Administration de Bases de Données

– SQL avancé –

SKLAB Youcef

April 25, 2017

Structured Query Language (SQL)

Structured Query Language (SQL)

1. Caractéristiques de SQL
2. Définition de données
   - Contraintes d'intégrité
   - Créer / Modifier /Supprimer une table
   - Index
3. Modifier une base de données
4. Interroger une base de données
5. Contrôle des Données
   - Restriction des accès- Sous Schéma / Schéma Externe
   - Gestion des vues
   - Restriction des Actions: Les privilèges
   - Les Rôles
6. Les Déclencheurs

Introduction

- **SQL**: Structured Query Language
- Accéder aux données de la base de données,
- Langage adapté aux bases de données relationnelles,
- Existe sur tous les SGBD relationnels (Oracle, SQL Server, Access,...),
- Défini par une norme ISO

Caractéristiques de SQL

LDD - LMD - LCD

SQL est un Langage de Définition des Données (LDD)

Il permet de créer des tables dans une base de données relationnelle, ainsi que d'en modifier ou en supprimer.

LDD - LMD - LCD

SQL est un Langage de Définition des Données (LDD)

Il permet de créer des tables dans une base de données relationnelle, ainsi que d'en modifier ou en supprimer.

SQL est un langage de manipulation de données

Il permet de sélectionner, insérer, modifier ou supprimer des données dans une table d'une base de données relationnelle.

LDD - LMD - LCD

SQL est un Langage de Définition des Données (LDD)

Il permet de créer des tables dans une base de données relationnelle, ainsi que d'en modifier ou en supprimer.

SQL est un langage de manipulation de données

Il permet de sélectionner, insérer, modifier ou supprimer des données dans une table d'une base de données relationnelle.

SQL est un langage de contrôle (protection) d'accès

Il est possible avec SQL de définir des permissions au niveau des utilisateurs d'une base de données. On parle de Langage de Contôle de Données (LCD).

Définition de données

Créer / supprimer une Bases de données

Créer une Bases de données

CREATE DATABASE nom_bdd;

Supprimer une Bases de données

DROP DATABASE nom_bdd;

Créer une table

CREATE TABLE

```
CREATE TABLE   nom_table
(col1 TYPE(taille) [DEFAULT expression] [NOT NULL],
col2 TYPE(taille) [DEFAULT expression] [NOT NULLl],
col3 TYPE(taille) [DEFAULT expression] [NOT NULL] ,
...
[PRIMARY KEY (col n, col M...)]
[FOREIGN KEY (col N, col M...) REFERENCES nom_table2 (col I)]
);
```

Créer une table

Exemple 1

```
CREATE TABLE ELEVE
(NOM CHAR(20) PRIMARY KEY,
PNOM CHAR(20) NOT NULL,
NOTE NUMBER(4,2) DEFAULT 0,
NUMCLASSE CHAR(4),
TELEPHONE CHAR(10) NOT NULL UNIQUE,
FOREIGN KEY (NUMCLASSE) REFERENCES CLASSE (CODECLASSE)
);
```

Attention à l'ordre de création des tables : une clé étrangère ne peut pas faire référence à la clé primaire d'une autre table qui n'existe pas.

Contraintes d'intégrité

Contraintes d'intégrité

Règle spécifiant les valeurs permises pour certaines données, éventuellement en fonction d'autres données, et permettant d'assurer une certaine cohérence de la base de données.

Types de contraintes

- contrainte de valeurs (contrainte de domaine)
- unicité d'occurrences (unicité de lignes)
- clé primaire (unicité et indexation pour l'intégrité d'entité)
- clé étrangère (intégrité référentielle)
- caractère obligatoire ou facultatif des valeurs.

Contraintes d'intégrité (suite)

- Les contraintes peuvent s'exprimer:
  - soit au niveau colonne (contraintes locales) : valables pour une colonne;
  - soit au niveau table (contraintes globales) : valables pour un ensemble de colonnes d'une table.
- Les contraintes se définissent:
  - soit lors de la création des tables, dans l'ordre **CREATE TABLE**;
  - soit après la création des tables, par l'ordre **ALTER TABLE** permettant certaines modifications de la structure des tables.

Contraintes d'intégrité (suite)

- Les contraintes de colonnes:
  - **NULL** ou **NOT NULL**: Saisie obligatoire ou pas.
  - **UNIQUE**: Valeur unique pour un champ dans la table.
  - **PRIMARY KEY**: Clé primaire élémentaire.
  - **REFERENCES**: Intégrité de référence.
  - **CHECK**: Contraintes de valeurs.
- Les contraintes de tables:
  - **UNIQUE**: Composition des colonnes.
  - **PRIMARY KEY**: Clé primaire composée
  - **FOREIGN KEY ... REFERENCES ...** : Composition de colonnes
  - **CHECK**: Condition particulière sur les valeurs.

Contraintes d'intégrité

Contraintes d'intégrité - Syntaxe

```
CREATE TABLE nom_table (
nom_col_1 type_1 [CONSTRAINT nom_1_1] contrainte_de_colonne_1_1
                 [CONSTRAINT nom_1_2] contrainte_de_colonne_1_2
...
nom_col_n type_n [CONSTRAINT nom_n_1] contrainte_de_colonne_n_1
                 [CONSTRAINT nom_n_2] contrainte_de_colonne_n_2
                 ...                  ...
                 [CONSTRAINT nom_n_m] contrainte_de_colonne_n_m,
[CONSTRAINT nom_1] contrainte_de_table_1,
[CONSTRAINT nom_2] contrainte_de_table_2,
...                 ...
[CONSTRAINT nom_p] contrainte_de_table_p
)
```

Exemple Contraintes d'intégrité

Exemple

CREATE TABLE Clients(
Nom char(30) **NOT NULL**,
Prenom char(30) **NOT NULL**,
Age **integer**, **check** (Age < 100),
Email **char(50) NOT NULL**, **check** (Email **LIKE** "%@%")
)

Créer une table

Création d'une table avec insertion de données

CREATE TABLE NomTable (
Nom_de_colonne1 Type_de_donnée,
Nom_de_colonne2 Type_de_donnée,
...
)
AS SELECT NomChamp1,
NomChamp2,
...
FROM NomTable2
WHERE ...;

Types de données

Quelques types de données

- Numériques: **SMALLINT** (16 bits), **INTEGER** (32 bits), **FLOAT**
- Chaînes de caractères: **CHAR(n)** (longueur fixée), **VARCHAR(n)** (longueur maximale fixée)
- Date: **DATE, TIME, TIMESTAMP**

Modifier une table

Il est possible de modifier la structure d'une table:

- Ajout de colonnes ou/et de contraintes;
- Modification de colonnes ou/et de contraintes;
- Changement du caractère obligatoire/facultatif (NULL);
- Rectification de la largeur d'une colonne (VARCHAR).

Modifier une table: ALTER TABLE

Renommer une table

ALTER TABLE nom_table RENAME TO nouveau_nom

Ajout ou modification de colonne

ALTER TABLE nom_table {ADD/MODIFY} ([nom_colonne type [contrainte], ...])

Renommer une colonne

ALTER TABLE nom_table RENAME COLUMN ancien_nom TO nouveau_nom

Supprimer une colonne

ALTER TABLE nom_table DROP COLUMN nom_colonne

Modifier une table: ALTER TABLE

Ajout d'une contrainte de table

ALTER TABLE nom_table ADD [CONSTRAINT nom_contrainte] contrainte

Supprimer des contraintes

ALTER TABLE NomTable
DROP CONSTRAINT NomContrainte

Modifier une table: Exemples

Exemples

- **ALTER TABLE ELEVE ADD (NAISS date);**
- **ALTER TABLE ELEVE MODIFY (NOM char(35));**

Modifier une table

TRUNCATE: suppression de données uniquement

TRUNCATE TABLE 'NomTable'

⇒ Permet de supprimer toutes les données d'une table sans supprimer la table en elle-même.

COMMENT: ajouter des commentaires à une table

COMMENT NomTable **IS** 'Commentaires';
COMMENT NomVue **IS** 'Commentaires';
COMMENT NomTable.NomColonne **IS** 'Commentaires';

Les index

Afin d'améliorer les temps de réponses ou tout simplement traduire la notion de clef primaire d'une table, SQL propose un mécanisme d'index. Il est utile pour accélérer l'exécution d'une requête SQL qui lit des données.

Les index

Un index est associé à une table, mais il est stocké à part. Un index peut ne porter que sur une colonne (**index simple**) ou sur plusieurs (**index multiple**). Chaque valeur d'index peut ne désigner qu'une et une seule ligne, on parlera alors d'**index unique** donc de **clef primaire**, dans le cas contraire on parlera d'**index dupliqué**.

$\Rightarrow$ **Remarque** : Un index est automatiquement créé lorsqu'une table est créée avec la contrainte PRIMARY KEY.

