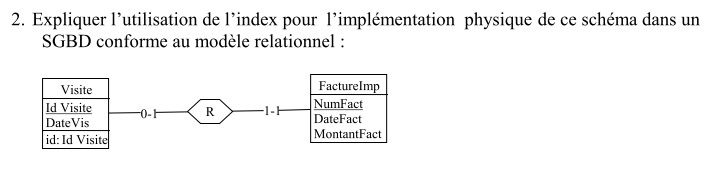


* Spécialisation
  + 
  + **+ :** Il n’y a pas d’attribut « type » dans le T.E Client, et donc pas de contrainte additionnelles à gérer sur le type.
  + **- :** il faut ajouter une contrainte d’exclusion, c-à-d, qu’un Client ne peut être que d’un genre (soit ClientAssujetti soit ClientNonAssujetti)
* Représentation des valeurs distinctes
  + Intérêt ?
    - Utile si on veut réutiliser le T.E TypeAssujetti (très peu probable);
    - Utile si on veut contrôler le fait que le numéro de TVA soit présent dans la DB ;
  + 

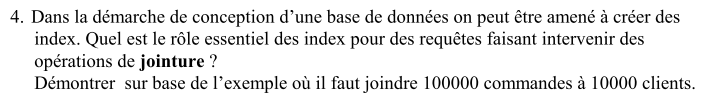


Quand on a une relation telle, une fois que l’on passe au modèle relationnel on perd la 0-1 qui devient une 0-N. Du coup il s’agit de mettre un index sur la clé étrangère dans la 1-1. Et mettre cet index comme UNIQUE ce qui permet de conserver l’information 0-1 dans la logique





* Le schéma permet de retrouver le représentant au moment de la commande mais aussi de retrouver le représentant actuel pour la région. Pas de redondances présentes.



Sans index,

* Avec l’algorithme de boucles imbriquées : on aurait NbLecture = 10000 + 10000 \* 100000
* Avec l’algorithme tri-fusion : Fonction de l’algorithme
  + Tirer R1
  + Trier R2
  + Fusionner
    - Le calcul devient :
      * Card(R1) \* Log(Card(R1)) + (10.000 \* 4)
      * Card(R2) \* Log(Card(R2)) + (100.00 \* 5)
      * Card(R1) + Card(R2) (110.000)
    - Nous avons donc un total de 650.000 lectures

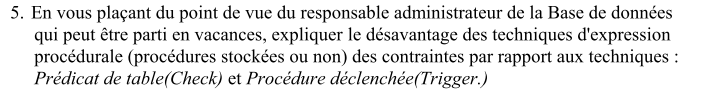
Avec index,

Soit R1 indexé et R2 non indexé :

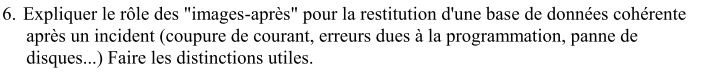
Si une table est indexée, il suffit de lire la table non indexée et d’aller rechercher son correspondant dans la table indexée ce qui nous fait un nombre de lectures : 10000 + 100000

Soit R1 et R2 indexé :

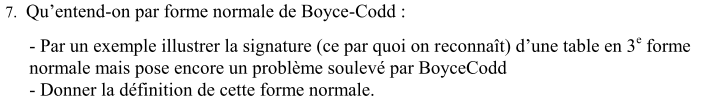
1. On fusionne les deux index pour disposer des couples d’adresses des couples joignants. Cela peut se faire en majeure partie en mémoire.
2. Il suffit d’accéder alors directement aux lignes concernées par les adresses.
   1. Nombre de lectures ? 10 000 + 100 000
   2. Indexé sur la FK pour plus de rapidité



* Contrairement aux triggers et aux Check, on doit faire appel aux procédures stockées
* Gros problème de portabilité



* L’ « image-après » permet de sauvegarder des données en cas de panne. Elle est réalisée après chaque opération sur la base de données. Le but d’ « image-après » est de pouvoir restaurer à l’identique les données comprises dans la base de données avant la panne. S’il y a eu une image avant on peut comparer les modifications apportées lors d’une opération entre les 2 images et ainsi revenir à la situation de départ en cas de soucis.

