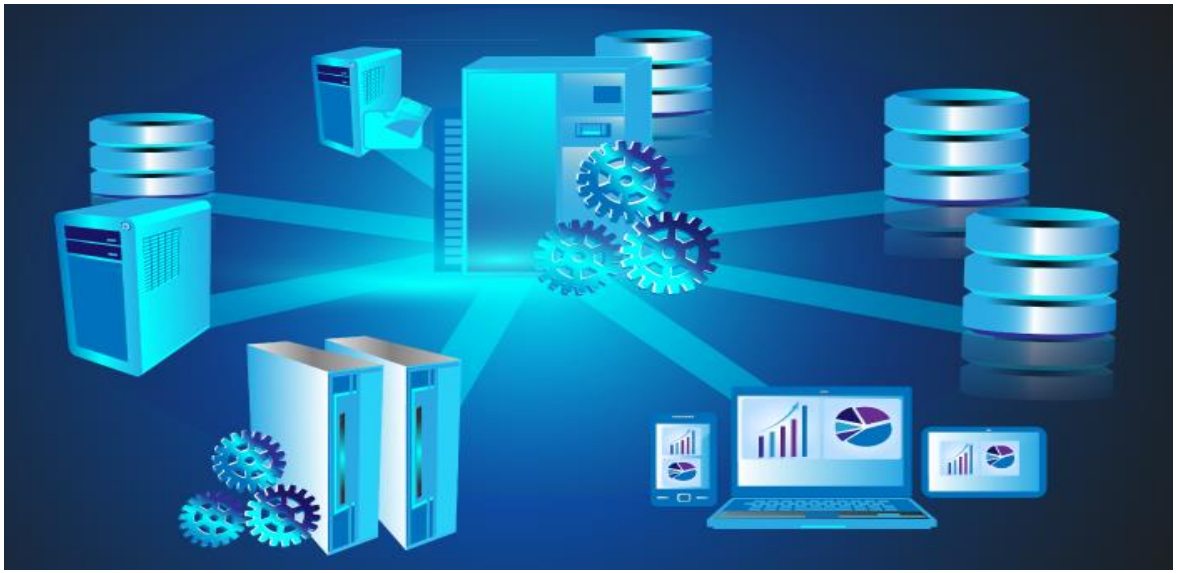
Les Bases de données avancées



Mustapha AMROUCH
Filière :GI2. EST d'Agadir
Année: 20-21

Rappel sur les cours précédents

Rappel sur SI ET BDR
Norme SQL: LDD,LMD ET LID

Cours 1 &2

Transactions
Requetes préparées

Cours 3

Procédures et fonctions stockées
Structures de contrôle
Gestions des erreurs

Cours 4

Les curseurs
Les triggers
Les vues

Cours 5

Plan de déroulement

Partie 1

- Les curseurs
- Les triggers
- Les vues

Partie 2

- Correction de TP 3 et TP 4

Plan de séance 5

- **Les curseurs**
- **Les triggers**
- **Les vues**

illustrations avec des exemples d'applications

Les curseurs

Les curseurs permettent de parcourir un jeu de résultats d'une requête SELECT, quel que soit le nombre de lignes récupérées, et d'en exploiter les valeurs.

Exemples:

- SELECT colonne(s) INTO variable(s),

Comment exploiter plusieurs lignes? SELECT… FROM ?

Gestion d'un curseur:

L'utilisation d'un curseur nécessite quatre étapes:

1— Déclaration du curseur : avec une instruction DECLARE.

2— Ouverture du curseur : on exécute la requête SELECT du curseur et on stocke le résultat dans celui-ci.

3— Parcours du curseur : on parcourt une à une les lignes.

4— Fermeture du curseur.

Les curseurs

Création et déclaration:

La déclaration d'un curseur doit se faire au début du bloc d'instructions par le mot clé DECLARE, après les variables locales et les conditions, mais avant les gestionnaires d'erreur.

```
DECLARE nom_curseur CURSOR FOR requete_select;
```

La déclaration des variables locales
La déclaration des conditions
La déclaration des curseurs
La déclaration des gestionnaires d'erreur

Exemples:

```
DECLARE curseur_client CURSOR
FOR SELECT *
FROM Client;
```

```
DECLARE curseur_entreprise CURSOR
FOR SELECT *
FROM Entreprise;
```

La déclaration permet d'associer un nom et une requête SELECT à un curseur

Les curseurs

Ouverture et fermeture:

L'ouverture du curseur va provoquer:
— l'exécution de la requête SELECT → Extraction des lignes de résultats.
La fermeture du curseur: se fait après le parcours et l'exploitation des résultats,
Si on ne le fait pas explicitement, le curseur sera fermé à la fin du bloc d'instructions.

```
OPEN nom_curseur;
-- Parcours du curseur et instructions diverses
CLOSE nom_curseur;
```

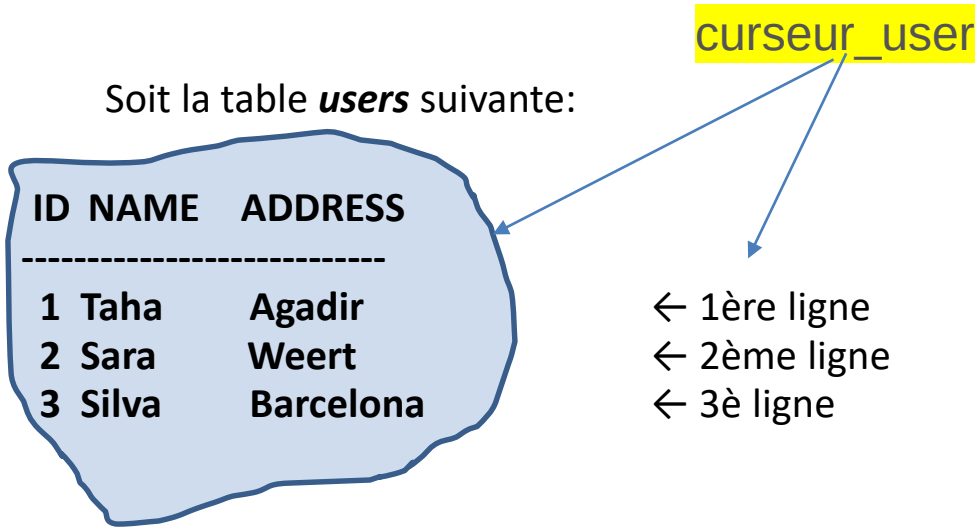
Exemples:

```
OPEN  curseur_entreprise
-- INSTRUCTIONS
CLOSE  curseur_entreprise;
```

Les curseurs

Parcours des résultats:

Une fois que le curseur a été ouvert et le jeu de résultats récupéré, le curseur place un pointeur sur la première ligne de résultats.




```
DECLARE curseur_user CURSOR
FOR SELECT *
FROM users;

OPEN  curseur_user
-- INSTRUCTIONS
CLOSE  curseur_user;
```

FETCH, récupère la ligne sur laquelle pointe le curseur, et fait avancer le pointeur vers la ligne de résultats suivante.

Les curseurs

Parcours des résultats ligne par ligne:

FETCH nom_curseur INTO variable(s);

il faut donner autant de variables dans la clause INTO qu'on a récupéré de colonnes dans la clause SELECT du curseur

curseur_user

Soit la table ***users*** suivante:

ID	NAME	ADDRESS
1	Taha	Agadir
2	Sara	Weert
3	Silva	Barcelona

← 1ère ligne
← 2ème ligne
← 3è ligne

```
DECLARE V_NAME,V_ADRESSE VARCHAR(100);
DECLARE V_ID INT;
DECLARE curseur_user CURSOR
FOR SELECT *
FROM users;

OPEN  curseur_user

FETCH curseur_user INTO V_ID,V_NAME,V_ADRESSE;
SELECT CONCAT (V_ID, ' ',V_NAME, '  ',V_ADRESSE);
```

1 Taha Agadir

```
FETCH curseur_user INTO V_ID,V_NAME,V_ADRESSE;
SELECT CONCAT (V_ID, ' ',V_NAME, '  ',V_ADRESSE);
```

2 Sara Weert

```
FETCH curseur_user INTO V_ID,V_NAME,V_ADRESSE;
SELECT CONCAT (V_ID, ' ',V_NAME, '  ',V_ADRESSE);
CLOSE  curseur_user;
```

3. Silva barcelona

Les curseurs

Parcours des résultats par une boucle:

```
DELIMITER |
CREATE PROCEDURE entreprise_boucle()
BEGIN
DECLARE v_nrue,v_rue,v_ville, v_nom_entreprise VARCHAR(100);
DECLARE v_id_entre INT;
DECLARE curseur_entreprise CURSOR
FOR SELECT *
FROM Entreprises;

OPEN  curseur_entreprise;
LOOP
FETCH curseur_entreprise INTO v_id_entre,v_nrue, v_rue, v_ville, v_nom_entreprise;
SELECT CONCAT (v_id_entre, ' ', v_nrue, ' ',v_rue, ' ' , v_nom_entreprise);
END LOOP;
CLOSE  curseur_entreprise;
END|
DELIMITER ;
Call entreprise_boucle();
```

Avec la boucle LOOP, et FETCH nous avons parcouru toutes les lignes de résultat, et l'erreur engendrée est à cause de débordement de notre curseur (application de FETCH sur aucune ligne: il n y a plus de lignes à récupérer).