Créer / Supprimer un Index

Création

CREATE INDEX ‘index_nom‘ **ON** table;

⇒ index sur une seule colonne:

CREATE INDEX ‘index_nom‘ **ON** table (‘colonne1‘);

⇒ index sur plusieurs colonnes:

CREATE INDEX ‘index_nom‘ **ON** table (‘colonne1‘, ‘colonne2‘);

Suppression

DROP INDEX nom_index ;

Exemple de création d’un index unique

CREATE UNIQUE INDEX I-Eleve **ON** ELEVE (nom,prenom);

Modifier une base de données

INSERT INTO

Syntaxe

INSERT INTO NomTable(colonne1,colonne2,colonne3,...)
VALUES (Valeur1,Valeur2,Valeur3,...);

Insertion par une selection

INSERT INTO NomTable(colonne1,colonne2,...)
SELECT colonne1,colonne2,... FROM NomTable2
WHERE condition

$\Longrightarrow$ Les valeurs inconnues prennent la valeur **NULL**

INSERT (suite)

Exemple

INSERT INTO client (prenom, nom, ville, age)
VALUES ('ALI', 'Armand', 'Bejaia', 24),
('OMAR', 'Hebert', 'Bejaia', 36),
('IDIR', 'Ribeiro', 'Alger', 27);

Exemple - Insert avec le contenu d'une ou plusieurs tables

```
INSERT INTO etudiant_deug
SELECT * FROM etudiant
WHERE cycle = 1;
```

```
INSERT INTO etudiant_deug (nomd, prenomd, cycled)
SELECT nom,prenom,1 FROM etudiant
WHERE cycle = 1;
```

UPDATE

Syntaxe

UPDATE NomTable
SET Colonne = ValeurOuExpression
WHERE condition

Exemple

```
UPDATE employe
SET    nom = 'Michel', adresse = 'Toulouse'
WHERE idEmp = 100;
```

```
UPDATE employe
SET    salaire = salaire * 1.1
WHERE idSer = 'info';
```

DELETE

Syntaxe

DELETE FROM NomTable
WHERE condition

Exemple

DELETE FROM utilisateur
WHERE date_inscription < '2012-04-10'

Interroger une base de données

SELECT

L'instruction SELECT permet de visualiser les données stockées dans les bases et/ou d'effectuer des calculs ou des transformations sur ces données.

Syntaxe de base

SELECT [ALL|DISTINCT] NomColonne1,... | *
FROM NomTable1,...
WHERE Condition

- **ALL**: toutes les lignes, **DISTINCT**: pas de doublons, *: toutes les colonnes.
- La clause **AS**: **SELECT** Compteur **AS** Ctp **FROM** Vehicule

SELECT

Exemples

- **SELECT * FROM** Articles;
- **SELECT** code,reference **FROM** Articles
- **SELECT** numero, nom, prenom, ville **FROM** Clients **WHERE** ville='Bejaia'
- **SELECT** nom, prenom **FROM** Clients **WHERE** codePostal **BETWEEN** 06000 **AND** 06200
- **SELECT** CAT.libelle, ART.designattion **FROM** Categorie CAT, Articles Art
- **SELECT** p_id, p_nom_fr **AS** nom, p_description_fr AS description, p_prix_euro **AS** prix **FROM** produit

SELECT

Restriction

Les conditions peuvent faire appel aux opérateurs suivants:

- Opérateurs logiques : **AND**, **OR**, **NOT**
- Comparateurs de chaînes : **IN**, **BETWEEN**, **LIKE**
- Opérateurs arithmétiques: $+$, $-$, $/$, $\%$
- Comparateurs arithmétiques: $=$, $\neq$, $<$, $>$, $\geq$, $\leq$, <>

Restriction - Exemple

SELECT * **FROM** Vehicules **WHERE** Marque **LIKE** 'e%'

2⇒ **SELECT** * **FROM** Vehicules **WHERE** (Compteur>10000) **AND** (Compteur<=30000)
3⇒ **WHERE** Compteur **BETWEEN** 10000 **AND** 30000
4⇒ **WHERE** Marque **IN** ('Peugeot','Citroën')

SELECT - Différentes clauses

..

SELECT *
FROM table
WHERE condition
GROUP BY expression
HAVING condition
{ **UNION** | **INTERSECT** | **EXCEPT** }
ORDER BY expression
LIMIT count

GROUP BY

⇒ **GROUP BY** est utilisé pour grouper plusieurs résultats sur un groupe de résultat.

Exemple - GROUP BY

SELECT client, **SUM**(tarif)
FROM achat
GROUP BY client

⇒ Sur une table qui contient toutes les ventes d'un magasin, il est possible de regrouper les ventes par clients identiques et d'obtenir le coût total des achats pour chaque client.

GROUP BY - Exemple

⇒ **SELECT** client, **SUM**(tarif) **FROM** achat **GROUP BY** client

id	client	tarif	date
1	Pierre	102	2012-10-23
2	Simon	47	2012-10-27
3	Marie	18	2012-11-05
4	Marie	20	2012-11-14
5	Pierre	160	2012-12-03

Résultats :

client	SUM(tarif)
Pierre	262
Simon	47
Marie	38

HAVING

⇒ **HAVING** est presque similaire à **WHERE** à la seule différence que **HAVING** permet de filtrer en utilisant des fonctions telles que **SUM()**, **COUNT()**, **AVG()**, **MIN()** ou **MAX()**.

Exemple - HAVING

SELECT client, **SUM**(tarif)
FROM achat
GROUP BY client
HAVING SUM(tarif) $>$ 40

HAVING - Exemple

⇒ **SELECT** client, **SUM**(tarif) **FROM** achat **GROUP BY** client **HAVING SUM**(tarif) > 40

id	client	tarif	date_achat
1	Pierre	102	2012-10-23
2	Simon	47	2012-10-27
3	Marie	18	2012-11-05
4	Marie	20	2012-11-14
5	Pierre	160	2012-12-03

Résultat :

client	SUM(tarif)
Pierre	162
Simon	47

ORDER BY

⇒ **ORDER BY** permet de trier les lignes dans un résultat d'une requête SQL. Il est possible de trier les données sur une ou plusieurs colonnes, par ordre ascendant ou descendant (**ASC**|**DESC**).

Exemples - ORDER BY

SELECT *
FROM utilisateur
ORDER BY nom **DESC**

ORDER BY - Exemple

⇒ **SELECT * FROM** utilisateur **ORDER BY** nom **DESC**

id	nom	prenom	date_inscription	tarif_total
1	Durand	Maurice	2012-02-05	145
2	Dupond	Fabrice	2012-02-07	65
3	Durand	Fabienne	2012-02-13	90
4	Dubois	Chloé	2012-02-16	98
5	Dubois	Simon	2012-02-23	27

Résultat :

id	nom	prenom	date_inscription	tarif_total
4	Dubois	Chloé	2012-02-16	98
5	Dubois	Simon	2012-02-23	27
2	Dupond	Fabrice	2012-02-07	65
1	Durand	Maurice	2012-02-05	145
3	Durand	Fabienne	2012-02-13	90

CASE

⇒ **CASE** ... **WHEN** ... permet d'utiliser des conditions de type « si / sinon » pour retourner un résultat disponible entre plusieurs possibilités. Le **CASE** peut être utilisé dans n'importe quelle instruction ou clause, telle que **SELECT**, **UPDATE**, **DELETE**, **WHERE**, **ORDER BY** ou **HAVING**.

Exemples - CASE

```
SELECT id, nom, surcharge, prix_unitaire, quantite,
CASE
  WHEN surcharge=1 THEN 'Prix ordinaire'
  WHEN surcharge>1 THEN 'Prix supérieur à la normale'
ELSE 'Prix inférieur à la normale'
END
FROM achat
```

CASE - Exemple

⇒ SELECT id, nom, surcharge, prix_unitaire, quantite,
CASE
  WHEN surcharge=1 THEN 'Prix ordinaire'
  WHEN surcharge>1 THEN 'Prix supérieur à la normale'
ELSE 'Prix inférieur à la normale'
END

FROM achat

Table « achat » :

id	nom	surcharge	prix_unitaire	quantite
1	Produit A	1.3	6	3
2	Produit B	1.5	8	2
3	Produit C	0.75	7	4
4	Produit D	1	15	2

Résultat :

id	nom	surcharge	prix_unitaire	quantite	CASE
1	Produit A	1.3	6	3	Prix supérieur à la normale
2	Produit B	1.5	8	2	Prix supérieur à la normale
3	Produit C	0.75	7	4	Prix inférieur à la normale
4	Produit D	1	15	2	Prix ordinaire

UNION

⇒ **UNION** permet de mettre bout-à-bout les résultats de plusieurs requêtes utilisant elles-même la commande **SELECT**. Elle permet, donc, de concaténer les résultats de 2 requêtes ou plus. Pour l'utiliser il est nécessaire que chacune des requêtes à concaténer retournes le même nombre de colonnes, avec les mêmes types de données et dans le même ordre.