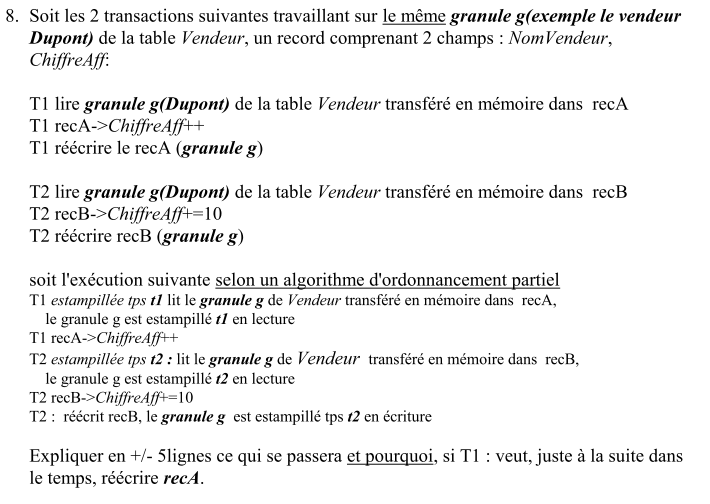


* **Définition :** une relation sous 1FN est sous forme de normale de BOYCE-CODD (FBNC) si tout déterminant strict y est identifiant.
* On sait différencier la forme de BOYCE-CODD de la 3FN car FBNC élimine les dépendances fonctionnelles à l’intérieur de l’identifiant.
  + Exemple :

|  |  |  |
| --- | --- | --- |
| **Matière** | **Etudiant** | **Professeur** |
| Histoire | Eric | Prof 1 |
| Math | Michel | Prof3 |
| Histoire | Jean-Luc | Prof1 |

🡺 On peut isoler la dépendance directe Professeur 🡪 matière

* Cependant on peut encore observer 1 anomalie en s’attardant à l’extension de la relation :
* Si plus d’un étudiant suit la même matière d’un même professeur, le fait élémentaire que ce professeur donne cette matière est représenté plusieurs fois.

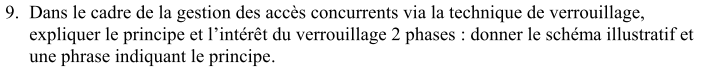


Granule : unité de base de données dont l’accès est contrôlé individuellement : l’enregistrement, la page, la table ou toute la BD. (Unité de base de données sur laquelle on va vérifier les accès concurrents).

Estampiller : les estampilles permettent d’installer un ordre total entre les actions et ainsi de les sérialiser. L’estampille est donc un identifiant unique des transactions qui permet en plus de les ordonner.

Si t1 : veut mettre à jour le granule g, l’opération est refusée, la transaction t1 est reprise.

Le système contrôle les conflits au fur et à mesure de l’exécution. ***Ecrire*** granule g de **t1** est refusé parce que t1 est estampillé avec un tps plus petit que celui de la dernière estampille en lecture sur le granule g. (Condition suffisante) et plus petit que la dernière estampille en écriture. La transaction T1 va être relancée. « t1 màj Granule g » aurait été refusé même s’il était arrivé avant « t2 màj granule g » car le granule g a été relu entre-temps par t2.

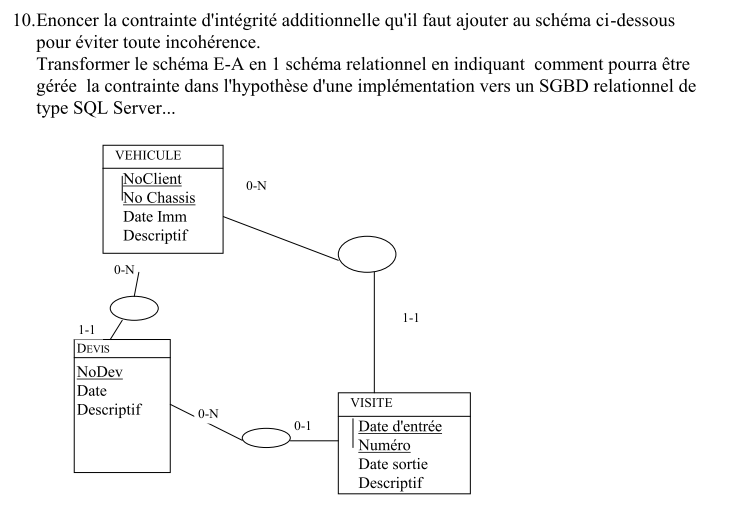


Une transaction est à verrouillage 2 phases si:

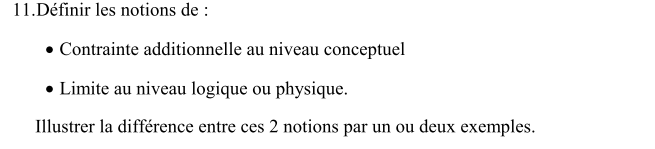
* Avant opération, transaction demande un verrou sur l’élément
* Quand le processus a fini il unlock mais ne demandera plus de verrou

Il y a un intérêt sécuritaire de manipulation d’un même objet. La première phase est donc: acquisition des verrous et seconde phase relâchement.

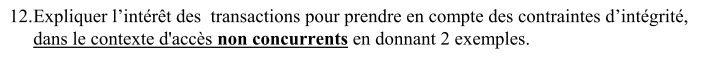
Attention soit il relâche les verrous au fur et a mesure soit il relâche tous les verrous d’un coup.



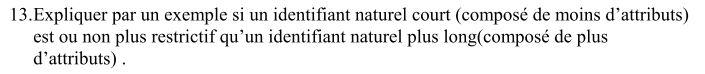
* **Schéma relationnel**
* 
* **Contrainte d’intégrité additionnelle à ajouter :**  le véhicule de la visite doit être le même que pour le véhicule du devis.
* **Comment gérer ce type de contrainte ?** Via une procédure déclenchée (trigger).



* **Contrainte additionnelle au niveau conceptuel :** 
  + Une contrainte additionnelle est une spécificité de l’énoncé qu’il n’est pas possible de représenter sur le schéma. Or cette exigence pourra se révéler par la suite particulièrement importante. Lors de l’implémentation physique, chaque contrainte sera examinée et une décision sera prise quant au fait de comment l’implémenter.
* **Limite au niveau logique ou physique :**
  + Lorsque toutes les contraintes auront été mises en évidences, on devra faire clairement le bilan pour chaque contrainte conservée aux stades conceptuel et logique en se posant les questions suivantes :
    - Prise en charge pas le SGBD ? Si oui, comment ?
    - ET / OU prise en compte dans le programme (vérification à plusieurs niveaux)
    - Si pas pris en compte par le SGBD alors => limite d’implémentation.
* **Exemple :**
  + MySQL ne gère pas les check



* Dans le cas d’une contrainte de cardinalité 1-1 1-N
* Dans le cas d’une contrainte de redondance FactureImpayee.Montant et Client.Solde



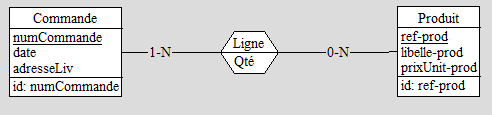
Plus un ID est court (= moins il contient de champ), moins il y a de risques d'avoir des dépendances fonctionnelles au sein de l'identifiant.

Id’(Emprunt) = Exemplaire, Emprunteur, Date => (moins restrictif) car l’exemplaire peut-être emprunter plusieurs fois par jour par différents emprunteur (concept d’exemplaire donc impossible).

Id’(Emprunt) = Exemplaire, Date => (plus restrictif) car l’exemplaire ne peut-être emprunter qu’une seule fois par jour. (Je ne pourrai pas emprunter le même livre qu’un autre emprunteur le même jour te devient donc plus restrictif).