Comment stopper la boucle lorsque le curseur atteint la fin des lignes de résultat?

Id_entre	nrue	rue	ville	nom_entreprise
1	7	Bisheim	casa	IBM
2	9	Honeim	stras	FORD
3	9	hay K1	Ifran	BMW
4	11	schilik	karlsruhe	Audi
6	45	Vosges	Nice	instek
7	65	place kleber	Obernai	Google

17 12:35:55 C Error Code: 1329 No data - zero rows fetched, selected, or processed

Les curseurs

Parcours des résultats par une boucle:

Pour stopper la boucle LOOP,

1) IL est nécessaire d'ajouter une instruction LEAVE pour l'arrêter.
2) En exploitant l'erreur NOT FOUND engendrée par le serveur dans ce cas; puisque on tente d'utiliser une ligne inexistante

Détails:

- ✓ Création d'un label pour la boucle,
- ✓ Utilisation d'une variable locale témoin
- ✓ Création d'un gestionnaire d'erreur de type NOT FOUND
- ✓ Une structure IF et LEAVE

Les curseurs

Parcours des résultats par une boucle:

```
DELIMITER |
CREATE PROCEDURE entreprise_boucle()
BEGIN
        DECLARE flag INT DEFAULT 0;
DECLARE v_nrue,v_rue,v_ville, v_nom_entreprise VARCHAR(100);
DECLARE v_id_entre INT;
DECLARE curseur_entreprise CURSOR
FOR SELECT *
FROM Entreprises;
        DECLARE CONTINUE HANDLER FOR NOT FOUND SET flag = 1;
OPEN  curseur_entreprise;
boucle_curseur: LOOP
        FETCH curseur_entreprise INTO v_id_entre,v_nrue, v_rue, v_ville, v_nom_entreprise;
        SELECT CONCAT (v_id_entre, ' ', v_nrue, ' ',v_rue, ' ' , v_nom_entreprise);
        IF flag = 1 THEN
        LEAVE boucle_curseur;
        END IF;
END LOOP;
CLOSE  curseur_entreprise;
END|
DELIMITER ;
Call entreprise_boucle();
```

Id_entre	nrue	rue	ville	nom_entreprise
1	7	Bisheim	casa	IBM
2	9	Honeim	stras	FORD
3	9	hay K1	Ifran	BMW
4	11	schilik	karlsruhe	Audi
6	45	Vosges	Nice	instek
7	65	place kleber	Obernai	Google

Les curseurs

Parcours des résultats par une boucle while:

Id_entre	nrue	rue	ville	nom_entreprise
1	7	Bisheim	casa	IBM
2	9	Honeim	stras	FORD
3	9	hay K1	Ifran	BMW
4	11	schilik	karlsruhe	Audi
6	45	Vosges	Nice	instek
7	65	place kleber	Obernai	Google

```
CREATE PROCEDURE entreprise_boucle2()
BEGIN
DECLARE str varchar(500) default '';
DECLARE i int default 0;
DECLARE taille int default 0;
DECLARE v_nrue,v_rue,v_ville, v_nom_entreprise VARCHAR(100);
DECLARE v_id_entre INT;
DECLARE curseur_entreprise CURSOR
FOR SELECT *
FROM Entreprise;
OPEN  curseur_entreprise;
      select FOUND_ROWS() into taille;
boucle_curseur: WHILE i<taille  DO
FETCH curseur_entreprise INTO v_id_entre,v_nrue, v_rue, v_ville, v_nom_entreprise;
SET str= CONCAT (v_id_entre, '  ', v_nrue, ' ',v_rue, ' ' , v_nom_entreprise,'*',str);
SET i=i+1;
END WHILE boucle_curseur;
SELECT str;
CLOSE  curseur_entreprise;
END|
Call entreprise_boucle2();
```

Exemple (affichage de la liste des noms des clients

```
DELIMITER $
CREATE PROCEDURE get_list_name (INOUT name_list varchar(800))
BEGIN
-- déclarer la variable pour le gestionnaire NOT FOUND.
DECLARE flag INTEGER DEFAULT 0;
-- déclarer la variable qui va contenir les noms des clients récupérer par le curseur .
DECLARE c_name varchar(100) DEFAULT "";
-- déclarer le curseur
DECLARE client_cursor CURSOR FOR
SELECT nom FROM Client;
-- déclarer le gestionnaire NOT FOUND
DECLARE CONTINUE HANDLER FOR NOT FOUND SET flag = 1;
-- ouvrir le curseur
OPEN client_cursor;
-- parcourir la liste des noms des clients et concaténer tous
-- les noms où chaque nom est séparé par un point-virgule(;)
get_list: LOOP
FETCH client_cursor INTO c_name;
IF flag= 1 THEN
LEAVE get_list;
END IF;
SET name_list = CONCAT(c_name,";",name_list);
END LOOP get_list;
-- fermer le curseur
CLOSE client_cursor;
END$
```

```
DELIMITER ;
SET @name_list ="";
CALL
get_list_name(@name_list);
SELECT @name_list;
```

Les Triggers -déclencheurs-

— Les triggers sont des objets de la base de données.
— Les triggers sont liés à une table.
— Les triggers comporte un ou plusieurs instructions SQL.
— il n'est pas possible d'appeler un trigger comme des fonctions ou des procédures stockées.
— un trigger doit être déclenché par un événement.
— Les triggers déclenchent l'exécution d'une instruction, ou une série d'instructions, lorsqu'une, ou plusieurs lignes sont insérées, supprimées ou modifiées dans la table à laquelle ils sont attachés

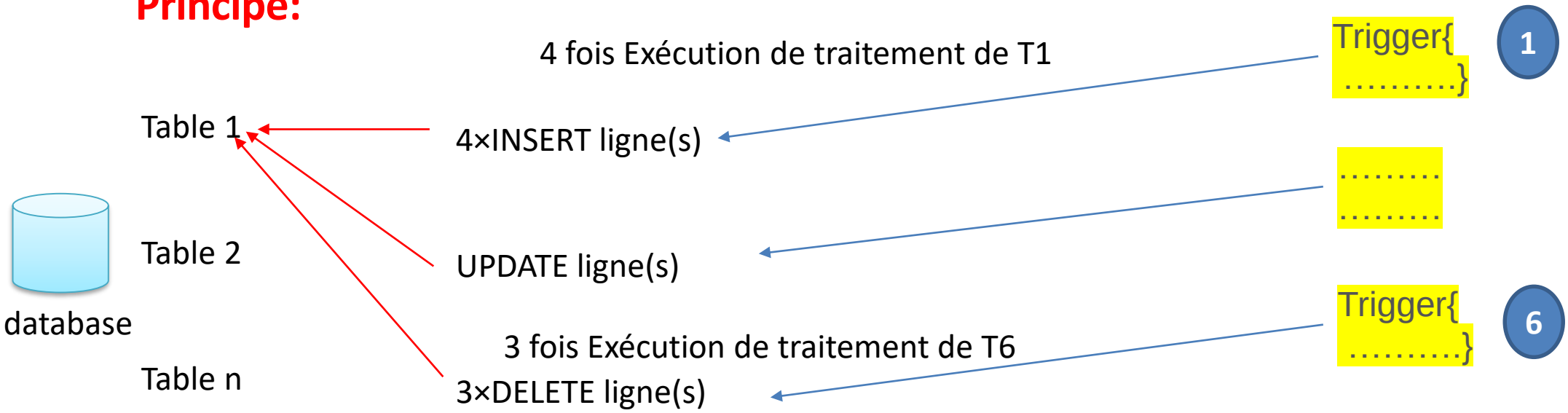
Evènements déclenchants:

— une insertion dans la table (INSERT) ;
— la suppression d'une partie des données de la table (DELETE) ;
— la modification d'une partie des données de la table (UPDATE).

une fois le trigger déclenché, ses instructions peuvent être exécutées soit juste avant l'exécution de l'événement déclencheur, soit juste après.