Exemples - UNION

```
SELECT * FROM magasin1_client
UNION
SELECT * FROM magasin2_client
```

UNION - Exemple

⇒ **SELECT * FROM magasin1_client UNION SELECT * FROM magasin2_client**

magasin1_client

prenom	nom	ville	date_naissance	total_achat
Léon	Dupuis	Paris	1983-03-06	135
Marie	Bernard	Paris	1993-07-03	75
Sophie	Dupond	Marseille	1986-02-22	27
Marcel	Martin	Paris	1976-11-24	39

magasin2_client

prenom	nom	ville	date_naissance	total_achat
Marion	Leroy	Lyon	1982-10-27	285
Paul	Moreau	Lyon	1976-04-19	133
Marie	Bernard	Paris	1993-07-03	75
Marcel	Martin	Paris	1976-11-24	39

Résultat :

prenom	nom	ville	date_naissance	total_achat
Léon	Dupuis	Paris	1983-03-06	135
Marie	Bernard	Paris	1993-07-03	75
Sophie	Dupond	Marseille	1986-02-22	27
Marcel	Martin	Paris	1976-11-24	39
Marion	Leroy	Lyon	1982-10-27	285
Paul	Moreau	Lyon	1976-04-19	133

INTERSECT

⇒ **INTERSECT** permet d'obtenir l'intersection des résultats de 2 requêtes (récupérer les enregistrements communs à 2 requêtes). Cela permet de trouver s'il y a des données similaires sur 2 tables distinctes.

Exemples - INTERSECT

SELECT * **FROM** magasin1_client
INTERSECT
SELECT * **FROM** magasin2_client

INTERSECT - Exemple

⇒ **SELECT * FROM magasin1_client INTERSECT SELECT * FROM magasin2_client**

magasin1_client

prenom	nom	ville	date_naissance	total_achat
Léon	Dupuis	Paris	1983-03-06	135
Marie	Bernard	Paris	1993-07-03	75
Sophie	Dupond	Marseille	1986-02-22	27
Marcel	Martin	Paris	1976-11-24	39

magasin2_client

prenom	nom	ville	date_naissance	total_achat
Marion	Leroy	Lyon	1982-10-27	285
Paul	Moreau	Lyon	1976-04-19	133
Marie	Bernard	Paris	1993-07-03	75
Marcel	Martin	Paris	1976-11-24	39

Résultat :

prenom	nom	ville	date_naissance	total_achat
Marie	Bernard	Paris	1993-07-03	75
Marcel	Martin	Paris	1976-11-24	39

SELECT - Jointure SQL

Types de jointures

1. **INNER JOIN:**
2. **CROSS JOIN:**
3. **LEFT JOIN:**
4. **RIGHT JOIN:**
5. **FULL JOIN:**
6. **SELF JOIN:**
7. **NATURAL JOIN:**
8. **UNION JOIN:**

INNER JOIN

⇒ INNER JOIN

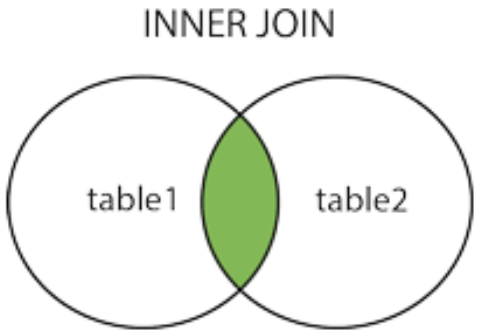


Figure: **INNER JOIN** retourne les enregistrements lorsqu'il y a au moins une ligne dans chaque colonne qui correspond à la condition.

■ Exemples - INNER JOIN

SELECT id, prenom, nom, date_achat, num_facture, prix_total
FROM utilisateur
INNER JOIN commande **ON** utilisateur.id = commande.utilisateur_id

INNER JOIN - Exemple

Table utilisateur :

id	prenom	nom	email	ville
1	Aimée	Marechal	aime.marechal@example.com	Paris
2	Esmée	Lefort	esmee.lefort@example.com	Lyon
3	Marine	Prevost	m.prevost@example.com	Lille
4	Luc	Rolland	lucrolland@example.com	Marseille

Table commande :

utilisateur_id	date_achat	num_facture	prix_total
1	2013-01-23	A00103	203.14
1	2013-02-14	A00104	124.00
2	2013-02-17	A00105	149.45
2	2013-02-21	A00106	235.35
5	2013-03-02	A00107	47.58

Résultats :

id	prenom	nom	date_achat	num_facture	prix_total
1	Aimée	Marechal	2013-01-23	A00103	203.14
1	Aimée	Marechal	2013-02-14	A00104	124.00
2	Esmée	Lefort	2013-02-17	A00105	149.45
2	Esmée	Lefort	2013-02-21	A00106	235.35

CROSS JOIN

⇒ CROSS JOIN

permet de retourner le produit cartésien entre deux tables. Cela permet de retourner chaque ligne d'une table avec chaque ligne d'une autre table. Ainsi effectuer le produit cartésien d'une table ***A*** qui contient 10 résultats avec une table ***B*** de 20 résultats va produire 200 résultats (10 x 20 = 200).

■ Exemples - CROSS JOIN

```
SELECT l_id, l_nom_fr_fr, f_id, f_nom_fr_fr
FROM legume
CROSS JOIN fruit
```

CROSS JOIN - Exemple

Table legume :

l_id	l_nom_fr_fr	l_nom_en_gb
45	Carotte	Carott
46	Oignon	Onion
47	Poireau	Leek

Table fruit :

f_id	f_nom_fr_fr	f_nom_en_gb
87	Banane	Banana
88	Kiwi	Kiwi
89	Poire	Pear

Résultats :

l_id	l_nom_fr_fr	f_id	f_nom_fr_fr
45	Carotte	87	Banane
45	Carotte	88	Kiwi
45	Carotte	89	Poire
46	Oignon	87	Banane
46	Oignon	88	Kiwi
46	Oignon	89	Poire
47	Poireau	87	Banane
47	Poireau	88	Kiwi
47	Poireau	89	Poire

LEFT JOIN

⇒ LEFT JOIN

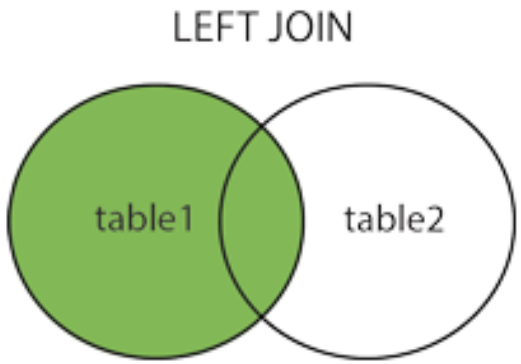


Figure: **LEFT JOIN** (aussi appelée **LEFT OUTER JOIN**) – *jointure entre 2 tables* – permet de lister tous les résultats de la table de gauche (left = gauche) même s'il n'y a pas de correspondance dans la deuxième tables.

■ Exemples - LEFT JOIN

```
SELECT id, prenom, nom, date_achat, num_facture, prix_total
FROM utilisateur
LEFT JOIN commande ON utilisateur.id = commande.utilisateur_id
```

LEFT JOIN

Table utilisateur :

id	prenom	nom	email	ville
1	Aimée	Marechal	aime.marechal@example.com	Paris
2	Esmée	Lefort	esmee.lefort@example.com	Lyon
3	Marine	Prevost	m.prevost@example.com	Lille
4	Luc	Rolland	lucrolland@example.com	Marseille

Table commande :

utilisateur_id	date_achat	num_facture	prix_total
1	2013-01-23	A00103	203.14
1	2013-02-14	A00104	124.00
2	2013-02-17	A00105	149.45
2	2013-02-21	A00106	235.35
5	2013-03-02	A00107	47.58

Résultats :

id	prenom	nom	date_achat	num_facture	prix_total
1	Aimée	Marechal	2013-01-23	A00103	203.14
1	Aimée	Marechal	2013-02-14	A00104	124.00
2	Esmée	Lefort	2013-02-17	A00105	149.45
2	Esmée	Lefort	2013-02-21	A00106	235.35
3	Marine	Prevost	NULL	NULL	NULL
4	Luc	Rolland	NULL	NULL	NULL

RIGHT JOIN

⇒ RIGHT JOIN

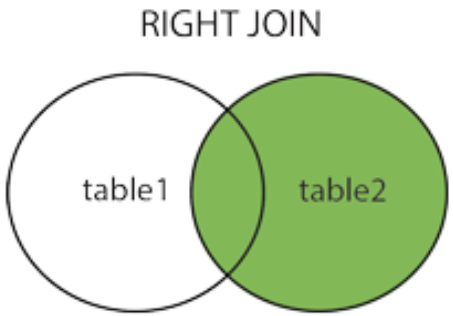


Figure: **RIGHT JOIN** (ou **RIGHT OUTER JOIN**) – *jointure entre 2 tables* – permet de retourner tous les enregistrements de la table de droite (right = droite) même s'il n'y a pas de correspondance avec la table de gauche.

