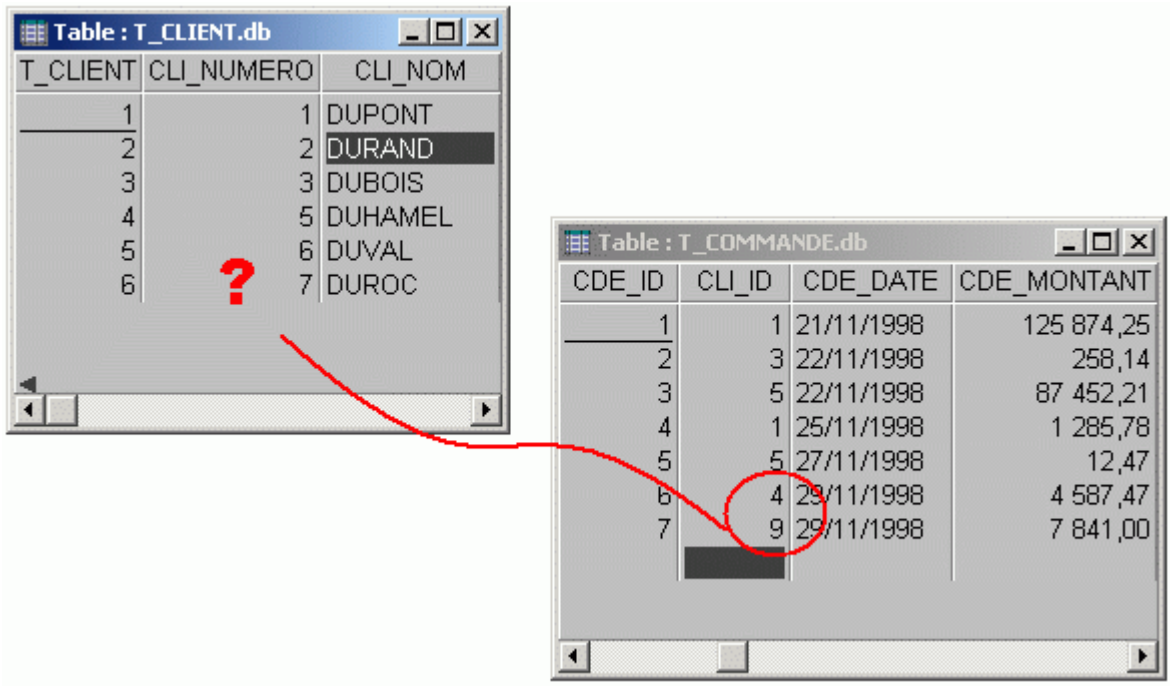
LES TRIGGERS (DÉCLENCHEURS)

L'intégrité référentielle sert à empêcher qu'une ligne d'une table qui référence une ligne d'une autre table voit le lien logique entre les deux lignes brisée. Que serait une facture si le client venait à être effacé de la table des clients ?



Ici, les commandes 6 et 7 font références aux client 4 et 9 qui n'existent pas dans la table T_CLIENT...

Le mécanisme

Le mécanisme d'intégrité référentielle doit permettre d'assurer :

- que tout client référencé par une autre table ne soit pas supprimé, **ou alors**
- que l'on supprime aussi toutes les lignes filles des tables qui référence le client supprimé
- que la référence du client, si elle est modifié, soit répercutée dans toutes les lignes des tables filles qui la référence, **ou alors**
- que toute modification de cette référence soit interdite si des lignes de tables filles l'utilise
- une dédicace à islème, mathieu l & d, célia, amandine, serge ... vous êtes trop nombreux à toute la promo !