Les Triggers

Principe:



R1

Un trigger peut modifier et/ou insérer des données dans n'importe quelle table sauf les tables utilisées dans la requête qui l'a déclenché.

R2

Concernant la table à laquelle le trigger est attaché, ce dernier peut lire et modifier uniquement la ligne insérée, modifiée ou supprimée

Les Triggers

Création :

Pour créer un trigger, on utilise la syntaxe suivante:

```
CREATE TRIGGER nom_trigger moment_trigger {BEFORE | AFTER}
evenement_trigger{INSERT | DELETE | UPDATE}
ON nom_table FOR EACH ROW
corps_trigger
```

— moment_trigger : servent à définir quand est déclenché.
(ses instructions peuvent être exécutées à deux moments différents BEFORE |AFTER)
— evenement_trigger: spécification de l'événement déclenchant le trigger.
— ON nom_table. : c'est là qu'on définit à quelle table le trigger est attaché.
— FOR EACH ROW : pour chaque ligne insérée/supprimée/modifiée" selon l'événement qui a déclenché le trigger.
— corps_trigger : c'est le contenu du trigger..

On peut créer maximum 6 triggers par table pour couvrir chaque événement.

Les Triggers

Exemples:

T1
```
CREATE TRIGGER before_insert_entreprise BEFORE INSERT
ON entreprise FOR EACH ROW
corps_trigger;
```

T2
```
CREATE TRIGGER after_insert_entreprise AFTER INSERT
ON entreprise FOR EACH ROW
corps_trigger;
```

T3
```
CREATE TRIGGER before_update_entreprise BEFORE UPDATE ON entreprise
FOR EACH ROW
corps_trigger;
```

T4
```
CREATE TRIGGER after_update_entreprise AFTER UPDATE ON entreprise
FOR EACH ROW
corps_trigger;
```

T5
```
CREATE TRIGGER before_delete_entreprise BEFORE UPDATE ON entreprise
FOR EACH ROW
corps_trigger;
```

T6
```
CREATE TRIGGER after_delete_entreprise AFTER UPDATE ON entreprise
FOR EACH ROW
corps_trigger;
```

Règle de nommage

nom_trigger = moment_evenement_table.

Règle

Il ne peut exister qu'un seul trigger par combinaison_trigger/evenement_trigger par table.

Les Triggers

Les mots clé : OLD et NEW

À l'intérieur d'un trigger sous MySQL, vous pouvez utilisez deux mots-clés : OLD et NEW.

Trigger moment_INSERT_T1 {

…….
…….

Pas de OLD
NEW disponible
}

Insertion d'une nouvelle Ligne donc pas de **OLD** et **NEW** existe.

Trigger moment_UPDATE_T2 {

…….
…….

OLD disponible
NEW disponible

}

modification d'une ligne ancienne Par une nouvelle ligne donc pas de **OLD** existe et **NEW** existe.

Trigger moment_DELETE_T1 {

…….
…….

OLD disponible
Pas de NEW
}

suppression d'une ligne ancienne donc pas de **OLD** existe et pas de **NEW**

Les Triggers

Les mots clé : OLD et NEW

À l'intérieur d'un trigger sous MySQL, vous pouvez utilisez deux mots-clés : OLD et NEW.

— OLD : représente les valeurs des colonnes de la ligne traitée avant qu'elle ne soit modifiée par l'événement déclencheur.
Ces valeurs peuvent être lues, mais pas modifiées.

— NEW : représente les valeurs des colonnes de la ligne traitée après qu'elle a été modifiée par l'événement déclencheur.
Ces valeurs peuvent être lues et modifiées.

résumé

Evénement déclencheur	OLD	NEW
INSERT	n'existe pas	existe
UPDATE	existe	existe
DELETE	existe	n'existe pas

Les Triggers

Exemples:

id_entre	nrue	rue	ville	nom_entreprise
1	7	Bisheim	casa	IBM
2	9	Honeim	stras	FORD
3	9	hay K1	Ifran	BMW
4	11	schilik	karlsruhe	Audi
6	45	Vosges	Nice	instek

7	23	clark	bern	interdigit

1 CALL InsertionE(7, 23 , ' clark', 'Kehl', 'interdigit');

Pendant le traitement de cette ligne par le trigger correspondant,

- NEW.id_entre vaudra 7 ;
- NEW.nrue vaudra 23 ;
- NEW.rue vaudra clark ;
- NEW.ville vaudra bern ;
- NEW.nom_entreprise vaudra interdigit;

Les valeurs de OLD ne seront pas définies.

2 DELETE from entreprise where id_entre=2

Pendant le traitement de cette ligne par le trigger correspondant,

- OLD.id_entre vaudra 2;
- OLD.nrue vaudra 9 ;
- OLD.rue vaudra Honeim ;
- OLD.ville vaudra stras ;
- OLD.nom_entreprise vaudra FORD;

Les valeurs de NEW ne seront pas définies.

3 UPDATE entreprise SET rue= 'ain vital'
Where id_entre=3;

Pendant le traitement de cette ligne par le trigger correspondant,

- OLD.id_entre = 3 = NEW.id_entre
- OLD.nrue = hay k1; NEW.rue=ain vital
- OLD.rue= Honeim =NEW.rue
- OLD.ville = stras = NEW.ville
- OLD.nom_entreprise =FORD= NEW.nom_entreprise

Les Triggers

Exemples (vérification et validation des types et des données)

Pour cela, on considère la table Client de la base de données magasin, on rajoute une colonne specilaite de type char(1) et cette derniere accepte seulement 3 valeurs ‘S’ (scientifique) , ’L’ (littéraire) et la valeur NULL,

```
ALTER TABLE `magasin`.`Client`
ADD COLUMN `specialite` CHAR(1) NULL after `ville`;
update magasin.Client set specialite='S' where idClient IN (1,3,5,7,9,11,21,23);
update magasin.Client set specialite='L' where idClient NOT IN (1,3,5,7,9,11,21,23);
```

idClient	nom_Client	prenom	age	adresse	ville	specialite	mail
1	Marti	Jean	36	5 av. Einstein	Strasbourg	S	mart@marti.com
2	Rapp	Paul	44	32 av. Foch	Paris	L	rapp@libert.com
3	Devos	Marie	18	75 bd Hochimin	Lile	S	grav@waladoo.fr
4	Hochon	Paul	22	33 rue Tsétsé	Chartres	L	hoch@fiscali.fr
5	Grave	Nuyen	18	75 bd Hochimin	Lile	S	NULL
6	Hachette	Jeanne	45	60 rue d'Amiens	Versailles	L	NULL
7	Marti	Pierre	25	4 av. Henri Paris	Paris	S	martin7@fiscali.fr
8	Mac Neal	John	52	89 rue Diana	Lyon	L	mac@freez.fr
9	Basile	Did	37	26 rue Gallas	Nantes	S	bas@walabi.com
10	Darc	Jeanne	19	9 av. d'Orléans	Paris	L	NULL
11	Gaté	Bill	45	9 bd des Bugs	Lyon	S	bill@microhard.be
20	nougeaine	Fahd	25	5 av. Einstein	barcelona	L	fah@yahoo.com
21	Sabri	wissam	22	32 av. Foch	Weert	S	Sabri@libert.com
22	Kablan	Katy	30	75 bd Hochimin	Poitier	L	katy@waladoo.fr
23	hassan	ali	36	5 av. Einstein	Agadir	S	ali@marti.com
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Les Triggers

Exemples (vérification et validation des types et des données)