■ Exemples - RIGHT JOIN

```
SELECT id, prenom, nom, utilisateur_id, date_achat, num_facture
FROM utilisateur
RIGHT JOIN commande ON utilisateur.id = commande.utilisateur_id
```

RIGHT JOIN - Exemple

Table utilisateur :

id	prenom	nom	email	ville	actif
1	Aimée	Marechal	aime.marechal@example.com	Paris	1
2	Esmée	Lefort	esmee.lefort@example.com	Lyon	0
3	Marine	Prevost	m.prevost@example.com	Lille	1
4	Luc	Rolland	lucrolland@example.com	Marseille	1

Table commande :

utilisateur_id	date_achat	num_facture	prix_total
1	2013-01-23	A00103	203.14
1	2013-02-14	A00104	124.00
2	2013-02-17	A00105	149.45
3	2013-02-21	A00106	235.35
5	2013-03-02	A00107	47.58

Résultats :

id	prenom	nom	utilisateur_id	date_achat	num_facture
1	Aimée	Marechal	1	2013-01-23	A00103
1	Aimée	Marechal	1	2013-02-14	A00104
2	Esmée	Lefort	2	2013-02-17	A00105
3	Marine	Prevost	3	2013-02-21	A00106
NULL	NULL	NULL	5	2013-03-02	A00107

FULL JOIN

⇒ FULL JOIN

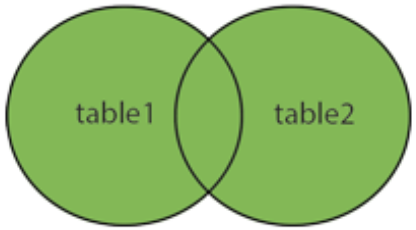


Figure: **FULL JOIN** (ou **FULL OUTER JOIN**) – *jointure entre 2 tables* – permet de combiner les résultats des 2 tables, les associer entre eux grâce à une condition et remplir avec des valeurs NULL si la condition n'est pas respectée.

■ Exemples - FULL JOIN

```
SELECT id, prenom, nom, utilisateur_id, date_achat, num_facture
FROM utilisateur
FULL JOIN commande ON utilisateur.id = commande.utilisateur_id
```

FULL JOIN - Exemple

Table utilisateur :

id	prenom	nom	email	ville	actif
1	Aimée	Marechal	aime.marechal@example.com	Paris	1
2	Esmée	Lefort	esmee.lefort@example.com	Lyon	0
3	Marine	Prevost	m.prevost@example.com	Lille	1
4	Luc	Rolland	lucrolland@example.com	Marseille	1

Table commande :

utilisateur_id	date_achat	num_facture	prix_total
1	2013-01-23	A00103	203.14
1	2013-02-14	A00104	124.00
2	2013-02-17	A00105	149.45
3	2013-02-21	A00106	235.35
5	2013-03-02	A00107	47.58

Résultat :

id	prenom	nom	utilisateur_id	date_achat	num_facture
1	Aimée	Marechal	1	2013-01-23	A00103
1	Aimée	Marechal	1	2013-02-14	A00104
2	Esmée	Lefort	2	2013-02-17	A00105
3	Marine	Prevost	3	2013-02-21	A00106
4	Luc	Rolland	NULL	NULL	NULL
NULL	NULL		5	2013-03-02	A00107

SELF JOIN

⇒ SELF JOIN

permet d'effectuer une jointure d'une table avec elle-même comme si c'était une autre table.

■ Exemples - SELF JOIN

SELECT u1.u_id, u1.u_nom, u2.u_id, u2.u_nom
FROM utilisateur as u1
LEFT OUTER JOIN utilisateur as u2 **ON** u2.u_manager_id =u1.u_id

SELF JOIN - Exemple

Table utilisateur :

id	prenom	nom	email	manager_id
1	Sebastien	Martin	s.martin@example.com	NULL
2	Gustave	Dubois	g.dubois@example.com	NULL
3	Georgette	Leroy	g.leroy@example.com	1
4	Gregory	Roux	g.roux@example.com	2

Résultat :

u1_id	u1_prenom	u1_nom	u1_email	u1_manager_id	u2_prenom	u2_nom
1	Sebastien	Martin	s.martin@example.com	NULL	NULL	NULL
2	Gustave	Dubois	g.dubois@example.com	NULL	NULL	NULL
3	Georgette	Leroy	g.leroy@example.com	1	Sebastien	Martin
4	Gregory	Roux	g.roux@example.com	2	Gustave	Dubois

NATURAL JOIN

⇒ **NATURAL JOIN** – *jointure entre 2 tables* – permet de faire une jointure à la condition qu'il y ai des colonnes du même nom et de même type dans les 2 tables. Le résultat est la création d'un tableau avec autant de lignes qu'il y a de paires correspondant à l'association des colonnes de même nom.

Exemples - FULL JOIN

```
SELECT pays_id, user_id, user_prenom, user_ville, pays_nom
FROM utilisateur
NATURAL JOIN pays
```

NATURAL JOIN - Exemple

Table « utilisateur » :

user_id	user_prenom	user_ville	pays_id
1	Jérémie	Paris	1
2	Damien	Lyon	2
3	Sophie	Marseille	NULL
4	Yann	Lille	9999
5	Léa	Paris	1

Table « pays » :

pays_id	pays_nom
1	France
2	Canada
3	Belgique
4	Suisse

Résultat :

pays_id	user_id	user_prenom	user_ville	pays_nom
1	1	Jérémie	Paris	France
2	2	Damien	Lyon	Canada
NULL	3	Sophie	Marseille	NULL
9999	4	Yann	Lille	NULL
1	5	Léa	Paris	France

SELECT - Requêtes imbriquées

Exemple 1

```
SELECT * FROM `table`
WHERE `nom_colonne` = (
    SELECT `valeur`
    FROM `table2`
    LIMIT 1
)
```

Exemple 2

```
SELECT * FROM `table`
WHERE `nom_colonne` IN (
    SELECT `colonne`
    FROM `table2`
    WHERE `cle_etrangere` = 36
)
```

SELECT - avec EXISTS

⇒ **EXISTS** s'utilise dans une clause conditionnelle pour savoir s'il y a une présence ou non de lignes lors de l'utilisation d'une sous-requête.

Exemple

```
SELECT nom_colonne1
FROM 'table1'
WHERE EXISTS (
    SELECT nom_colonne2
    FROM 'table2'
    WHERE nom_colonne3 = 10
)
```

EXISTS - Exemple

⇒ SELECT * FROM commande WHERE EXISTS (
SELECT * FROM produit WHERE c_produit_id = p_id

);

Table commande :

c_id	c_date_achat	c_produit_id	c_quantite_produit
1	2014-01-08	2	1
2	2014-01-24	3	2
3	2014-02-14	8	1
4	2014-03-23	10	1

Table produit :

p_id	p_nom	p_date_ajout	p_prix
2	Ordinateur	2013-11-17	799.9
3	Clavier	2013-11-27	49.9
4	Souris	2013-12-04	15
5	Ecran	2013-12-15	250

Résultat :

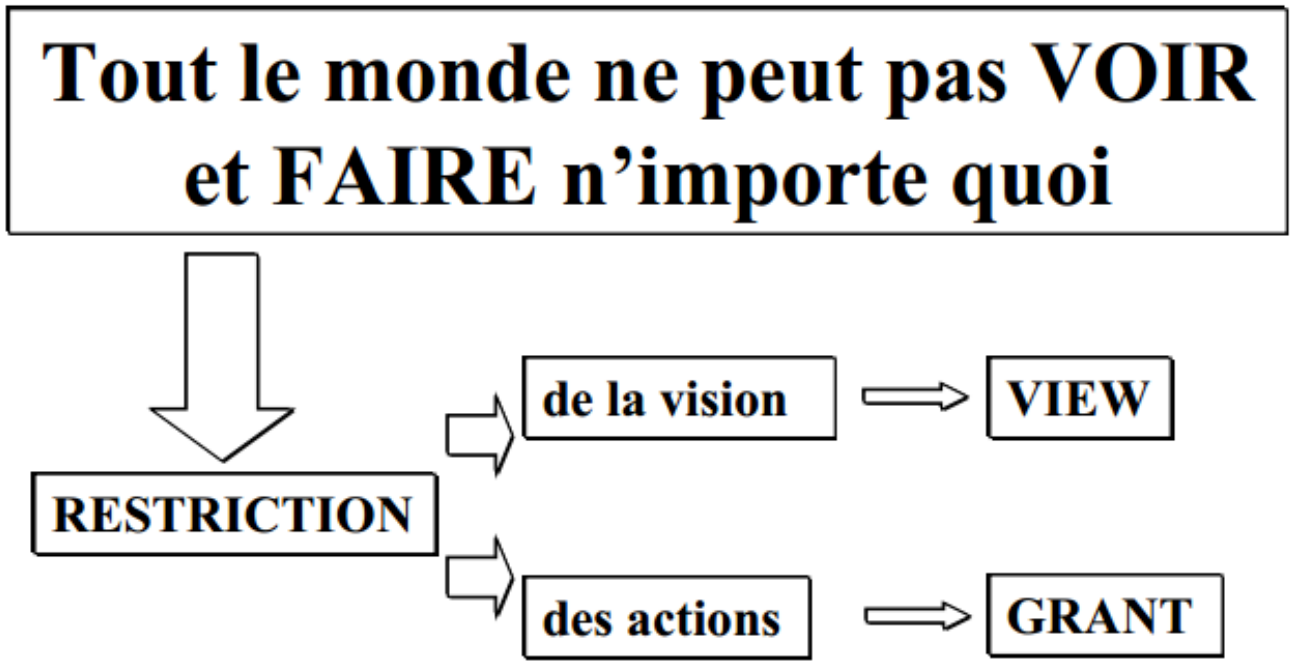
c_id	c_date_achat	c_produit_id	c_quantite_produit
1	2014-01-08	2	1
2	2014-01-24	3	2