La norme SQL 2 a prévu les modes de gestion des intégrités référentielles suivants :

ON DELETE { NO ACTION | CASCADE } et **ON UPDATE { NO ACTION | CASCADE }**

```
CREATE TABLE client (
  cl_id    INTEGER PRIMARY KEY AUTO_INCREMENT,
 cl_name  TEXT NOT NULL
) ENGINE=InnoDB;

CREATE TABLE commande (
  cl_id      INTEGER NOT NULL REFERENCES client( cl_id ) ON DELETE NO
ACTION,
  rc_date     DATE NOT NULL,
  rc_montant  NUMERIC( 12, 2 ),
  PRIMARY KEY( cl_id, rc_date ),
  KEY( rc_date, cl_id )
) ENGINE=InnoDB;
```

Qui signifie que :

ON DELETE NO ACTION	Si une supression intervient alors que le client est référencé par une commande, la suppression est avortée
ON DELETE CASCADE	Si une supression intervient alors que le client est référencé par une commande, la ou les commandes référencées pour ce client sont aussi supprimées
ON UPDATE NO ACTION	Si une mise a jour de la clef du client intervient alors que ce client est référencé par une commande, la modification est avortée
ON UPDATE CASCADE	Si une mise a jour de la clef du client intervient alors que ce client est référencé par une commande, la ou les commandes référencée pour ce client voit leur clef étrangère prendre la nouvelle valeur

Bien entendu on peut parfaitement supprimer un client qui ne possède pas de commande ou modifier la valeur de sa clef même lorsque le mode NO ACTION est actif.

Dans tous les cas, le mode NO ACTION est à préférer. En effet le mode CASCADE peut entraîner une avalanche de suppressions ou de mise à jour du plus mauvais effet sur les performances du SGBDR !

Ce mécanisme simple et robuste est complété par d'autres possibilités qu'offre la norme SQL et qui sont : SET DEFAULT et SET NULL.

ON DELETE SET DEFAULT	Si une supression intervient alors que le client est référencé par une commande, les lignes des tables filles référencées par ce client voit la valeur de la clef passer à la valeur par défaut définie lors de la création de la table.
ON DELETE SET NULL	Si une supression intervient alors que le client est référencé par une commande, la ou les commandes référencées pour ce client voit la valeur de la clef étrangère passer à NULL.
ON UPDATE SET DEFAULT	Si une mise a jour de la clef du client intervient alors que ce client est référencé par une commande, les lignes des tables filles référencées par ce client voit la valeur de la clef passer à la valeur par défaut définie lors de la création de la table.
ON UPDATE SET NULL	Si une mise a jour de la clef du client intervient alors que ce client est référencé par une commande, la ou les commandes référencées pour ce client voit la valeur de la clef étrangère passer à NULL.

Ces deux nouvelles règles proposent une alternative afin de gérer de manière intelligente une pseudo cascade...
Quel intérêt me direz vous d'avoir des commandes dont la référence du client est NULL ou bien 0 (un client générique dont l'existence physique est fausse, par exemple moi même...) ?

L'intérêt est simple : pouvoir faire le ménage dans les tables de manière ultérieure... Par exemple on pourra prévoir de supprimer toutes les commandes dont la référence client est NULL ou 0 la nuit, lorsque le traffic du serveur est très faible dans un batch planifié déclenché automatiquement. Bien entendu, dans ce cas il faut faire attention à ne pas comptabiliser dans les requêtes les lignes dont la référence du client est NULL ou 0. Ainsi un cumul du montant des commandes pour connaître le chiffre d'affaire généré mensuellement ne doit pas prendre en compte les lignes avec une référence de client 0 ou NULL...
Tous les SGBDR ne sont pas aussi performant que la norme l'impose en matière de gestion de l'intégrité référentielle.

L'intégrité référentielle, fondement des SGBDR...

La gestion de l'intégrité référentielle est de loin le point le plus important pour décider si un SGBD est relationnel ou non. Dépourvu de ce mécanisme il agit comme au bon vieux temps des fichiers COBOL. Muni de ce mécanisme il agit en responsable de l'intégrité des données. **Autrement dit un SGBD qui n'est pas doté d'une mécanisme de gestion des intégrités référentielles ne mérite tout simplement pas le nom de système de gestion de bases de données "relationnelles"**.

Peut on se passer de l'intégrité référentielle ?