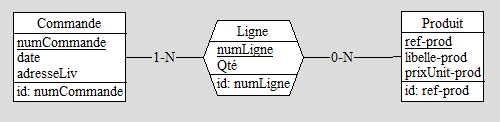


***Autre exemple :***



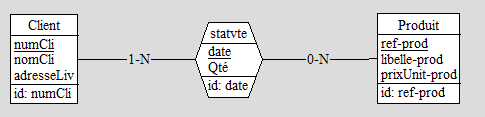
Id(ligne) : commande, produit

* On ne peut enregistrer qu’une fois un produit. Il faut modifier la ligne.



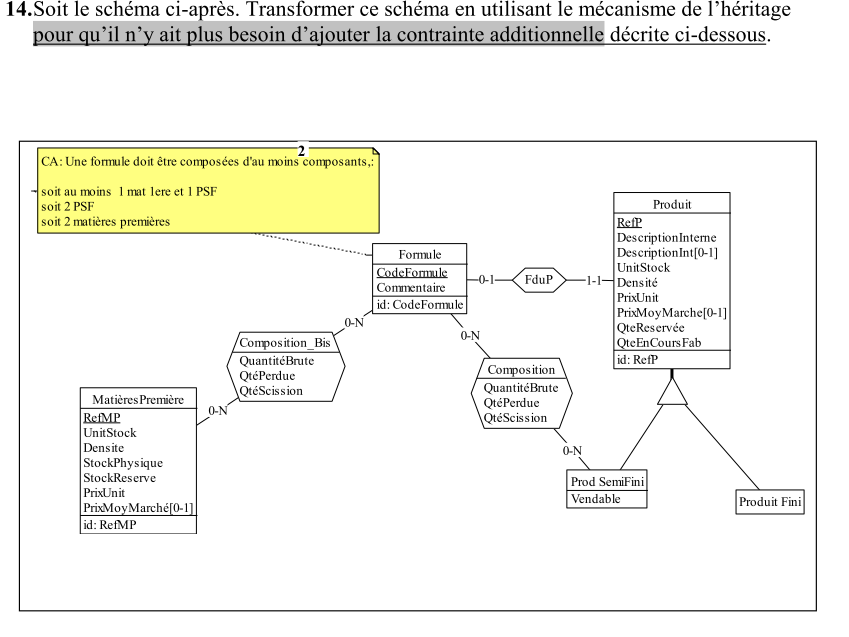
Id(ligne) : commande, numLigne

* Un article peut apparaitre plusieurs fois

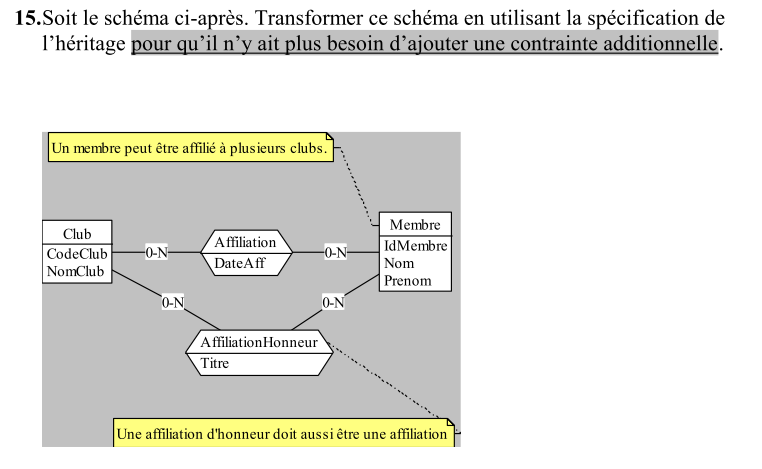


