULL)
```

Ajout d'une clé étrangère sur la table *voyages* :

```
ALTER TABLE voyages –
ADD FOREIGN KEY r_nocir (nocir) REFERENCES circuits ON DELETE
  CASCADE
```

Remarques :

1- la table MERE (circuits) doit déjà exister

2- l'attribut de la table MERE (circuits) doit avoir été définie en clé primaire

3- s'il existe des tuples de la table voyages dont la valeur de l'attribut *nocir* n'est pas une valeur prise par l'attribut *nocir* de la table circuits, commande ALTER n'aboutit pas (violation de la contrainte)

Désactivation et réactivation de contraintes

- **Désactivation** d'une **clé primaire :**

```
ALTER TABLE circuits DESACTIVATE PRIMARY KEY
```

1 - les insertions, suppressions et mises à jour sont suspendues tant que la clé est désactivée

2 - les contraintes référentielles concernées par la clé primaire désactivées sont aussi désactivées (exemple: la contrainte de référence définie dans la table voyages est désactivée, cette table n'est plus accessible)

3 - aucun autre utilisateur ne peut accéder à la table tant que la clé est désactivée

- **Réactivation** d'une **clé primaire :**

```
ALTER TABLE circuits ACTIVATE PRIMARY KEY
```

- **Désactivation/réactivation** d'une **clé étrangère** (**r_nocir** est le nom de la contrainte référentielle concernée) :

```
ALTER TABLE voyages DESACTIVATE/ACTIVATE FOREIGN KEY r_nocir
```

- **Désactivation/réactivation** d'une **contrainte d'unicité:**

```
ALTER TABLE circuits DESACTIVATE/ACTIVATE UNIQUE designcir
```

insertions et mises à jour sont suspendues tant que la clé est désactivée.

3. Prise en compte en SQL de CI spécifiques et Triggers

- Types de CI spécifiques
- CI générales
- CI spécifiques associées aux tables
- CI spécifiques liées à un événement : Triggers

Types de CI spécifiques

- **Contraintes générales :**
  - CHECK ASSERTION, CHECK …
- **Contraintes attachées aux tables :**
  - CONSTRAINT
  - DEFERRABLE / NOT DEFERRABLE
- **Triggers :**
  - définition
  - usages

CI générales : CREATE ASSERTION

Création d'une contrainte générale : commande CREATE ASSERTION

Syntaxe : `CREATE ASSERTION nom_contrainte CHECK (prédicat) ;`

- **prédicat** = une propriété des données qui doit être vérifiée à chaque instant
- Si **une commande SQL viole le prédicat**, cette commande est annulée et un message d'erreur (exception) est généré

Suppression d'une contrainte générale : commande DROP ASSERTION

Syntaxe : `DROP ASSERTION nom_contrainte;`

CI spécifiques : exemples

Tables *(xxx : clé primaire ; xxx* : clé étrangère) :*

- ELEVES (numEleve, …) ;
- COURS (numCours, …) ;
- RESULTATS (numEleve*, numCours*, points) ;
- PROFESSEURS (numProf, …, salaire_base, salaire_actuel, …) ;
- CHARGES (numCours, numProf) ;

1 - Chaque professeur doit posséder un salaire de base positif :

```
CREATE ASSERTION c1 CHECK
(NOT EXISTS (SELECT * FROM professeurs
                  WHERE (salaire_base<0))
);
```

2 - Le salaire actuel de chaque professeur doit être supérieur à son salaire de base :

```
CREATE ASSERTION c2 CHECK
(NOT EXISTS (SELECT * FROM professeurs
                  WHERE (salaire_base < salaire_actuel))
);
```

3 - Tout prof. doit donner au moins un cours ⇔ Il n'existe pas de prof. qui ne donne pas de cours :

```
CREATE ASSERTION c3 CHECK
(NOT EXISTS (SELECT * FROM professeurs
                  WHERE numProf NOT IN
                      (SELECT DISTINCT numProf FROM charge))
) ;
```


CI spécifiques associées aux tables (1)

Contraintes définies lors de la création de la table (CREATE TABLE) ou après la création de la table (ALTER TABLE)

Syntaxe : CONSTRAINT