Dans la réalité c'est impossible ! La tentation de ne pas utiliser l'intégrité référentielle (ou d'avoir un SGBD qui ne le supporte pas) et faire cela dans du code client est séduisante... Mais casse gueule : l'idée phalacieuse qui consiste

à dire, je supprime d'abord les factures ensuite les clients (voir tableau ci dessus), est inacceptable si elle n'est pas conduite dans une transaction. En effet rien n'empêche le code client de s'arrêter (panne disque, coupure de courant, réseau HS, pause café...) entre les deux requêtes ce qui supprime les commandes sans avoir pu supprimer le client...
On peut effectivement se passer de l'intégrité référentielle à condition que la base de données supporte les triggers...

A quoi servent les déclencheurs (Triggers) ?

S'il y avait 2 choses à retenir pour constituer un SGBDR au sens relationnel du terme, se serait les transactions et les triggers. Avec ces deux outils, on peut aisément créer tous les mécanismes d'intégrité référentielle que l'on souhaite et assurer une parfaite intégrité des données. C'est d'ailleurs pour cela que la norme SQL 3 a imposé l'utilisation des triggers.

C'est quoi un trigger ?

Le terme français est "déclencheur". Un trigger est donc un élément de code qui se déclenche sur un événement précis, se rapportant à un objet de la base de données. C'est exactement le même concept que la programmation événementielle dans le cadre d'interfaces graphiques.

Au sens de la norme SQL 3, un trigger se définit uniquement sur les objets de type TABLE, concerne les événements suivants : INSERT, UPDATE, DELETE et peut être déclenché BEFORE (c'est à dire avant survenance de l'événement) ou AFTER (c'est à dire après survenance de l'événement).

La syntaxe normative SQL 3 d'un trigger est la suivante :

```
CREATE TRIGGER nom_trigger
{BEFORE | AFTER} {INSERT | UPDATE | DELETE}
ON nom_table
code
```

Elle est plus conséquente dans son étendue, mais seule cette partie nous intéresse car elle en synthétise toutes les possibilités.

La chose la plus attractive dans un trigger, sont les pseudo tables OLD et NEW qui contiennent les données en cours d'insertion, de mise à jour ou de suppression...

Quand vous aurez fini cette introduction vous pourrez lire (si vous le souhaitez) le tutoriel :

http://openclassrooms.com/courses/administrez-vos-bases-de-donnees-avec-mysql/triggers

Comment ça marche ?

Examinons ce qui se passe dans un cas concret.
Soit les tables :

CREATE TABLE T_CLIENT (CLI_ID INTEGER NOT NULL PRIMARY KEY, CLI_NOM CHAR(32) NOT NULL, CLI_PRENOM VARCHAR(32), CLI_STATUT CHAR(1))	INSERT INTO T_CLIENT VALUES (1, 'DUPONT', 'Marcel', 'C') INSERT INTO T_CLIENT VALUES (4, 'DUFOUR', 'Martin', 'C') INSERT INTO T_CLIENT VALUES (17, 'DUHAMEL', 'Manuel', 'C') INSERT INTO T_CLIENT VALUES (161, 'DUBOIS', 'Marie', 'C')
CREATE TABLE T_PROSPECT (PRP_ID INTEGER NOT NULL PRIMARY KEY, PRP_NOM CHAR(32) NOT NULL, PRP_PRENOM VARCHAR(32))	INSERT INTO T_PROSPECT VALUES (14, 'Laporte', 'Eric') INSERT INTO T_PROSPECT VALUES (17, 'Lambert', 'Ernest') INSERT INTO T_PROSPECT VALUES (161, 'Laurent', 'Emeric') INSERT INTO T_PROSPECT VALUES (4874, 'Lautier', 'Etienne')

Notre utilisateur veut faire passer tout les prospects dans la table T_CLIENT avec le statut 'P'. Oui, mais, au passage il veut leur donner une clef comptatible avec le CLI_ID et reformater le nom en majuscule...
Or un simple ordre d'insertion basique du SQL va violer la contrainte d'unicité de la clef et rejetter en bloc l'insertion...

```
INSERT INTO T_CLIENT
SELECT PRP_ID, PRP_NOM, PRP_PRENOM, 'P'
FROM T_PROSPECT
```

renvoie l'erreur : « violation d'intégrité »