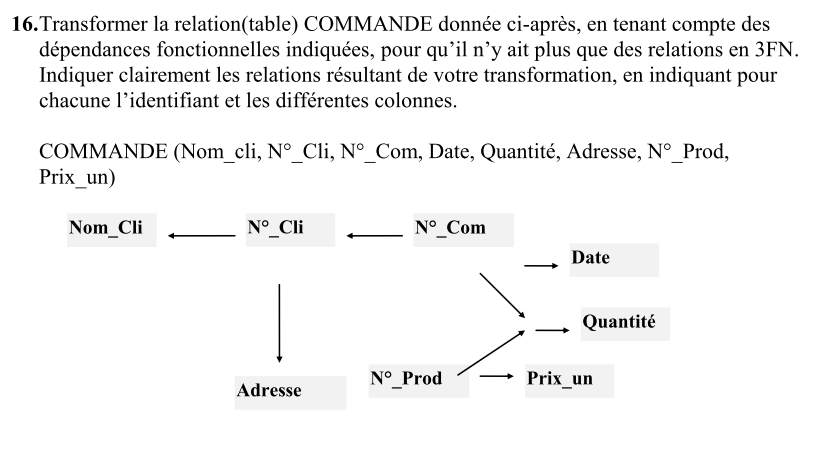
Id(ligne) : client, date, produit

* Une seule ligne le même jour
* On peut distinguer les différents achats d’un même produit









**N°\_Com**

**N°\_Cli**

**Nom\_Cli**

**N°\_Prod**

**Date**

**Quantité**

**Prix\_un**

**Adresse**

COMMANDE (N°\_Com, Date, N°\_Cli, Nom\_Cli, Adresse)

LIGNE (N°\_Com, N°\_Prod, Quantité)

PRODUIT (N°\_Prod, Prix\_un)