Contrôle des Données

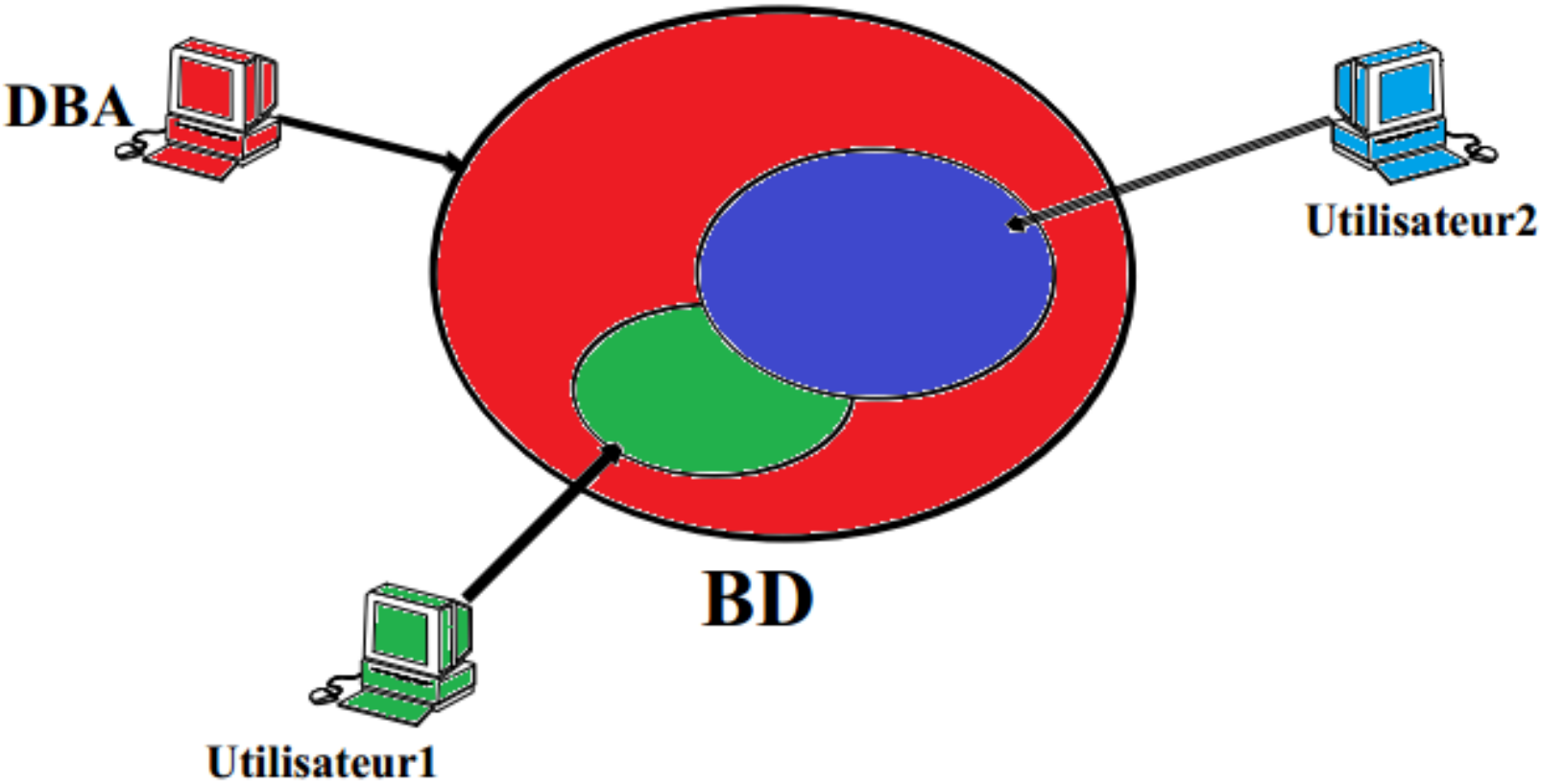
Contrôle des Données

1. **Notion de Sous–Schéma**
   - Restriction de la vision
   - Restriction des actions
2. **Privilèges**
   - Système
   - Objet
3. **Rôles**
   - Regroupement de privilèges
4. **Contraintes évènementielles: Déclencheur**
   - Contrôles avant une modification
   - Mises à jour automatique (avant/après une modification)

Restreindre les accès à une BDD



Restriction des accès - Sous Schéma / Schéma Externe



L'objet **VUE**

1. Une VUE est une table virtuelle : aucune implémentation physique de ses données.
2. La définition de la vue est enregistrée dans le DD
3. Une vue est modifiable s'il est possible d'insérer et de supprimer des tuples dans la base au-travers de la vue.
4. Vue mono-table: crée à partir d'une table
5. Vue multi-table: crée à partir de plusieurs tables

Utilisation d'une **VUE**

1. Simplification de requêtes pour des non spécialistes
2. Création de résultats intermédiaires pour des requêtes complexes
3. Présentation différente de la base de données : schéma externe
4. Mise en place de la confidentialité (éviter de divulguer certaines informations)
5. Assurer l'indépendance du schéma externe

Création et suppression d'une **VUE**

Création

CREATE VIEW NomVue(NomColonne1,...)
AS SELECT NomColonne1,..
FROM NomTable
WHERE Condition
[WITH CHECK OPTION]

Suppression

DROP VIEW nom_vue;

RENAME

RENAME VIEW nom_vue TO nouveau_nom;

Exemples de Création de **VUES** (1)

Exemple de Création - Vue mono-table avec restriction horizontale

```
CREATE VIEW enseignant_info AS
SELECT * FROM enseignant
WHERE idDip IN (
   SELECT idDip FROM diplome
   WHERE UPPER(nomDiplome) LIKE '%INFO%');
```

Exemple de Création - Vue mono-table avec restriction verticale

```
CREATE VIEW etudiant_scol AS
SELECT idEtu, nomEtu, adrEtu, idDip FROM etudiant;
```

Exemples de Création de **VUES** (2)

Exemple de Création - Vue mono-table avec restriction mixte

```
CREATE VIEW etudiant_info (numEtudiant, nomEtudiant, adrEtudiant,
dip) AS
SELECT idEtu, nomEtu, adrEtu, idDip
FROM etudiant
WHERE idDip IN
    (SELECT idDip FROM diplome
    WHERE UPPER(nomDiplome) LIKE '%INFO%');
```

Exemple de Création - Vue mono-table avec groupage

```
CREATE VIEW emp_service (ns, nombreEmp, moyenneSal) AS
SELECT idService,COUNT(*),AVG(sal) FROM employe
GROUP BY idService;
```

Exemples d'utilisation de **VUES** (3)

Utilisation de la vue

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

Exemples de **Vues** multi-tables

Exemple 1

```
CREATE VIEW emp_ser(nom_service, nom_employe) AS
SELECT s.noms, e.nome
FROM emp e, service s
WHERE e.idSer=s.idSer;
```

Exemple 2

```
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;
```

Restriction des Actions: Les Privilèges

1. **Types de Privilèges**:
   - **Système**: droit global d'exécuter un type d'ordre SQL.
   - **Objet**: droit d'exécuter un type d'action (lecture, mise à jour...) sur un objet précis
2. **Objectif**: Contrôler l'accès à la base de données
3. **Niveaux**:
   1. **Sécurité système** : couvre l'accès à la base de données et son utilisation au niveau du système (nom de l'utilisateur et mot de passe, opérations système autorisées par/pour l'utilisateur)
   2. **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.

Privilèges système

- Plusieurs privilèges système:
  1. Création d'utilisateurs (**CREATE USER**)
  2. Création de table (**CREATE TABLE**)
  3. Suppression de table (**DROP ANY TABLE**)
  4. Sauvegarde des tables (**BACKUP ANY TABLE**)
  5. .....
- Les privilèges peuvent être regroupés dans des rôles (voir plus loin)

Délégation et Suppression de privilèges système

Délégation: GRANT

GRANT privilèges **TO** utilisateur | role

Suppression: REVOKE

REVOKE privilèges **FROM** utilisateur | role

Exemple de délégation et de suppression de privilèges

Système

- Délégation: **GRANT**

GRANT CREATE SESSION, CREATE TABLE, DROP ANY TABLE TO utilisateur1;

- Suppression: **REVOKE**

REVOKE DROP ANY TABLE FROM utilisateur1;

Exemples de **Privilèges système**

- Exemple - >Les privilèges d'un utilisateur *user1*

Privilèges	ALTER	CREATE	DROP
TABLE		X	
USER	X	X	X
VIEW		X	
.....			