```
CREATE TABLE nom_table (liste_def_colonne)
[liste_contrainte_table];

contrainte_table ::=
CONSTRAINT nom_contrainte type_contrainte_table
mode_contrainte

type_contrainte_table ::=
      PRIMARY KEY
    | UNIQUE (liste_colonne)
    | CHECK (condition)
    | FOREIGN KEY (liste_colonne)
    | REFERENCES nom_table (liste_colonne)

mode_contrainte ::= [NOT] DEFERRABLE
```


CI spécifiques associées aux tables (2)

Modes de contrainte :

- **DEFERRABLE :** évaluation ultérieure (fin d'une transaction)
- **NOT DEFERRABLE :** évaluation lors de l'instruction de la mise à jour

1 - Contrainte sur le domaine de valeurs :

Exemple : Dans chaque cours, les points sont toujours compris entre 0 et 20 :

```
CREATE TABLE resultats
(numEleve numero_d_eleve,
 numCours num_de_cours,
 points integer
        CONSTRAINT plage_des_points
        CHECK (points BETWEEN 0 AND 20)
        NOT DEFERRABLE
) ;
```


CI soécifiques associées aux tables (3)

2 - Contrainte horizontale : la valeur d'un attribut est fonction des valeurs apparaissant dans les autres attributs

Ex: Le salaire actuel d'un professeur doit être supérieur ou égal à son salaire de base :

```
CREATE TABLE professeurs
( ...
      CONSTRAINT salaire_base_actuel
      CHECK (salaire_actuel >= salaire_base)
) ;
```

3 - Contrainte verticale : la valeur d'un attribut d'une ligne est fonction des valeurs de cet attribut dans les autres lignes

Ex: Le salaire actuel d'un professeur ne peut dépasser le double de la moyenne des salaires de sa spécialité :

```
CREATE TABLE professeurs
( …
   CONSTRAINT salaire raisonnable
   CHECK (salaire_actuel <= 2*(SELECT AVG(salaire_actuel)
                               FROM professeurs p
                               WHERE p.specialite =
                                 professeurs.specialite)
                               )
          )
);
```

CI spécifique associées aux tables (4)

4 - Contrainte faisant référence à une autre table :

Exemple : Le salaire actuel d'un professeur donnant le cours numéro 5 doit être supérieur à 6000 euros :

```
CREATE TABLE professeurs
( ...
     CONSTRAINT salaire_prof_cours_5
     CHECK (6000 < (SELECT salaire_actuel
                    FROM professeurs
                    WHERE numProf IN
                         (SELECT numProf FROM charges
                          WHERE numCours = 5)
                    )
            )
);
```


Contraintes associées à un événement : Trigger

TRIGGER (Déclencheur) = mécanisme permettant de définir un ensemble **d'actions** déclenchées automatiquement par le SGBD lorsque certains événements (phénomènes) se produisent.

- Constitué d'une section de **code** avec des **conditions** entraînant son exécution
- **Forme générale**: **E – C – A** (**E**vent – **C**ondition - **A**ction)

```
BEFORE | AFTER E
WHEN C
BEGIN
     A
END
```

SI un événement **E** (INSERT, DELETE, UPDATE) survient, SI la condition **C** est satisfaite, alors exécuter l'action **A** soit avant (BEFORE) soit après (AFTER) l'événement **E**

- **Usages :**
  - **définition de contraintes dynamiques** : contraintes concernant le passage de la base de données d'un état dans un autre,
  - **génération automatique d'un numéro de clé primaire,**
  - **résolution du problème des mises à jour en cascade**, …

Triggers – déclencheur : Oracle

Longtemps chaque SGBD a proposé sa propre syntaxe et ses propres extensions de triggers, maintenant partiellement normalisé dans SQL3.

- **Syntaxe Oracle :**

```
CREATE TRIGGER nom_déclencheur
BEFORE | AFTER
DELETE | INSERT | UPDATE [OF liste_colonne]
ON nom_table
[ FOR EACH ROW ]
[ WHEN condition ]
[bloc PL/SQL]
```

  - la clause **FOR EACH ROW** :
    - précise que le déclencheur est au niveau de **ligne** : le bloc PL/SQL **sera exécuté pour toutes les lignes** provoquant l'activation du déclencheur
    - l'absence de cette clause spécifie un déclencheur du niveau de la **table** : le bloc PL/SQL ne sera **exécuté qu'une seule fois**
  - la clause **WHEN** : permet de préciser une condition supplémentaire

Triggers – déclencheur : Oracle

Restrictions

- **Table mutante :** table en cours de modification (pour un trigger, il s'agit de la table sur laquelle il est défini)
- **Table contraignante :** table qui peut éventuellement être accédée en lecture afin de vérifier une contrainte de référence (exemple : la table ELEVES est contraignante pour la table RESULTATS)
- **Restrictions sur les commandes du bloc PL/SQL :**
  - Il **ne peut contenir de COMMIT ni de ROLLBACK**
  - **Si le déclencheur est de niveau ligne** :
    - Il **ne peut lire** ou **modifier** le **contenu** d'une **table mutante**
    - Il **ne peut lire** ou **modifier** les **colonnes d'une clé primaire, unique** ou **étrangère** d'une **table contraignante**

Usage des triggers : contraintes dynamiques

Exemple : Le salaire de base d'un professeur ne peut jamais diminuer