Voici un trigger pour l'insertion, qui empêche la saisie d'un autre caractère que "S" ou "L" pour specialite.

```
DELIMITER |
CREATE TRIGGER before_insert_client BEFORE INSERT
ON Client FOR EACH ROW
BEGIN
IF NEW.specialie IS NOT NULL -- le specialite n'est pas NULL
AND NEW.specialite != 'S' -- ni  "S"
AND NEW.specialite != 'L' -- ni   "L"
THEN
SET NEW.specialite = NULL;
END IF;
END |
DELIMITER ;
```

Ce trigger se déclenche avant l'insertion et mettra la valeur NULL pour les saisies hors de 'S' et 'L'.

```
INSERT INTO magasin.Client (idClient, nom_Client, prenom, age,adresse, ville, specialite, mail)
VALUES ('24', 'hadadi', 'idir', 23, 'ait hdidou', 'Tilmi', 'B','dir@gmail.com');
```

idClient	nom_Client	prenom	age	adresse	ville	specialite	mail
24	hadadi	idir	23	ait hdidou	Tilmi	NULL	dir@gmail.com

Les Triggers

Exemples (vérification et validation des types et des données)

Voici un autre trigger pour la modification, qui empêche la saisie d'un autre caractère que "S" ou "L" pour specialite.

```
DELIMITER |
CREATE TRIGGER before_update_client BEFORE UPDATE
ON Client FOR EACH ROW
BEGIN
IF NEW.specialie IS NOT NULL -- le specialite n'est pas NULL
AND NEW.specialite != 'S' -- ni  "S"
AND NEW.specialite != 'L' -- ni   "L"
THEN
SET NEW.specialite = NULL;
END IF;
END |
DELIMITER ;
```

Ce trigger se déclenche avant la modification et mettra la valeur NULL pour les saisies hors de 'S' et 'L'.

```
UPDATE magasin.Client SET specialite ='N' where idClient=24;
```

idClient	nom_Client	prenom	age	adresse	ville	specialite	mail
24	hadadi	idir	23	ait hdidou	Tilmi	NULL	dir@gmail.com

Les Triggers

Exemples (vérification et validation des types et des données)

trigger pour l'insertion, qui empêche la saisie d'un autre caractère que "S" ou "L" pour specialite
par la provocation d'une erreur et l'affichage d'un message explicatif. Pour cela, on va créer une table auxiliaire avec la structure et les contraintes suivantes:

```
CREATE TABLE Erreur (
id TINYINT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
message_erreur VARCHAR(255) UNIQUE);
```

Table Erreur

id	message_erreur
1	Specialite doit valoir "S", "L" ou NULL

```
-- Insertion de l'erreur qui nous intéresse
INSERT INTO Erreur (message_erreur) VALUES
('Erreur : specialite doit valoir "S", "L" ou NULL.');
```

principe

Notre trigger doit:

1) Utiliser notre table Erreur,
2) Exploiter la contrainte UNIQUE pour déclenché une erreur (violation en insérant la même valeur pour l'attribut 'message_erreur') en cas de valeur erroné saisie par l'utilisateur.
3) Se déclencher avant l'événement d'insertion ou de modification

Les Triggers

Exemples (vérification et validation des types et des données)

trigger pour l'insertion, qui empêche la saisie d'un autre caractère que "S" ou "L" pour specialite
par la provocation d'une erreur et l'affichage d'un message explicatif. Pour cela, on va créer une table auxiliaire avec la structure et les contraintes suivantes:

```
-- Création du trigger
DELIMITER |
CREATE TRIGGER before_insert_client BEFORE INSERT
ON Client FOR EACH ROW
BEGIN
IF NEW.specialie IS NOT NULL -- le specialite n'est pas NULL
AND NEW.specialite != 'S' -- ni  "S"
AND NEW.specialite != 'L' -- ni   "L"
THEN
INSERT INTO Erreur (message_erreur) VALUES
('Erreur : specialite doit valoir "S", "L" ou NULL.');
END IF;
END |
DELIMITER ;
```

Table Erreur

id	message_erreur
1	Specialite doit valoir "S", "L" ou NULL

```
INSERT INTO magasin.Client (idClient, nom_Client, prenom, age,adresse, ville, specialite, mail)
VALUES ('25', 'alami', 'hicham', 23, 'bensouda', 'nador' , 'B', 'hi@gmail.com');
```

ERROR 1062 (23000):Duplicate entry 'Erreur : specialite doit valoir "S", "L" ou NULL

Les Triggers

Exemples (vérification et validation des types et des données)

deux trigger pour l'insertion et la modification, qui traitent la saisie des valeurs hors 'TRUE' et 'FALSE' pour la colonne reussi

```
INSERT INTO Erreur (message_erreur) VALUES
('Erreur : reussi doit valoir TRUE (1) ou FALSE (0).');
DELIMITER |
CREATE TRIGGER before_insert_etudiant BEFORE INSERT
ON Etudiant FOR EACH ROW
BEGIN
IF NEW.reussi != TRUE -- ni TRUE
AND NEW.reussi != FALSE -- ni FALSE
THEN
INSERT INTO Erreur (message_erreur) VALUES
('Erreur : reussi doit valoir TRUE (1) ou FALSE (0).');
END IF;
END |
CREATE TRIGGER before_update_etudiant BEFORE UPDATE
ON Etudaint FOR EACH ROW
BEGIN
IF NEW.reussi != TRUE -- ni TRUE
AND NEW.reussi != FALSE -- ni FALSE
THEN
INSERT INTO Erreur (message_erreur) VALUES
('Erreur : reussi doit valoir TRUE (1) ou FALSE (0).');
END IF;
END |
DELIMITER ;
```

Table Erreur

id	message_erreur
1	Specialite doit valoir "S", "L" ou NULL
2	reussi doit valoir TRUE (1) ou FALSE (0)

Table Etudiant

id	nom	age	mail	reussi
1				TRUE
2				FALSE

```
UPDATE Etudiant
SET reussi = 45
WHERE client_id = 2;
```

ERROR 1062 (23000):Duplicate entry 'Erreur : reussi doit valoir TRUE (1) ou FALSE (0)

Les Triggers

Exemples (mise à jour des données/informations à travers les clés étrangères)

On considère une application simple qui porte sur une agence de location de voitures: soient deux tables suivantes:
Table *voitures* qui contient tous les véhicules de l'entreprise.
Table *reservation* qui comporte les véhicules loués et donc non disponibles

principe

1) Pour savoir les voitures disponible on passe par la requête 1,
2) On va répondre à la même question, en utilisant les triggers et une donnée stockant la disponibilité sur la table voitures

```
SELECT id_v, marque, modele,carburant FROM voitures
WHERE NOT EXISTS (
SELECT *
FROM reservation
WHERE voitures.id_v = reservation.id_v);
```

R1

Table *voitures*

id_v	modele	marque	puissance	carburant	proprietaire
1	2019	Audi	160	essence	TM
2	2018	VW	110	diesel	TM

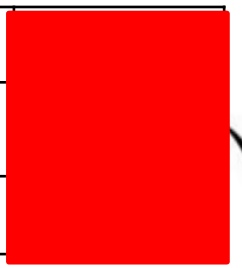


Table *reservation*

id	modele	.....	Date_res	prix	Id_client	id_v
1	….	….	…	….	2	2

Les Triggers

Exemples (mise à jour des données/informations à travers les clés étrangères)

On considère une application simple qui porte sur une agence de location de voitures: soient deux tables suivantes:
Table *voitures* qui contient tous les véhicules de l'entreprise.
Table *voitures_louees* qui comporte les véhicules loués et donc non disponibles

principe

3) On rajoute un attribut disponible
4) Cet attribut disponibe sera mise à jour grâce à trois triggers sur la table voitures

```
ALTER TABLE voitures ADD COLUMN disponible
 BOOLEAN DEFAULT TRUE;
```

```
Update voitures SET disponible=FALSE
WHERE  EXISTS (
SELECT *
FROM reservation
WHERE voitures.id_v = reservation.id_v);
```

Table *voitures*

id_v	modele	marque	puissance	carburant	proprietaire	disponible
1	2019	Audi	160	essence	TM	TRUE
2	2018	VW	110	diesel	TM	FALSE

Table *reservation*

Id_res	modele	marque	..	Date_res	Prix …	Id_client	id_v
1	2018	VW	…	..	…	7	2

Les Triggers

Exemples (mise à jour des données/informations à travers les clés étrangères)