Mais nous pouvons pallier à ce problème, sans rien toucher de notre ordre d'insertion en ajoutant un trigger qui va rectifier les données en cours d'insertion AVANT qu'elle ne soient réellement déposées dans la table :

```
DROP TRIGGER TRG_INS_BEF_CLIENT    /*SUPPRIME UN TRIGGER DU MEME NOM AU CAS OÙ*/
```

```
CREATE TRIGGER TRG_INS_BEF_CLIENT
BEFORE INSERT
ON T_CLIENT
FOR EACH ROW
SET NEW.CLI_ID = NEW.CLI_ID + (SELECT MAX(CLI_ID) FROM T_CLIENT),
    NEW.CLI_NOM = UPPER(NEW.CLI_NOM)
/*on modifie les données de la pseudo table avant de les insérer définitivement */
```

Dès lors notre insertion va bien se passer :

```
INSERT INTO T_CLIENT
SELECT PRP_ID, PRP_NOM, PRP_PRENOM, 'P'
FROM T_PROSPECT
```

A quoi ça sert ?

Mais, me direz vous, cela nous aurions pu le faire directement dans l'ordre d'insertion... Par exemple avec la requête suivante :

```
INSERT INTO T_CLIENT
SELECT PRP_ID + (SELECT MAX(CLI_ID) FROM T_CLIENT), UPPER(PRP_NOM), PRP_PRENOM, 'P'
FROM T_PROSPECT
```

C'est tout à fait vrai, mais si vous aviez des données provenant de toutes autres tables, alors il aurait fallut adapter votre code et le modifier pour chaque cas, alors que dans le cas d'un trigger, ce code est générique quelque soit l'insertion que vous voulez faire et d'où qu'elle provienne !

Les cas d'utilisation des triggers sont assez large :

- le formatage de données
- l'auto incrémentation de clefs
- la suppression en cascade
- l'abandon d'une suppression si elle entraîne des lignes orphelines

et plus généralement le traitements d'associations provenants de modèles complexes comme

- les relations 1:n ou n est une limite fixe (par exemple 3)
- les modèles à héritage (par exemple une entité générique VEHICULE et des entités spécialisées comme VOITURE, AVION et BATEAU...)
- ...

Essayez donc de modéliser une relation dans laquelle le lien entre table mère et table fille ne doit pas dépasser un maximum de 3 occurrences... Par exemple le prêt de livres dans une bibliothèque municipale ?

Dans un tel cas, pour empêcher une insertion surnuméraire, il suffit de compter les lignes déjà insérées, et dès que le nombre de ligne dépasse 3, alors l'invocation d'un ordre ROLLBACK empêche la pseudo table NEW d'atteindre la table cible. Les données surnuméraires ne sont alors pas insérées.

Voici l'exemple d'un tel trigger :

```
CREATE TRIGGER TRG_INS_BEF_PRET
BEFORE INSERT
ON T_CLIENT
FOR EACH ROW

IF EXISTS(SELECT 1
        FROM   T_PRET P
               JOIN NEW N
                   ON P.EMPRUNTEUR_ID = N.EMPRUNTEUR_ID
        HAVING COUNT(*) = 3)
THEN
   ROLLBACK
```

Que fait-on dans ce code ? D'abord notez l'expression **FOR EACH ROW**, qui signifie que le déclencheur va s'activer autant de fois qu'il y a de ligne dans la table NEW, en substituant la table NEW et ses n lignes en n déclenchements d'une table NEW ne contenant qu'une seule ligne.
Ensuite notez que l'on lie la table NEW à la table des prets sur l'identifiant de l'emprunteur. Puis on compte le nombre de lignes de cette jointure et on invoque un ROLLBACK si ce comptage est égal à tois, afin de ne pas rajouter de nouvelles lignes d'emprunts.
Simple non ? Et tellement élégant !