**N°\_Com**

**N°\_Cli**

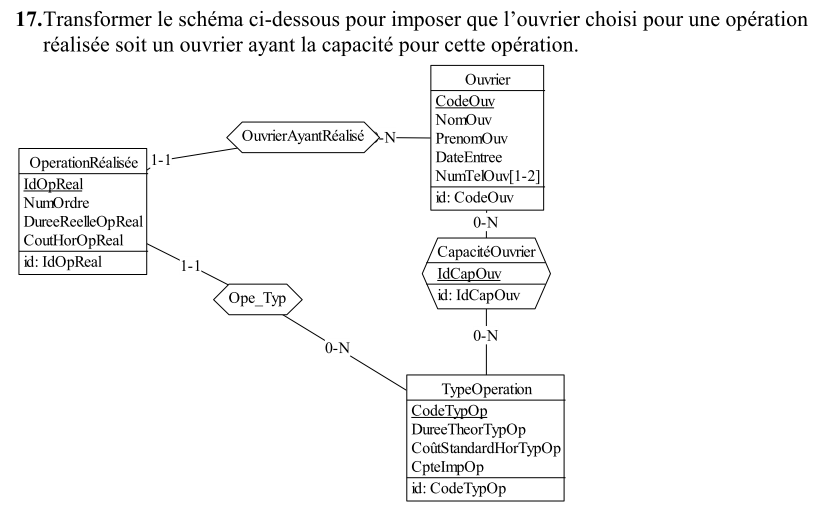
**Nom\_Cli**

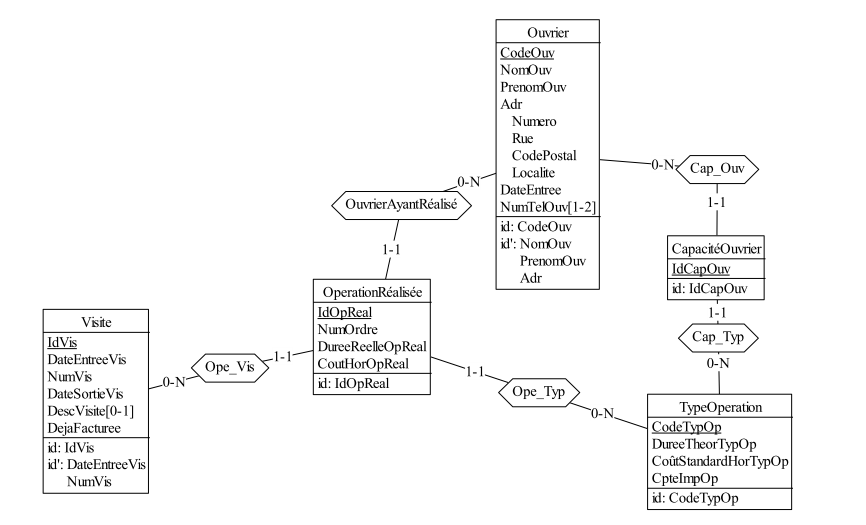
**Date**

**Adresse**

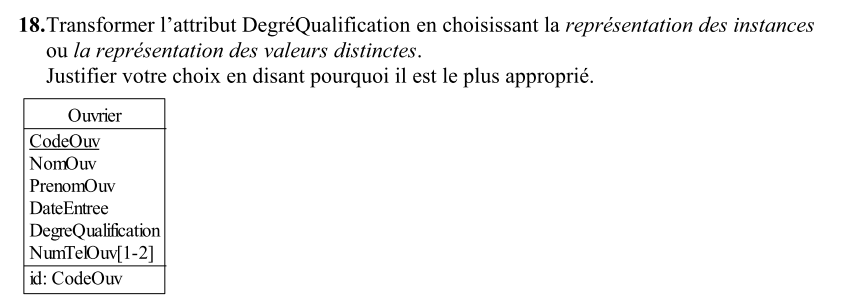
COMMANDE (N°\_Com, N°\_Cli, Date)

CLIENT (N°\_Cli, Nom\_Cli, Adresse)



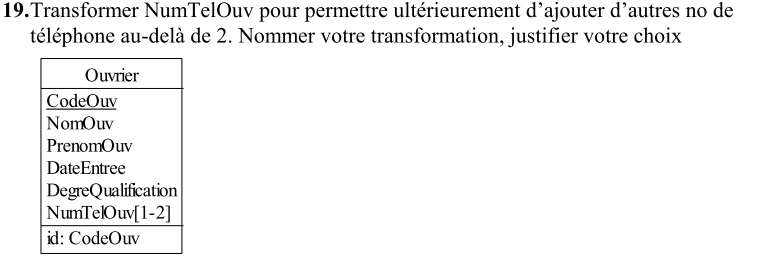


* Décision de contrôle donc on doit relier OpérationRéalisée à CapacitéOuvrier directement et supprimer le lien entre OpérationRéalisée et Ouvrier et OpérationRéalisée et TypeOpération.





* Permet de contrôler de la qualification de l’ouvrier, on ne peut pas mettre n’importe quelle valeur pour la qualification.
* Plusieurs ouvrier peuvent avoir la même qualification, il est donc bon d’en faire une seule table pour pouvoir la réutiliser.
* Permet de lister toutes les qualifications.