1)À l'insertion d'une nouvelle réservation, il faut retirer la voiture louée des voitures disponibles.

```
DELIMITER |
CREATE TRIGGER after_insert_reservation AFTER INSERT
ON reservation FOR EACH ROW
BEGIN
UPDATE voitures
SET disponible = FALSE
WHERE id_v = NEW.id_v;
END |
```

Après réservation d'une voiture; le trigger remet la valeur FALSE pour l'attribut disponible dans la table voitures → la voiture n'est plus disponible

```
INSERT INTO reservation(id, modele, marque,date_res, prix,id_client, id_v)
                VALUES (1, 2018, , 'VW', …., 400,7,2);
```

La voiture 2 doit devenir indisponible dans la table voitures

Les Triggers

Exemples (mise à jour des données/informations à travers les clés étrangères)

2) en cas de suppression, il faut faire le contraire

```
CREATE TRIGGER after_delete_reservation AFTER DELETE
ON reservation FOR EACH ROW
BEGIN
UPDATE voitures
SET disponible = TRUE
WHERE id_v = OLD.id_v;
END |
```

Quand un client retourne une voiture à l'entreprise; le trigger remet la valeur TRUE pour l'attribut disponible dans la table voitures → la voiture est disponible pour une nouvelle location

```
DELETE FROM reservation
WHERE id_v = 2;
```

La voiture 2 doit redevenir disponible dans la table voitures

Les Triggers

Exemples (mise à jour des données/informations à travers les clés étrangères)

2) en cas de modification d'une location si la voiture louée change, il faut remettre l'ancien parmi les voitures disponibles et retirer le nouveau.

```
DELIMITER |
CREATE TRIGGER after_update_reservation AFTER UPDATE
ON voireservation FOR EACH ROW
BEGIN
IF OLD.id_v <> NEW.id_v THEN
UPDATE voitures
SET disponible = TRUE
WHERE id= OLD.id_v;
UPDATE voitures
SET disponible = FALSE
WHERE id = NEW.id_v;
END IF;
END |
DELIMITER ;
```

Quand un client change une voiture, le trigger remet TRUE pour l'ancienne voiture et un FALSE pour la voiture louée

```
UPDATE reservation
SET id_v = 1;
WHERE id_v = 2;
```

La voiture 1 doit devenir indisponible

Et la voiture2 doit redevenir disponible

Les Triggers

Suppression d'un trigger:

Pour supprimer un trigger, on utilise DROP:

```
DROP TRIGGER nom_trigger;
```

```
DROP TRIGGER  before_insert_entreprise;
```

```
DROP TRIGGER  before_insert_fonctionnaire;
```

Tout comme pour les procédures stockées, il n'est pas possible de modifier un trigger. Il faut le supprimer puis le recréer différemment.

Toute suppression d'une table, supprime également tous les triggers qui y sont attachés.

Les Triggers

Comportement avec les erreurs:

— Si un trigger BEFORE génère une erreur (non interceptée par un gestionnaire d'erreur), la requête ayant déclenché le trigger ne sera pas exécutée. Si l'événement devait également déclencher un trigger AFTER, il ne sera bien sûr pas non plus exécuté

— Si un trigger AFTER génère une erreur, la requête ayant déclenché le trigger échouera.

— une transaction interrompue par une erreur déclenchée par un trigger, elle lancera un ROLLBACK.

Les Triggers

Conflits avec les clés étrangères:

— la table Client et table commande sont liées par une clé étrangère *Commande.idClient* qui référence

Client.idClient *la caractéristique :* ON UPDATE SET NULL

ON DELETE SET NULL

Par exemple, donc en cas de suppression d'un client, toutes les commandes de ce client seront modifiés, et leur *idClient* changé en NULL (Il s'agit donc d'une modification de données)

Faites attention pour ne pas avoir des modifications deux fois par une contrainte de clé étrangère et aussi par des éventuels triggers BEFORE UPDATE et AFTER UPDATE.

—Suppression des options ON UPDATE SET NULL pour les clés étrangères;

—Déplacement de ces traitement dans les triggers associé à la table client.

— OU BIEN d'éviter l'utilisation des triggers sur les tables concernés (Commande) et d'utiliser juste les transactions et les procédures

Les Triggers

Limitations:

— Il est impossible de travailler avec des transactions à l'intérieur d'un trigger.

— Les requêtes préparées ne peuvent pas non plus être utilisées.

— Les procédures appelées par un trigger ne peuvent pas envoyer d'informations au client MySQL.

(pas de requête select qui affiche des données sur le client) et elles ne peuvent utiliser ni les transactions ni les requêtes préparés

Les vues

— Les vues sont des objets de la base de données.

— Une vue est constituée d'un nom et d'une requête de sélection.

— Une vue sera considéré logiquement comme une table qui contiendra les données sélectionnées par la requête définissant la vue.

— Une vue stocke une requête, et non pas les résultats de celle-ci.

Les vues constituent un moyen logique permettant de présenter les données sous diverses formes.

Les vues sont stockés de manière durable, comme le sont les tables ou les procédures stockées.

plan:

— Intérêt et utilité

— La gestion d'une vue (création, modification et suppression);

Les vues

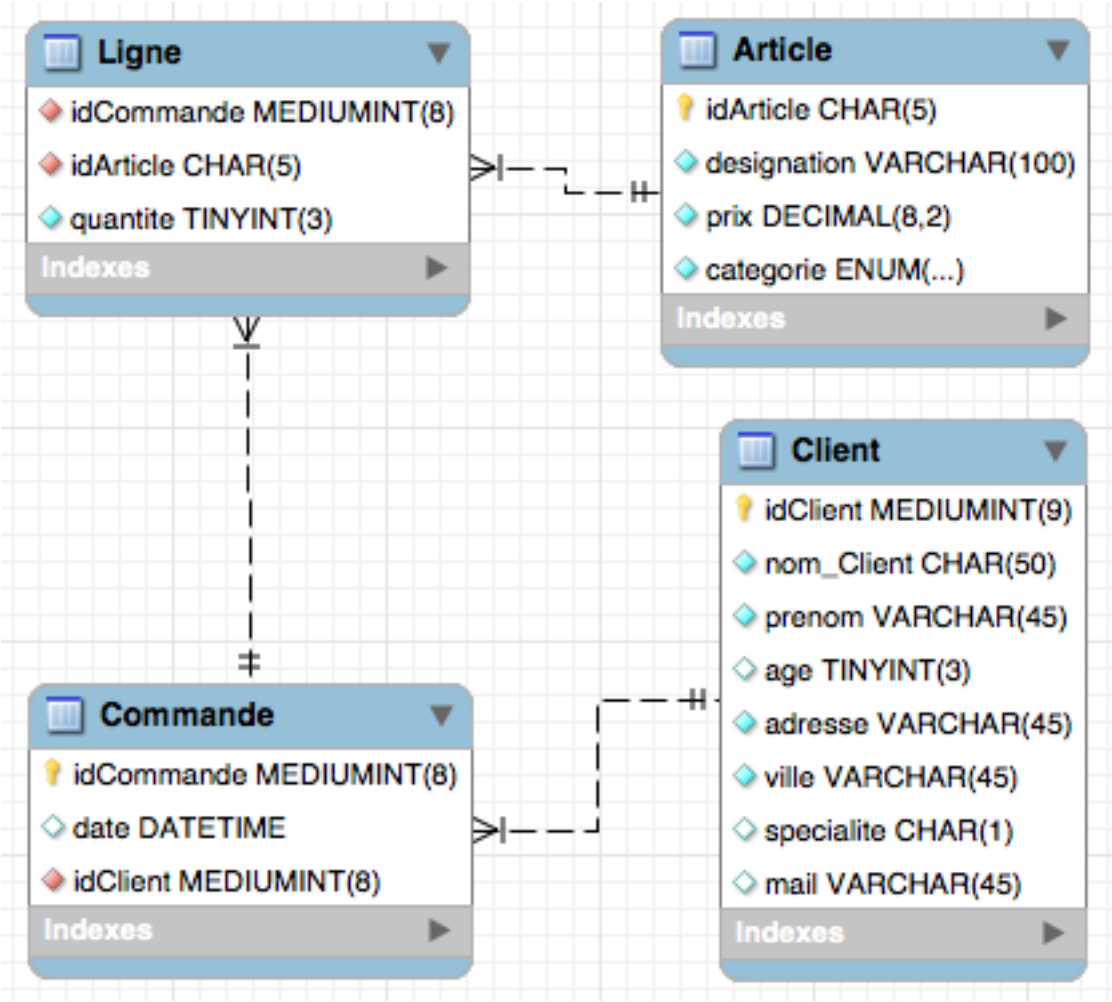
Principe:

Sur une base de données, souvent
On fait des requêtes complexes qui utilisent les données issues de plusieurs tables.