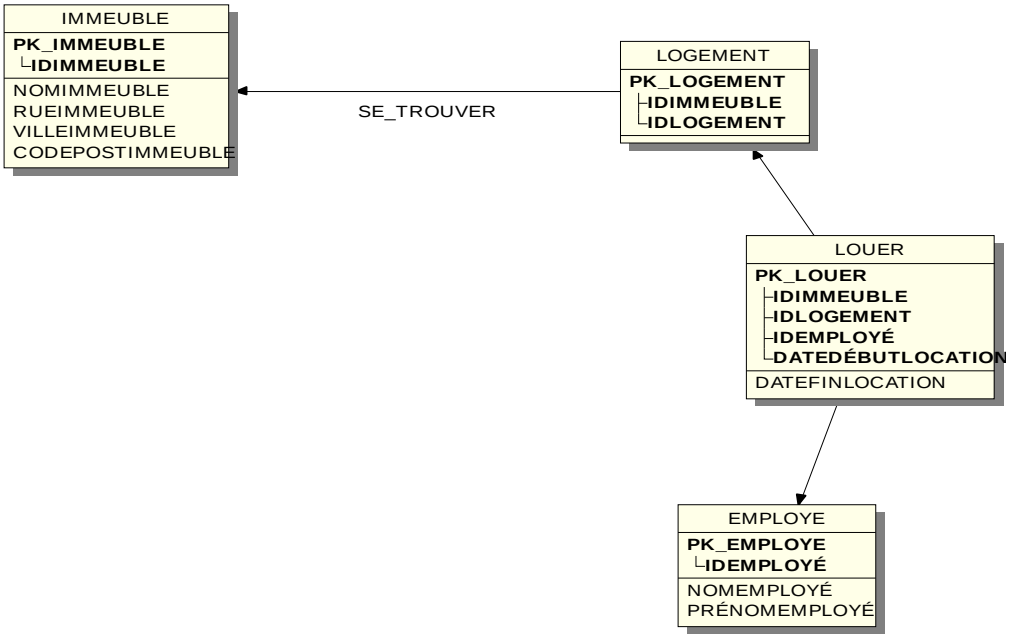
Et si on en a plus besoin un petit *DROP TRIGGER NOM_TRIGGER et il est supprimé*

Peut-on s'en passer ? Faire autrement ?

Si l'on n'utilise pas de modéles complexes et que l'intégrité référentielle est en place, alors il est parfaitement possible de se passer des triggers. Mais vu leur souplesse, la sécurité qu'ils apportent, et la concision du code qu'ils offrent, il serait folie de s'en priver ! Si si lukas

EXERCICES :

Soit le Modèle Logique des Données suivant :



Soit le trigger suivant :

```
create trigger TD_IMMEUBLE
before delete on IMMEUBLE
begin
    delete LOGEMENT
    from LOGEMENT,deleted
    where
        LOGEMENT.IDIMMEUBLE = deleted.IDIMMEUBLE
end
```

1. A quel moment est déclenché ce trigger ? Expliquez (en français correct) ce qu'il fait.

2. On vous demande de gérer toutes les contraintes d’intégrité référentielle de la base sous forme de triggers. Citez les différents triggers qui permettront de le faire en présentant pour chaque table concernée la liste des triggers à prévoir (sans préciser le code).

Exemple :

Table X

- *Trigger sur Insertion*
- *Trigger sur modification*

Si vous avez fini les corrections sont à la page suivante !

Qais arrête de manger les chocolats de Haïfa

Corrigé triggers

1. Quand est déclenché ce trigger ? Expliquez (en français correct) ce qu'il fait.

Ce trigger se déclenche lors de la suppression d'une ou plusieurs occurrences dans la table immeuble et supprime tous les logements des immeubles supprimés.

2. On vous demande de gérer toutes les contraintes d'intégrité référentielle de la base sous forme de triggers. Citez les différents triggers qui permettront de le faire en présentant pour chaque table concernée la liste des triggers à prévoir (sans préciser le code).

Table IMMEUBLE

- Trigger sur Modification
- Trigger sur suppression

Table LOGEMENT :

- Trigger sur Insertion,
- Trigger sur modification
- Trigger sur suppression

Table EMPLOYE

- Trigger sur Modification
- Trigger sur Suppression

Table LOUER

- Trigger sur Insertion
- Trigger sur Modification