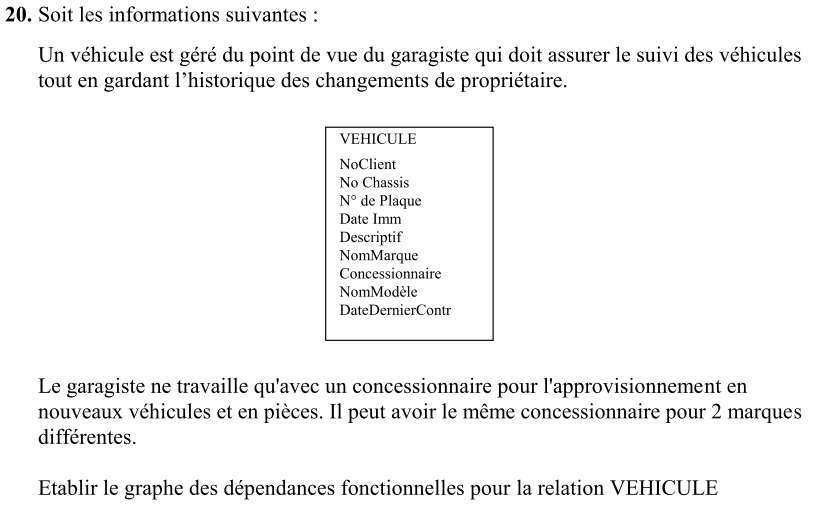


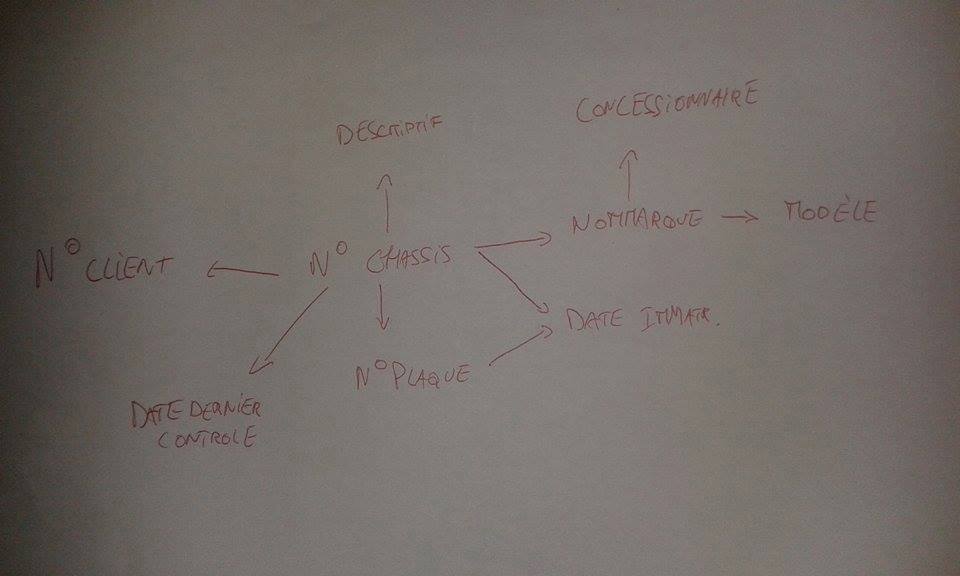


Représentation des instances.

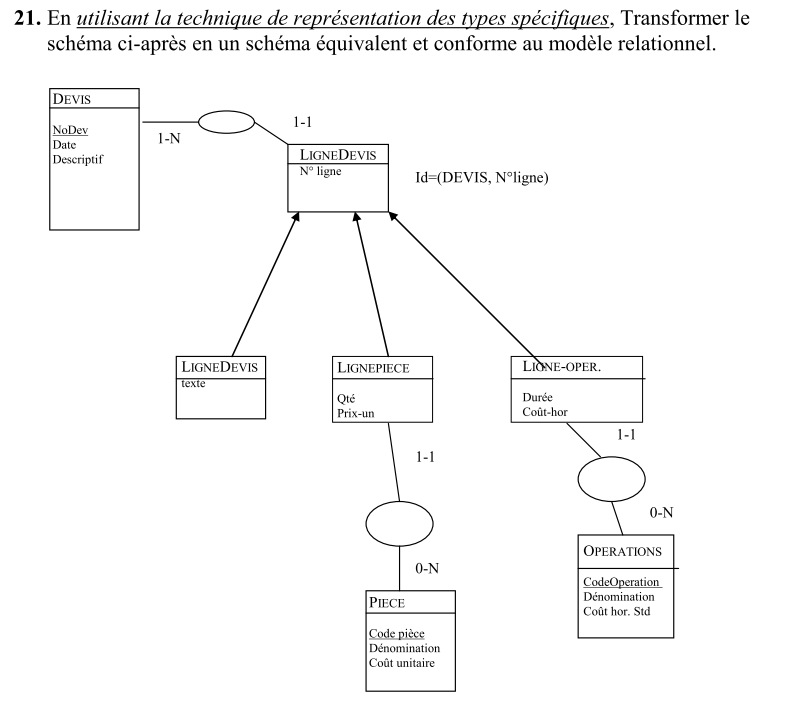
Un numéro de téléphone ne peut appartenir qu’à un seul client.