1) Commandes d'un client?
   jointure des tables client et commande.

2) Quantité de chaque article vendue?
   Tables: Article et ligne

3) Articles commandés par client ?
   Client, commande, ligne et article



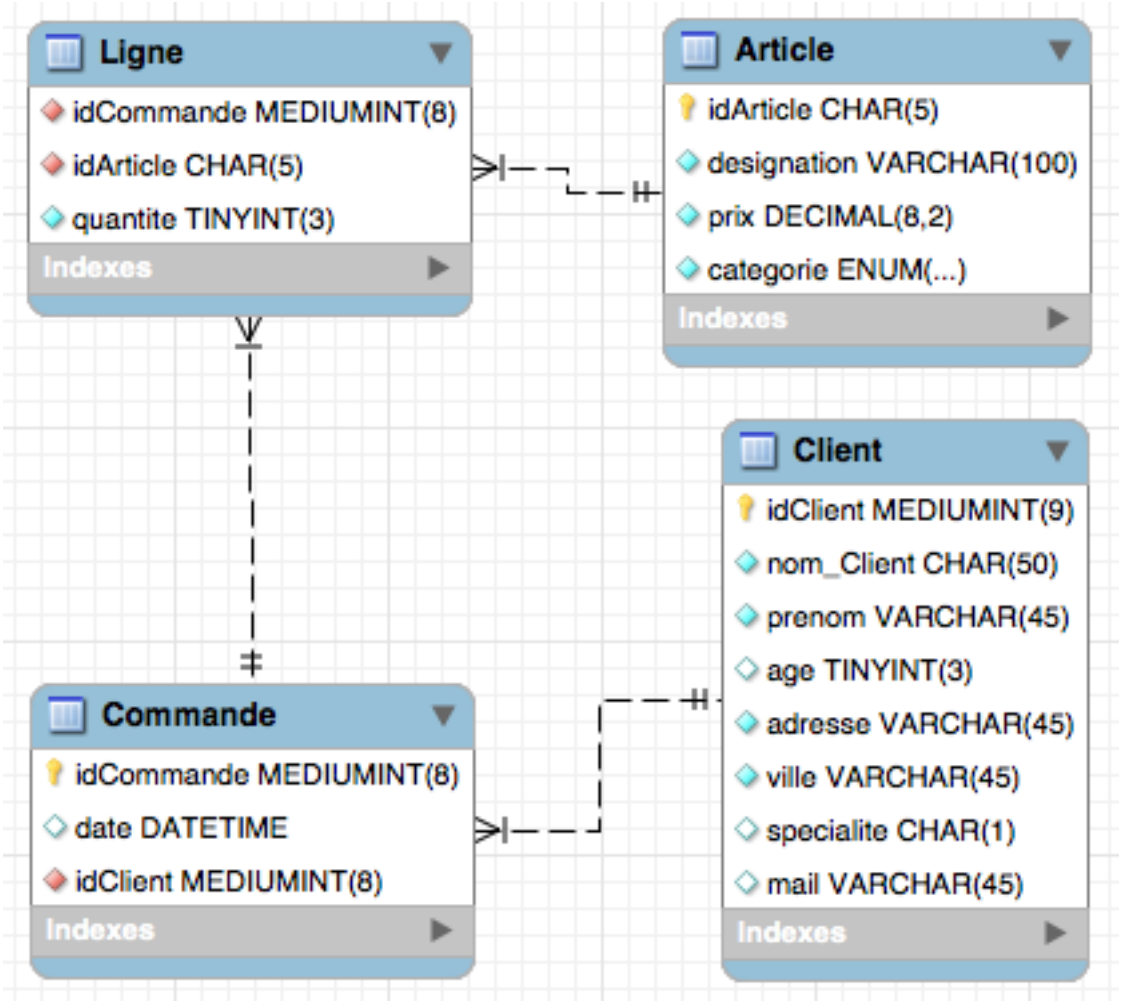
Les vues

Principe:

Parfois ça serait intéressant par exemple de trouver des renseignement sur nos clients, leurs commandes et aussi les produits achetés par ses client et leurs quantités.

Par conséquent, une requête comme suit est très utile:

```
SELECT, C.nom_Client, C.prenom, C.ville,
    Co.idCommande, A.designation, L.quantite
FROM Client C, Commande Co, Ligne L, Article A
 WHERE  C.idClient=Co.idClient
    AND Co.idCommande=L.idCommande
AND L.idArticle=A.idArticle;
```



Les vues

Principe:

```
SELECT  Client.idClient,  nom_Client, prenom, ville , Commande.idCommande, Article.designation, Ligne.quantite
FROM Client
INNER JOIN Commande ON Client.idClient=Commande.idClient
INNER JOIN Ligne ON Commande.idCommande=Ligne.idCommande
INNER JOIN Article ON  Ligne.idArticle=Article.idArticle;
```

idCli...	nom_Client	prenom	ville	idCommande	designation	quantite
1	Marti	Jean	Strasbo...	3	Nikon F80	5
2	Rapp	Paul	Paris	7	Portable Samsung X15 XVM	5
3	Devos	Marie	Lile	4	DVD vierge par 3	2
3	Devos	Marie	Lile	4	PC portable Sony Z1-XMP	3
4	Hochon	Paul	Chartres	10	Portable Dell X300	2
4	Hochon	Paul	Chartres	10	Portable Samsung X15 XVM	1
5	Grave	Nuyen	Lile	1	Cassette DV60 par 5	3
5	Grave	Nuyen	Lile	12	Cassette DV60 par 5	4
5	Grave	Nuyen	Lile	1	Caméscope Sony DCR-PC330	1
5	Grave	Nuyen	Lile	12	Caméscope Sony DCR-PC330	3
7	Marti	Pierre	Paris	8	Caméscope Panasonic SV-AV100	1
7	Marti	Pierre	Paris	8	PC portable Sony Z1-XMP	1
8	Mac Neal	John	Lyon	11	DVD vierge par 3	10
9	Basile	Did	Nantes	5	Canon EOS 3000V zoom 28/80	1
9	Basile	Did	Nantes	2	PC bureau HP497 écran TFT	2
9	Basile	Did	Nantes	13	Portable Samsung X15 XVM	2
10	Darc	Jeanne	Paris	6	Cassette DV60 par 5	3
10	Darc	Jeanne	Paris	6	Caméscope Panasonic SV-AV100	1
11	Gaté	Bill	Lyon	9	Nikon F55+zoom 28/80	1

Il serait donc bien pratique de stocker cette requête plutôt que de devoir la retaper en entier à chaque fois.

C’est très exactement le principe d’une vue : on stocke une requête SELECT en lui donnant un nom, et on peut ensuite appeler directement la vue par son nom.

Les vues

Création:

Pour créer une vue, on utilise la syntaxe suivante

```
CREATE [OR REPLACE] VIEW nom_vue
AS requete_select;
```

[OR REPLACE]: option facultative, qui permet de remplacer une vue existante par la nouvelle. dans le cas ou la vue n'existe pas elle sera crée, si cette option est omise et qu'une vue portant le même nom a été précédemment définie, cela déclenchera une erreur.