Privilèges Objet

1. Contrôler les actions sur les objets
   - **Objets**: tables, vues, ...
   - **Actions**: update, insert, ...
2. Le propriétaire ('owner') peut donner ces privilèges sur ses propres objets

$\Rightarrow$ Par défaut, un utilisateur a tous les privilèges objet sur les objets qui lui appartiennent, les autres aucun (sauf DBA)

Privilèges Objet – **Délégation** et **Suppression**

- Délégation: **GRANT**
  - **GRANT** Privilège **ON** objet **TO** utilisateur | role
  - **GRANT** Privilège **ON** objet(att1,..., attn) **TO** utilisateur | role
- Suppression: **REVOKE**
  - **REVOKE** Privilège **ON** objet **FROM** utilisateur | role
  - **REVOKE** Privilège **ON** obj(att1, ...,attn) **FROM** utilisateur | role

Exemples de délégation et de suppression de privilèges objet

- Exemple: **GRANT** et **REVOKE**
  - **GRANT INSERT,UPDATE** (adr,tel) **ON** etud_info **TO** Martine, Nicole;
  - **GRANT DELETE, UPDATE , INSERT ON** etud_info **TO** Michel;
  - **REVOKE UPDATE** (tel) **ON** etud_info **FROM** Nicole;

Les privilèges Objet: Exemple **Objets** et **Actions** possibles

- Exemple -> Privilèges objet pour un utilisateur *user1*

Objets / Actions	Table	Vue
ALTER		
DELETE	X	X
INDEX	X	
INSERT	X	X
REFERENCES	X	
SELECT	X	X
UPDATE	X	X

Vues et Contrôle d'accès (Exercice)

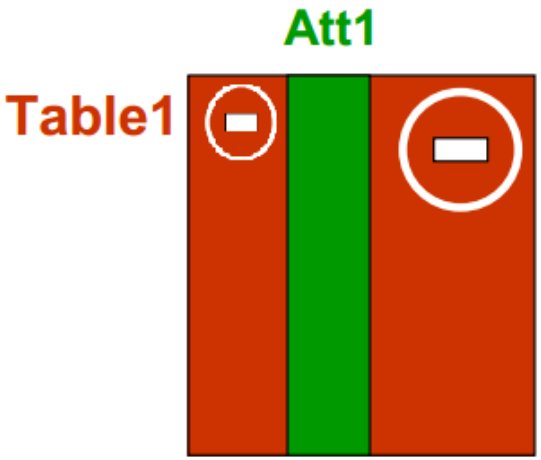


Figure: ⇒ **Question**: Comment contrôler l'accès sur une colonne d'une table? (de deux manières différentes, puis, quelle est la différence entre les deux?)

Vues et Contrôle d'accès (Exercice)

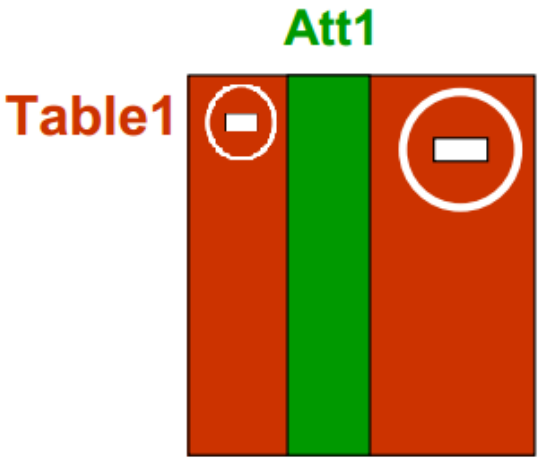


Figure: ⇒ **Question**: Comment contrôler l'accès sur une colonne d'une table? (de deux manières différentes, puis, quelle est la différence entre les deux?)

1. **GRANT** sur attributs: **GRANT SELECT ON** Table1 (Att1) **TO** PUBLIC;

Vues et Contrôle d'accès (Exercice)

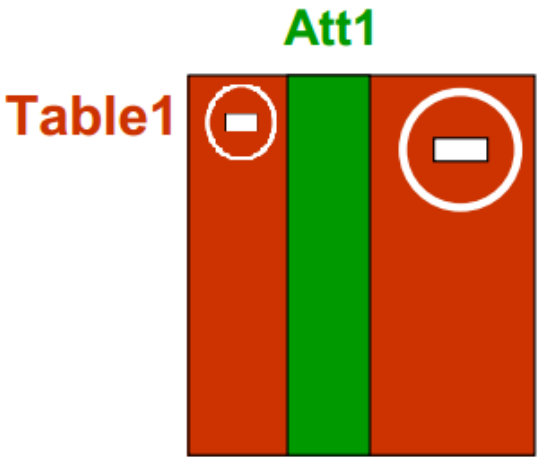


Figure: ⇒ **Question**: Comment contrôler l'accès sur une colonne d'une table? (de deux manières différentes, puis, quelle est la différence entre les deux?)

1. **GRANT** sur attributs: **GRANT SELECT ON** Table1 (Att1) **TO** PUBLIC;
2. **GRANT** sur vue:
   - **CREATE VIEW** S_ATT1 **AS SELECT** Attr1 **FROM** Table1;

Vues et Contrôle d'accès (Exercice)

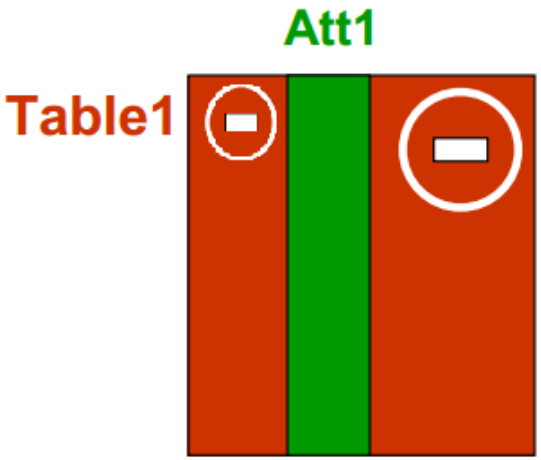


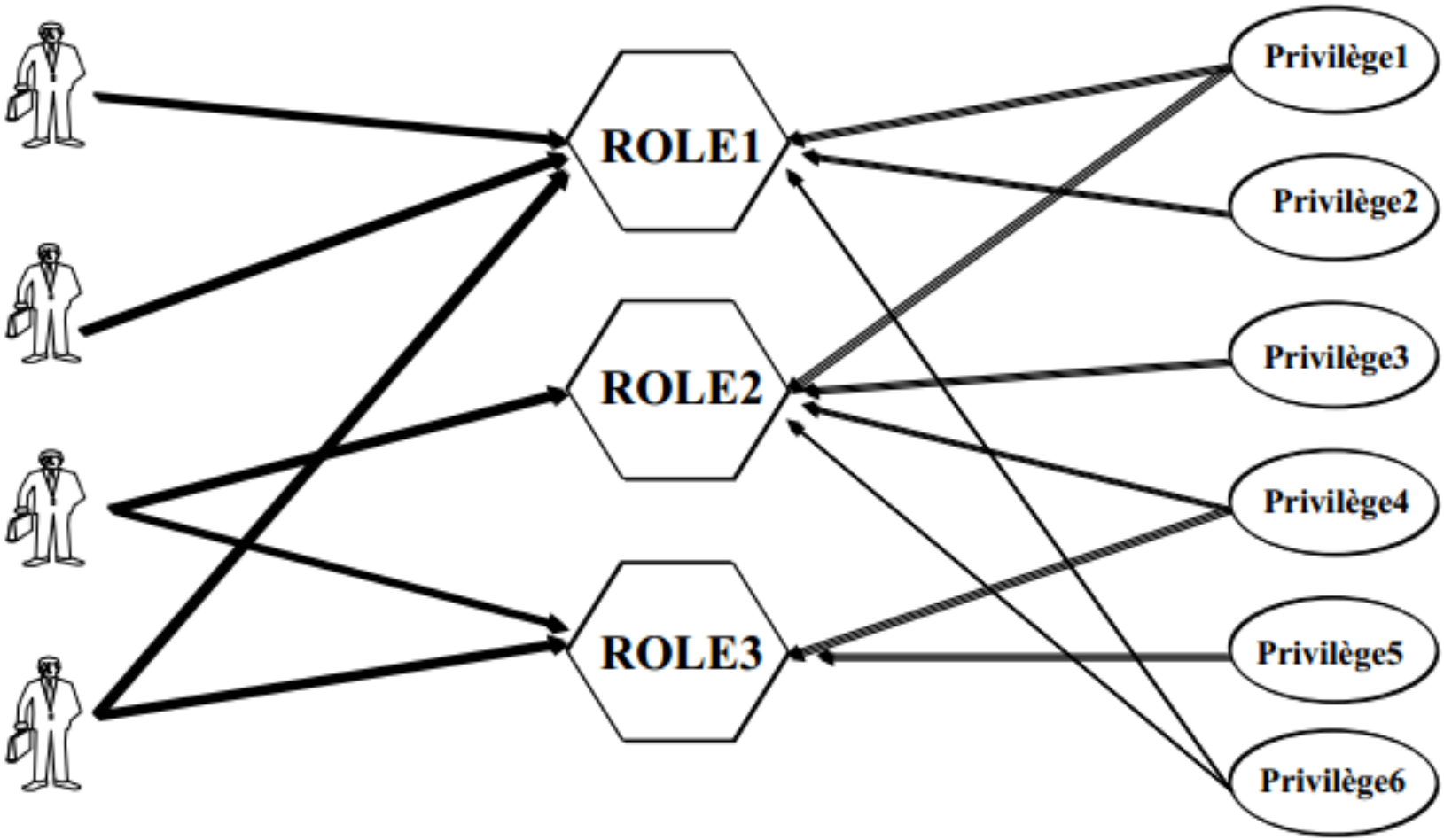
Figure: ⇒ **Question**: Comment contrôler l'accès sur une colonne d'une table? (de deux manières différentes, puis, quelle est la différence entre les deux?)