```
CREATE TRIGGER upd_salaire_professeur
BEFORE UPDATE OF salaire_base ON PROFESSEUR
FOR EACH ROW
WHEN (OLD.salaire_base > NEW.salaire_base)
DECLARE
    salaire_diminue EXCEPTION;
BEGIN
    RAISE salaire_diminue;                                          [1]
EXCEPTION
    WHEN salaire_diminue THEN                                       [2]
    raise_application_error (-20001, 'le salaire ne peut diminuer') ;
END;
```

- **OLD** et **NEW** sont prédéfinis et permettent d'accéder aux anciennes et aux nouvelles valeurs
- Le **bloc PL/SQL** se limite à lancer une **exception** [1] : l'exécution est interceptée et traitée en [2]

Usage des triggers : gestion automatique des clés primaires (1)

Exemple : Dans la table Eleve, on souhaite générer automatiquement la valeur de la clé primaire de la table numEleve

Méthode 1 :

nouvelle valeur de numEleve = nombre de lignes présentes dans la table + 1

trigger associé :

```
CREATE TRIGGER tri_cle-prim
AFTER INSERT ON eleves
DECLARE
    nbre INTEGER;
BEGIN
    SELECT COUNT(*) INTO nbre FROM eleves;
    UPDATE eleves SET numEleve = nbre + 1
    WHERE numEleve IS NULL;
END;
```


Usage des triggers : gestion automatique des clés primaires (2)

Exemple : Dans la table Eleve, on souhaite générer automatiquement la valeur de la clé primaire de la table numEleve

Méthode 2 : nouvelle valeur de numEleve = nombre maximum de numEleve + 1

Solution 1:

```
CREATE TRIGGER tri_cle_prim
AFTER INSERT ON eleves
DECLARE
    nbre INTEGER;
BEGIN
    SELECT MAX(numEleve) INTO nbre FROM eleves;
    UPDATE eleves SET numEleve = nbre + 1
    WHERE numEleve IS NULL;
END;
```

Pb non gérés:

- la fonction MAX appliquée à un ensemble vide donne pour résultat NULL
- NULL + 1 donne encore NULL

=> le déclencheur ainsi défini place NULL dans numEleve pour la première et toutes les autres lignes insérées

Usage des triggers : gestion automatique des clés primaires (3)

Méthode 2 : nouvelle valeur de numEleve = nombre maximum de numEleve + 1

Solution 2:

```
CREATE TRIGGER tri_cle_prim
AFTER INSERT ON eleves
DECLARE
    nbre INTEGER;
BEGIN
    SELECT MAX(numEleve) INTO nbre FROM eleves;
    IF nbre IS NULL THEN
        UPDATE eleves SET numEleve = 1;
    ELSE
        UPDATE eleves SET numEleve = nbre + 1
        WHERE numEleve IS NULL;
    END IF;
END;
```

On notera que le déclencheur est défini au niveau table.

Usages des triggers : mises à jour en cascade

Problèmes liés aux mises à jour effectuées uniformément dans plusieurs tables :

lorsqu'une clé primaire est modifiée ou lorsqu'on supprime d'une table une ligne qui est référencée par d'autres lignes d'autres tables

Exemple : Solution de déclencheurs pour les tables cours et charge

- Table cours : clé primaire numCours
- Table charges : clé étrangère numCours

```
CREATE TRIGGER maj_cascade
BEFORE DELETE OR UPDATE OF numCours ON cours
FOR EACH ROW
BEGIN
    IF (updating) THEN
        UPDATE charges SET numCours =:new.numCours
        WHERE numCours = :OLD.numCours ;
    ELSEIF (deleting) THEN
        DELETE FROM charges
        WHERE numCours = :OLD.numCours ;
    END IF;
END;
```