Exemples:

```
CREATE VIEW vue_ entreprise AS SELECT * FROM entreprise;

CREATE VIEW vue_fonctionnaire AS SELECT * FROM fonctionnaire;

CREATE VIEW vue_entreprise_details AS SELECT FROM entreprsie
INNER JOIN fonctionnaire ON entreprise.id_entre=fonctionnaire.id_entre;
```

Les vues

```
CREATE VIEW vue_client_details as SELECT Client.idClient, nom_Client, prenom, ville ,
Commande.idCommande, Article.designation, Ligne.quantite
FROM Client
INNER JOIN Commande ON Client.idClient=Commande.idClient
INNER JOIN Ligne ON Commande.idCommande=Ligne.idCommande
INNER JOIN Article ON Ligne.idArticle=Article.idArticle;
```

```
SELECT *
FROM
vue_client_details;
```

idCli...	nom_Client	prenom	ville	idCommande	designation	quantite
1	Marti	Jean	Strasbo...	3	Nikon F80	5
2	Rapp	Paul	Paris	7	Portable Samsung X15 XVM	5
3	Devos	Marie	Lile	4	DVD vierge par 3	2
3	Devos	Marie	Lile	4	PC portable Sony Z1-XMP	3
4	Hochon	Paul	Chartres	10	Portable Dell X300	2
4	Hochon	Paul	Chartres	10	Portable Samsung X15 XVM	1
5	Grave	Nuyen	Lile	1	Cassette DV60 par 5	3
5	Grave	Nuyen	Lile	12	Cassette DV60 par 5	4
5	Grave	Nuyen	Lile	1	Caméscope Sony DCR-PC330	1
5	Grave	Nuyen	Lile	12	Caméscope Sony DCR-PC330	3
7	Marti	Pierre	Paris	8	Caméscope Panasonic SV-AV100	1
7	Marti	Pierre	Paris	8	PC portable Sony Z1-XMP	1
8	Mac Neal	John	Lyon	11	DVD vierge par 3	10
9	Basile	Did	Nantes	5	Canon EOS 3000V zoom 28/80	1
9	Basile	Did	Nantes	2	PC bureau HP497 écran TFT	2
9	Basile	Did	Nantes	13	Portable Samsung X15 XVM	2
10	Darc	Jeanne	Paris	6	Cassette DV60 par 5	3
10	Darc	Jeanne	Paris	6	Caméscope Panasonic SV-AV100	1
11	Gaté	Bill	Lyon	9	Nikon F55+zoom 28/80	1

Les vues

```
CREATE VIEW  vue_client_details as  SELECT  Client.idClient,  nom_Client, prenom, ville ,
Commande.idCommande, Article.designation as nom_produit, Ligne.quantite
FROM Client
INNER JOIN Commande ON Client.idClient=Commande.idClient
INNER JOIN Ligne ON Commande.idCommande=Ligne.idCommande
INNER JOIN Article ON  Ligne.idArticle=Article.idArticle;
```

Version 1

Show tables;

Tables_in_magasin
Article
Client
Commande
Ligne
▶ vue_client_details

Pour visualiser la structure de notre vue
Describe vue_client_details ;

Field	Type	Null	Key	Default
▶ idClient	mediumint(9) unsigned	NO		0
nom_Client	char(50)	NO		NULL
prenom	varchar(45)	NO		NULL
ville	varchar(45)	NO		NULL
idCommande	mediumint(8) unsigned	NO		0
nom_produit	varchar(100)	NO		NULL
quantite	tinyint(3) unsigned	NO		NULL

Les champs apparaissent avec leurs alias au niveau de la vue .

Les vues

Création:

```
CREATE VIEW  vue_client_details (ID, NOM, PRENOM,VILLE,NUM_CMD,DES_PRT,QAUNTITE)
as  SELECT  Client.idClient,  nom_Client, prenom, ville, Commande.idCommande, Article.designation,
Ligne.quantite
FROM Client
INNER JOIN Commande ON Client.idClient=Commande.idClient
INNER JOIN Ligne ON Commande.idCommande=Ligne.idCommande
INNER JOIN Article ON  Ligne.idArticle=Article.idArticle;
```

Version 2

Une autre manière de Changer les noms des Colonnes sans utiliser les alias.

Show tables;

Tables_in_magasin
Article
Client
Commande
Ligne
vue_client_details

Pour visualiser la structure de notre vue

Describe vue_client_details ;

Field	Type	Null	Key	Default
ID	mediumint(9) unsigned	NO		0
NOM	char(50)	NO		NULL
PRENOM	varchar(45)	NO		NULL
VILLE	varchar(45)	NO		NULL
NUM_CMD	mediumint(8) unsigned	NO		0
DES_PRT	varchar(100)	NO		NULL
QAUNTITE	tinyint(3) unsigned	NO		NULL

Dans une vue il est Interdit d'avoir deux colonne qui portent le même nom

Les vues

Règles sur la requête SELECT définissant la vue:

La requête peut contenir une clause WHERE, une clause GROUP BY, des fonctions (scalaires ou d'agrégation), des opérations mathématiques, une autre vue, des jointures, etc.

— Il n'est pas possible d'utiliser une requête SELECT dont la clause FROM contient une sous-requête SELECT.

— La requête ne peut pas faire référence à des variables utilisateur et des variables locales (dans le cas d'une vue définie par une procédure stockée).

— Toutes les tables (ou vues) mentionnées dans la requête doivent exister (au moment de la création du moins).

Les vues

Exemple():

```
CREATE VIEW vue_scientifique
AS SELECT  idClient, nom_Client, prenom, age, adresse, ville, specialite, mail
FROM Client
WHERE specialite= 'S';
```

Une vue sur les clients Scientifiques.
Par la clause WHERE

```
CREATE OR REPLACE VIEW vue_nombre_fonctionnaires
AS SELECT entreprise.id_entre, nom_entreprise, COUNT(id_fct) AS nb
FROM entreprise
LEFT JOIN fonctionnaire ON fonctionnaire.id_entre = entreprise.id_entre
GROUP BY entreprise.id_entre;
```

```
SELECT * FROM vue_nombre_fonctionnaires;
```

id_entre	nom_entreprise	nb
1	IBM	1
2	FORD	1
3	BMW	0
4	Audi	4
6	instek	0
7	Google	0

Une vue retournant le nombre de fonctionnaires par entreprise
Par une jointure et la clause GROUP BY

Les vues

Exemple():

```
CREATE VIEW vue_sur_parisien_scientifique
AS SELECT  idClient, nom_Client, prenom, age, adresse, ville, specialite, mail
FROM vue_scientifique
WHERE ville= 'paris';
```

Les scientifiques parisiens par
Une vue exploitant une autre vue.

```
CREATE VIEW vue_article_dirhams
AS SELECT idArticle, designation, categorie, ROUND(prix*10.89, 2) AS prix_dirhams
FROM Article;
```

```
SELECT * FROM  vue_article_dirhams;
```

Une vue convertissant le prix d'article exprimé en € en dirhams
Par l'utilisation d'une expression

idArticle	designation	categorie	prix_dirhams
CA300	Canon EOS 3000V zoom 28/80	photo	3582.81
CAS07	Cassette DV60 par 5	divers	292.94
CP100	Caméscope Panasonic SV-AV100	vidéo	16226.10
CS330	Caméscope Sony DCR-PC330	vidéo	17739.81
DEL30	Portable Dell X300	informatique	18676.35
DVD75	DVD vierge par 3	divers	190.58
HP497	PC bureau HP497 écran TFT	informatique	16335.00
NIK55	Nikon F55+zoom 28/80	photo	2929.41
NIK80	Nikon F80	photo	5216.31
SAX15	Portable Samsung X15 XVM	informatique	21769.11
SOXMP	PC portable Sony Z1-XMP	informatique	26125.11

Les vues

Exemple(tri des données de la vue):

```
CREATE VIEW vue_fonctionnaire
AS SELECT id_fct, nom, nom_entreprise, ville
FROM fonctionnaire
INNER JOIN entreprise ON entreprise.id_entre = fonctionnaire.id_entre
ORDER BY nom;
```

```
SELECT *
FROM vue_fonctionnaire;
```

Sélection à partir de notre vue sans ORDER BY, on prend l'ORDER BY de la définition

```
FROM vue_fonctionnaire
ORDER BY nom_entreprise;
```

Sélection avec ORDER BY, c'est celui-là qui sera pris en compte ,celui de la vue est ignoré

Les vues

Exemple(requête SELECT reste toujours figée et constante):

les changements de structure faits par la suite sur la ou les tables sous-jacentes n'influent pas sur la vue.
Soit la vue suivante:

```
CREATE VIEW vue_client
AS SELECT *
FROM Client;
```

* Représente les colonnes: (idClient, nom_Client, prenom, age, adresse, ville, specialite, mail)

```
ALTER TABLE Client ADD COLUMN date_naissance DATE ;
```

describe vue_client;

Field	Type	Null	Key	Default
idClient	mediumint(9) unsigned	NO		0
nom_Client	char(50)	NO		NULL
prenom	varchar(45)	NO		NULL
age	tinyint(3) unsigned	YES		NULL
adresse	varchar(45)	NO		NULL
ville	varchar(45)	NO		NULL
specialite	char(1)	YES		NULL
mail	varchar(45)	YES		NULL

l'ajout d'une colonne dans la table Client ne changera pas les résultats d'une requête sur vue_client malgré le * de SELECT.