1. **GRANT** sur attributs: **GRANT SELECT ON** Table1 (Att1) **TO** PUBLIC;
2. **GRANT** sur vue:
   - **CREATE VIEW** S_ATT1 **AS SELECT** Attr1 **FROM** Table1;
   - **GRANT SELECT ON** S_ATT1 **TO** PUBLIC;

Les Rôles

1. Regroupement d'utilisateurs qui partagent les mêmes privilèges
2. Facilitent la gestion des autorisations des privilèges objet en évitant les ordres **GRANT**
   - Association d'un rôle à chaque utilisateur
   - Attribution de privilèges à chaque rôle
3. Un rôle par défaut est donné à un utilisateur
4. Un utilisateur peut posséder plusieurs rôles

Les Rôles


ROLE1
ROLE2
ROLE3
Privilège1
Privilège2
Privilège3
Privilège4
Privilège5
Privilège6

Manipulation des rôles: Ordres

Création d'un utilisateur

CREATE USER Nom_utilisateur;

Création / Modification d'un rôle

{**CREATE**|**ALTER**} **ROLE** nom_rôle { **NOT IDENTIFIED** | **IDENTIFIED** {**BY** mot_de_passe | **EXTERNALLY**}};

Manipulation des rôles: Ordres (suite)

Remplissage d'un rôle

GRANT {privilège1 | rôle1} **TO** nom_rôle;
GRANT {privilège2 | rôle2} **TO** nom_rôle;

Attribution d'un rôle

GRANT ROLE nom_role **TO** user;

Suppression / Révocation d'un rôle

DROP ROLE nom_rôle;
REVOKE ROLE nom_rôle **FROM** user;

Exemples: Manipulation des rôles

Création / Attribution

1. **CREATE ROLE** secretariat_info ;
2. **GRANT SELECT,UPDATE** (adr,tel) **ON** ens_info **TO** secretariat_info;
3. **GRANT SELECT,INSERT,UPDATE ON** etud_info **TO** secretariat_info;
4. **GRANT SELECT,INSERT ON** cours_info **TO** secretariat_info;
5. **GRANT** secretariat_info **TO** Ali, Farid, Mohamed;

⇒ **Question**: Quels sont les privilèges de l'utilisateur *Mohamed*?

Exemples: Manipulation des rôles (suite)

Création d'un rôle et octroi de privilèges à ce rôle

CREATE ROLE privilegesEtudiant ;
GRANT CREATE TABLE, CREATE VIEW TO privilegesEtudiant;

Création des utilisateurs et octroi du rôle

CREATE USER Etudiant1;
GRANT privilegesEtudiant **TO** Etudiant1;
CREATE USER Etudiant2;
GRANT privilegesEtudiant **TO** Etudiant2;

Les Déclencheurs

Les **Déclencheurs** - Définition

- Un déclencheur (**trigger** en anglais) est un dispositif logiciel qui provoque un traitement particulier en fonction d'événements prédéfinis.
- Associer l'exécution d'une procédure stockée à l'exécution d'une instruction.
- Règle E – C – A : Évènement – Condition – Action.
- Evénement: **INSERT**, **UPDATE**, **DELETE**.

Les **Déclencheurs** - Utilisation

- L'objectif des déclencheurs est de suivre l'évolution de la base pour réaliser au mieux les différentes tâches administratives.
- Les déclencheurs peuvent servir à vérifier des contraintes que l'on ne peut pas définir de façon déclarative.
- Mise à jour d'informations qui dépendent d'autres données (Ils peuvent servir à collecter des informations sur les mises-à-jour de la base)
- Ils peuvent aussi gérer de la redondance d'information.

Les **Déclencheurs** - Utilisation

- L'objectif des déclencheurs est de suivre l'évolution de la base pour réaliser au mieux les différentes tâches administratives.
- Les déclencheurs peuvent servir à vérifier des contraintes que l'on ne peut pas définir de façon déclarative.
- Mise à jour d'informations qui dépendent d'autres données (Ils peuvent servir à collecter des informations sur les mises-à-jour de la base)
- Ils peuvent aussi gérer de la redondance d'information.

$\Rightarrow$ **Question**: Quelle est la différence entre **CHECK** et **TRIGGER**?

Les **Déclencheurs** - Syntaxe

Syntaxe -

CREATE TRIGGER nom
// événement (avec paramètres)
{**BEFORE** | **AFTER** | **INSTEAD OF**}
{**INSERT** | **DELETE** | **UPDATE** [**OF** liste_de_colonnes]}
ON table
// *granularité*
[**FOR EACH ROW**]
[**REFERENCING** {**NEW**|**OLD**|**NEW_TABLE**|**OLD_TABLE**} **AS** <nom>]...
// *condition*
WHEN (<condition de recherche SQL>)
// *action*
<Procédure SQL>

Les **Déclencheurs** - Instant et Action

- **BEFORE**: Exécution avant modification des données.
- **AFTER**: Exécution après modification des données.
- **INSTEAD OF**: Exécution à la place de l'ordre SQL envoyé.
- **INSERT, UPDATE, DELETE**: Action concernée par le déclencheur.

⇒ **BEGIN** ... **END**;
⇒ **IF** ... **THEN**;

Les **Déclencheurs** - Référencement et Granularité

- **REFERENCING NEW AS** <nom>: permet de créer une variable pour faire référence à la nouvelle ligne.

Les **Déclencheurs** - Référencement et Granularité

- **REFERENCING NEW AS** <nom>: permet de créer une variable pour faire référence à la nouvelle ligne.
- **REFERENCING OLD AS** <nom>: permet de créer une variable pour faire référence à l'ancienne ligne.

Les **Déclencheurs** - Référencement et Granularité

- **REFERENCING NEW AS** <nom>: permet de créer une variable pour faire référence à la nouvelle ligne.
- **REFERENCING OLD AS** <nom>: permet de créer une variable pour faire référence à l'ancienne ligne.
- Des tables différentielles contenant les valeurs avant et après l'événement sont définies par les options **NEW_TABLE** et **OLD_TABLE**.

Les **Déclencheurs** - Référencement et Granularité

- **REFERENCING NEW AS** <nom>: permet de créer une variable pour faire référence à la nouvelle ligne.
- **REFERENCING OLD AS** <nom>: permet de créer une variable pour faire référence à l'ancienne ligne.
- Des tables différentielles contenant les valeurs avant et après l'événement sont définies par les options **NEW_TABLE** et **OLD_TABLE**.
- Les déclencheurs sur **INSERT** ne peuvent que déclarer les nouvelles valeurs.

Les **Déclencheurs** - Référencement et Granularité

- **REFERENCING NEW AS** <nom>: permet de créer une variable pour faire référence à la nouvelle ligne.
- **REFERENCING OLD AS** <nom>: permet de créer une variable pour faire référence à l'ancienne ligne.
- Des tables différentielles contenant les valeurs avant et après l'événement sont définies par les options **NEW_TABLE** et **OLD_TABLE**.
- Les déclencheurs sur **INSERT** ne peuvent que déclarer les nouvelles valeurs.
- Les déclencheurs sur **DELETE** ne peuvent que déclarer les anciennes.

Les **Déclencheurs** - Référencement et Granularité

- **REFERENCING NEW AS** <nom>: permet de créer une variable pour faire référence à la nouvelle ligne.
- **REFERENCING OLD AS** <nom>: permet de créer une variable pour faire référence à l'ancienne ligne.
- Des tables différentielles contenant les valeurs avant et après l'événement sont définies par les options **NEW_TABLE** et **OLD_TABLE**.
- Les déclencheurs sur **INSERT** ne peuvent que déclarer les nouvelles valeurs.
- Les déclencheurs sur **DELETE** ne peuvent que déclarer les anciennes.
- Deux granularités sont possibles:
  - **Ligne**: l'action est exécutée pour chaque ligne modifiée satisfaisant la condition
  - **Requête**: l'action est exécutée une seule fois pour l'événement.

Les **Déclencheurs** - OLD et NEW

	OLD	**NEW**
INSERT	NULL	Valeur créée
DELETE	Valeur avant suppression	NULL
UPDATE	Valeur avant modification	Valeur après modification

Les **Déclencheurs** - Ordres sur les déclencheurs

1. **REPLACE**: Remplacement
2. **DROP TRIGGER** nom_decl: Suppression
3. Activation/Désactivation:
   - **ALTER TRIGGER** nom_decl **DISABLE**
   - **ALTER TRIGGER** nom_decl **ENABLE**

Les **Déclencheurs** - Ordre d'exécution entre plusieurs déclencheurs sur le même objet

- Pour un déclencheur sur une table de la base, il peut y avoir 4 sortes de déclencheurs possibles selon l'instant (BEFORE, AFTER) et le type (requête ou ligne).
- Ces déclencheurs se déclenchent dans l'ordre suivant :

Les **Déclencheurs** - Ordre d'exécution entre plusieurs déclencheurs sur le même objet

- Pour un déclencheur sur une table de la base, il peut y avoir 4 sortes de déclencheurs possibles selon l'instant (BEFORE, AFTER) et le type (requête ou ligne).
- Ces déclencheurs se déclenchent dans l'ordre suivant :
  1. déclencheur(s) requête **BEFORE**

Les **Déclencheurs** - Ordre d'exécution entre plusieurs déclencheurs sur le même objet