Pour que vue_client sélectionne la nouvelle colonne ajoutée, il faudrait recréer la vue pour l'inclure.

Les vues

Exemple(sélection des données d'une vue):

```
CREATE VIEW vue_client_details as SELECT Client.idClient, nom_Client, prenom, age,ville,specialite,
mail, Commande.idCommande, Article.designation, Ligne.quantite
FROM Client
INNER JOIN Commande ON Client.idClient=Commande.idClient
INNER JOIN Ligne ON Commande.idCommande=Ligne.idCommande
INNER JOIN Article ON Ligne.idArticle=Article.idArticle;
```

```
SELECT idClient, nom_Client, prenom, age, ville, specialite, mail
FROM vue_client_details
WHERE quantite>= 3;
```

idClient	nom_Client	prenom	age	ville	specialite	designation	quantite
5	Grave	Nuyen	18	Lile	S	NULL	3
10	Darc	Jeanne	19	Paris	L	NULL	3
5	Grave	Nuyen	18	Lile	S	NULL	4
5	Grave	Nuyen	18	Lile	S	NULL	3
8	Mac Neal	John	52	Lyon	L	mac@freez.fr	10
1	Marti	Jean	36	Strasbourg	S	mart@marti.com	5
2	Rapp	Paul	44	Paris	L	rapp@libert.com	5
3	Devos	Marie	18	Lile	S	grav@waladoo.fr	3

sélection dans vue des clients qu'ont commandés une quantité supérieur au égal à 3

Les vues

Exemple(sélection des données d'une vue):

```
CREATE OR REPLACE VIEW vue_nombre_fonctionnaires
AS SELECT entreprise.id_entre, nom_entreprise, COUNT(id_fct) AS nb
FROM entreprise
LEFT JOIN fonctionnaire ON fonctionnaire.id_entre = entreprise.id_entre
GROUP BY entreprise.id_entre;
```

id_entre	nom_entreprise	nb
1	IBM	1
2	FORD	1
3	BMW	0
4	Audi	4
6	instek	0
7	Google	0

```
SELECT vue_nombre_fonctionnaires.id_entre,
vue_nombre_fonctionnaires.nom_entreprise, entreprise.rue, entreprise.ville,
vue_nombre_fonctionnaires.nb
FROM vue_nombre_fonctionnaires
INNER JOIN entreprise ON entreprise.id_entre = vue_nombre_fonctionnaires.id_entre;
```

id_entre	nom_entreprise	rue	ville	nb
1	IBM	Bisheim	casa	1
2	FORD	Honeim	stras	1
3	BMW	hay K1	Ifran	0
4	Audi	schilik	karlsruhe	4
6	instek	Vosges	Nice	0
7	Google	place kleber	Obernai	0

utilisation d'une vue comme une table pour faire la sélection et la jointure

Les vues

Modification d'une vue:

La modification se fait par la clause OR REPLACE de la requête en lui spécifiant le nom de la vue à modifier

```
CREATE VIEW OR REPLACE vue_ entreprise AS SELECT * FROM entreprise where id_entre=5;
```

```
CREATE VIEW OR REPLACE vue_fonctionnaire AS SELECT * FROM fonctionnaire where age>24;
```

Si le nom spécifié n'existe pas, une nouvelle vue sera crée.

Les vues

Modification d'une vue:

une autre alternative c'est l'utilisation de la commande ALTER VIEW en lui spécifiant aussi le nom de la vue à modifier.

```
ALTER VIEW  vue_article_dirhams
AS SELECT idArticle, designation, categorie, ROUND(prix*11.89, 2) AS prix_dirhams
FROM Article;
```

```
ALTER VIEW vue_fonctionnaire AS SELECT * FROM fonctionnaire where ville='casa';
```

Suppression d'une vue:

Pour supprimer une vue, on utilise simplement DROP VIEW [IF EXISTS] nom_vue;

```
DROP VIEW vue_fonctionnaire;
```

Les vues

Intérêt et utilité:

— Simplification et facilitation de la lisibilité des codes et des requêtes SQL

Avoir des vues pour des requêtes complexes (plusieurs tables, des calculs, regroupements et des conditions) permet de simplifier et d'améliorer par la suite la lisibilité des autres requêtes.

On reprend l'exemple de l'entreprise de location de voiture avec le schéma (voiture, modèle et réservation).
Si on veut savoir quels sont modèles rentables rapportant plus d'argent, année après année.
On doit formuler la requête suivante:

```
CREATE OR REPLACE VIEW vue_revenus_annee_modele
AS SELECT YEAR(date_reservation) AS annee, modele.id AS modele_id,
SUM(reservation.prix) AS somme, COUNT(reservation.id) AS nb
FROM reservation
INNER JOIN voiture ON voiture.id = reservation.voiture_id
INNER JOIN modele ON voiture.modele_id = modele.id
GROUP BY annee, modele.id;
```

Cette vue va nous permettre de faire des requêtes très simple

Les vues

Intérêt et utilité:

— Simplification et facilitation des lisibilité des codes et des requêtes SQL

```
CREATE OR REPLACE VIEW vue_revenus_annee_modele
AS SELECT YEAR(date_reservation) AS annee, modele.id AS modele_id,
SUM(reservation.prix) AS somme, COUNT(reservation.id) AS nb
FROM reservation
INNER JOIN voiture ON voiture.id = reservation.voiture_id
INNER JOIN modele ON voiture.modele_id = modele.id
GROUP BY annee, modele.id;
```

Les revenus obtenus par année:

```
SELECT annee, SUM(somme) AS total
FROM vue_revenus_annee_modele
GROUP BY annee;
```

Les revenus obtenus pour chaque modèle, toutes années :

```
SELECT modele.nom, SUM(somme) AS total
FROM vue_revenus_annee_modele
INNER JOIN modele ON
vue_revenus_annee_espece.modele_id = modele.id
GROUP BY modele;
```

Les vues

Intérêt et utilité:

— efficacité et rapidité d'interfaçage entre les applications (C#,PHP,JAVA) et les bases de données

À travers ces applications, si les requêtes utilisent directement les tables sans les vues, la moindre modification de schéma de la base de données influencera toutes ses requêtes. En revanche, s'elles utilisent des vues un changement de la base de données nécessite seulement une simple modification au niveau des vues pour que les mêmes requêtes fonctionnent sans aucun souci.

Par exemple, on considère la table client:

idClient	nom_Client	prenom	age	adresse	ville	specialite	mail

```
CREATE VIEW vue_client
AS SELECT *
FROM Client;
```

Évolution de la base magasin, la table client sera scindée en 2:

client

IdClient	Nom_Client	prenom	age	specialite	mail	Id_adresse

adresse

Id	adresse	ville	Code_postal	ville

```
CREATE VIEW vue_client
AS SELECT idClient, nom_Client, prenom, adresse, code_postal, ville, pays, mail
FROM Client
LEFT JOIN Adresse ON Client.adresse_id = Adresse.id
```

On garde toutes es requêtes basé sur l'ancienne vue_client qu'a utilisé la table client seulement

Les vues

Intérêt et utilité:

— le contrôle stricte de la visibilité des données par les utilisateurs

On suppose qu'on a 3 utilisateurs (U1,U2,U3)

U1 peut faire les requêtes CRUD sur la table T1 ;

U2 peut faire des requêtes de sélections sur la table T1;

U3 peut faire des sélections sur une partie de la table T1;

Il suffit de créer la vue vue_T1, n'ayant accès qu'aux colonnes/lignes "publiques" de la table T1, et de donner à U3 les droits sur la vue vue_T1, mais pas sur la table T1.

```
CREATE VIEW vue_client_stagiaire
AS SELECT idClient, nom_Client, ville, FROM Client;
```

On lui donne un accès limité grâce à la vue vue_client_stagiaire. Ceci permet de cacher certaines données sensibles dans la table selon bien sur les utilisateurs.