- Pour un déclencheur sur une table de la base, il peut y avoir 4 sortes de déclencheurs possibles selon l'instant (BEFORE, AFTER) et le type (requête ou ligne).
- Ces déclencheurs se déclenchent dans l'ordre suivant :
  1. déclencheur(s) requête **BEFORE**
  2. Pour chaque ligne concernée

Les **Déclencheurs** - Ordre d'exécution entre plusieurs déclencheurs sur le même objet

- Pour un déclencheur sur une table de la base, il peut y avoir 4 sortes de déclencheurs possibles selon l'instant (BEFORE, AFTER) et le type (requête ou ligne).
- Ces déclencheurs se déclenchent dans l'ordre suivant :
  1. déclencheur(s) requête **BEFORE**
  2. Pour chaque ligne concernée
     1. déclencheur(s) ligne **BEFORE**

Les **Déclencheurs** - Ordre d'exécution entre plusieurs déclencheurs sur le même objet

- Pour un déclencheur sur une table de la base, il peut y avoir 4 sortes de déclencheurs possibles selon l'instant (BEFORE, AFTER) et le type (requête ou ligne).
- Ces déclencheurs se déclenchent dans l'ordre suivant :
  1. déclencheur(s) requête **BEFORE**
  2. Pour chaque ligne concernée
     1. déclencheur(s) ligne **BEFORE**
     2. déclencheur(s) ligne **AFTER**

Les **Déclencheurs** - Ordre d'exécution entre plusieurs déclencheurs sur le même objet

- Pour un déclencheur sur une table de la base, il peut y avoir 4 sortes de déclencheurs possibles selon l'instant (BEFORE, AFTER) et le type (requête ou ligne).
- Ces déclencheurs se déclenchent dans l'ordre suivant :
  1. déclencheur(s) requête **BEFORE**
  2. Pour chaque ligne concernée
     1. déclencheur(s) ligne **BEFORE**
     2. déclencheur(s) ligne **AFTER**
  3. déclencheur(s) requête **AFTER**

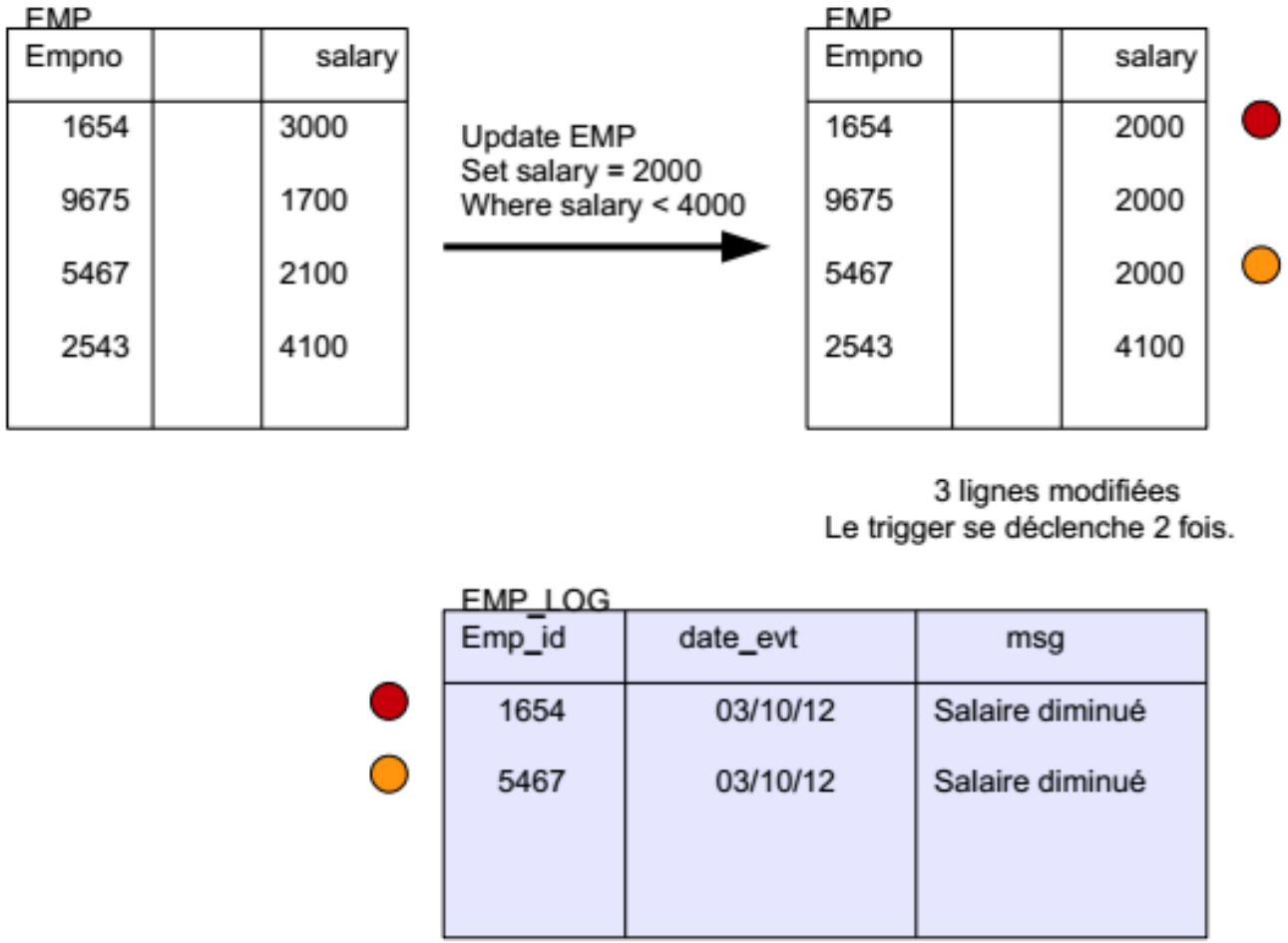
Les **Déclencheurs** - Exemple (1)

Exemple

```
CREATE OR REPLACE TRIGGER journal_emp
AFTER UPDATE OF salary ON EMPLOYEE
FOR EACH ROW
WHEN (NEW.salary < OLD.salary)
BEGIN
INSERT INTO EMP_LOG(emp_id, date_evt, msg)
VALUES (NEW.empno, sysdate, 'salaire diminué');
END;
```

Les **Déclencheurs** - Exemple (2)

Exemple



Les **Déclencheurs** - Exercice - déclencheur requête (instruction) et déclencheur ligne

$\Rightarrow$ On veut mettre à jour le nombre des commandes *nbCmdes* dans la table *Client* pour chaque client (identifié par *num_client*), à modification de la table commande (Ajout, suppression, mise-à-jour) *commande*.

$\Rightarrow$ Donner deux déclencheurs (Instruction et ligne)

Les **Déclencheurs** - Solution - Exercice – déclencheur requête (instruction) et déclencheur ligne

Déclencheur Instruction

```
CREATE OR REPLACE TRIGGER calcul_nbcmdes
AFTER INSERT OR DELETE OR UPDATE OF num_client ON
commande
BEGIN
UPDATE client SET nbCmdes = (SELECT COUNT(*) FROM
commande WHERE commande.num_client = client.num_client);
END ;
```

Les **Déclencheurs** - Solution - Exercice – déclencheur requête (instruction) et déclencheur ligne

Déclencheur ligne

```
CREATE OR REPLACE TRIGGER calcul_nbcmdes
AFTER INSERT ON commande
FOR EACH ROW
BEGIN
UPDATE client SET NEW.nbCmdes = OLD.nbCmdes+1 WHERE
num_client = new.num_client;
END ;
```

⇒ Dans le transparent précédent, le déclencheur recalcule tous les nbCmdes, même pour des clients pour qui ce nombre n'a pas changé.

⇒ Faire la meme chose pour DELETE et UPDATE

Les **Déclencheurs** -Exercice 01

$\Rightarrow$ Contrôler l'existence d'un fournisseur lors de l'ajout d'un produit. Si pas de fournisseur, annuler la transaction.

Les **Déclencheurs** -Exercice 01

⇒ Contrôler l'existence d'un fournisseur lors de l'ajout d'un produit. Si pas de fournisseur, annuler la transaction.

■ Exercice 1 - *Contrôle d'intégrité référentielle* -

```
CREATE TRIGGER InsertProduit
BEFORE INSERT ON Produit
REFERENCING NEW AS P (
  WHEN (
    NOT EXIST (
      SELECT * FROM Fournisseur WHERE IdFournisseur=P.fournisseur
    )
  ) ABORT TRANSACTION
FOR EACH ROW);
```

Les **Déclencheurs** - Exercice 02

⇒ Supprimer les produits correspondant au fournisseur supprimé (On suppose que chaque produit peut être livré par un seul fournisseur).

Les **Déclencheurs** - Exercice 02

⇒ Supprimer les produits correspondant au fournisseur supprimé (On suppose que chaque produit peut être livré par un seul fournisseur).

■ Exercice 2 - *Contrôle d'intégrité référentielle* -

CREATE TRIGGER DeleteFournisseur
BEFORE DELETE ON Fournisseur
REFERENCING OLD AS F
(**DELETE FROM** Produit **WHERE** fournisseur = F.idFournisseur
FOR EACH ROW) ;