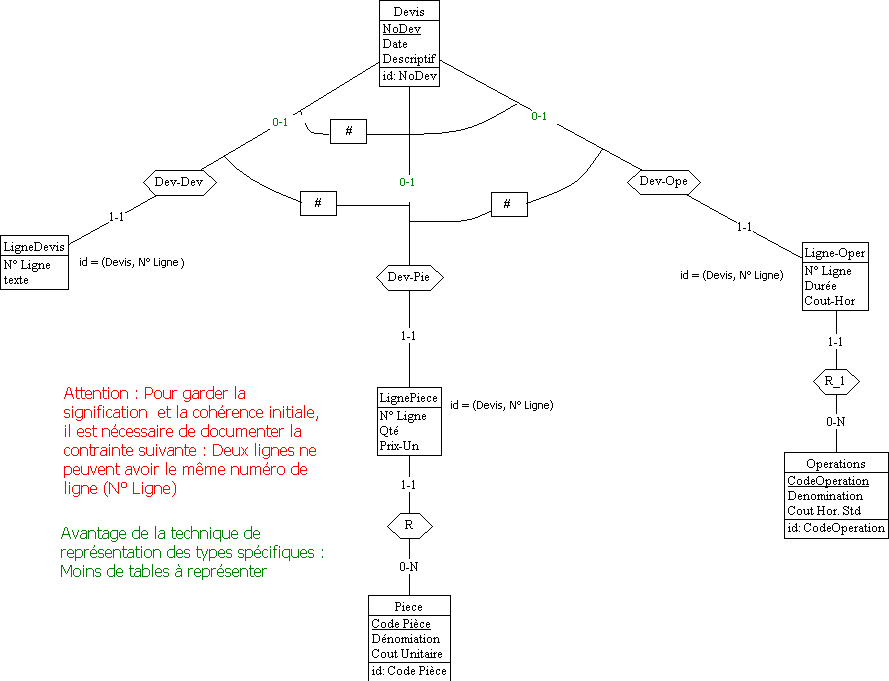
On peut ajouter des numéros à un client par la suite car on ne connait pas tous les numéros de téléphone existant, pas de contrôle pas les valeurs.



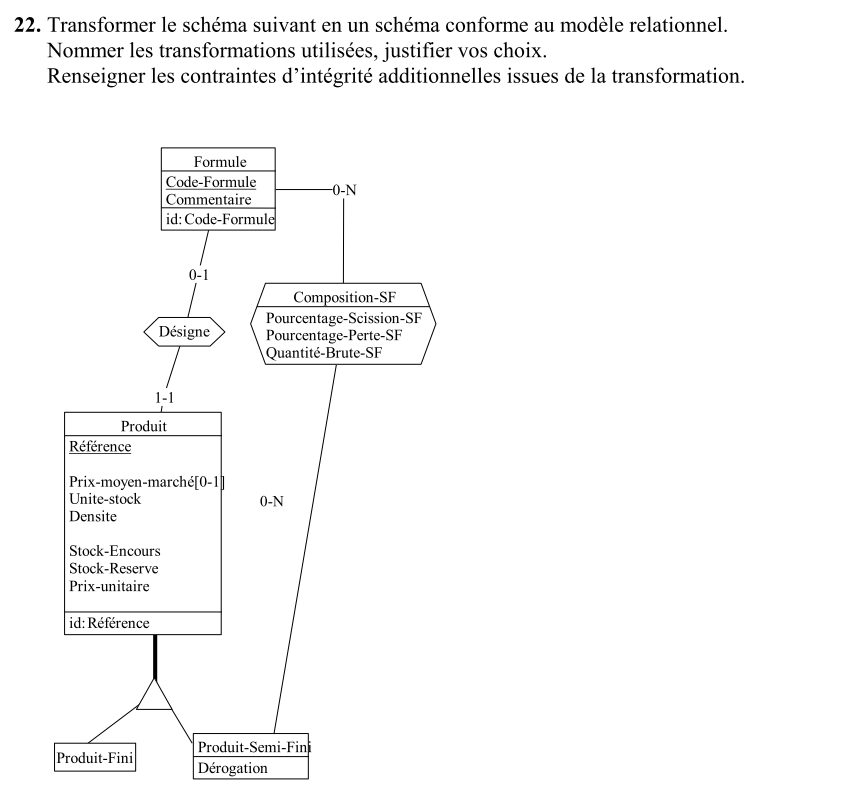


Effacer la dépendance entre N°client et N°chassis





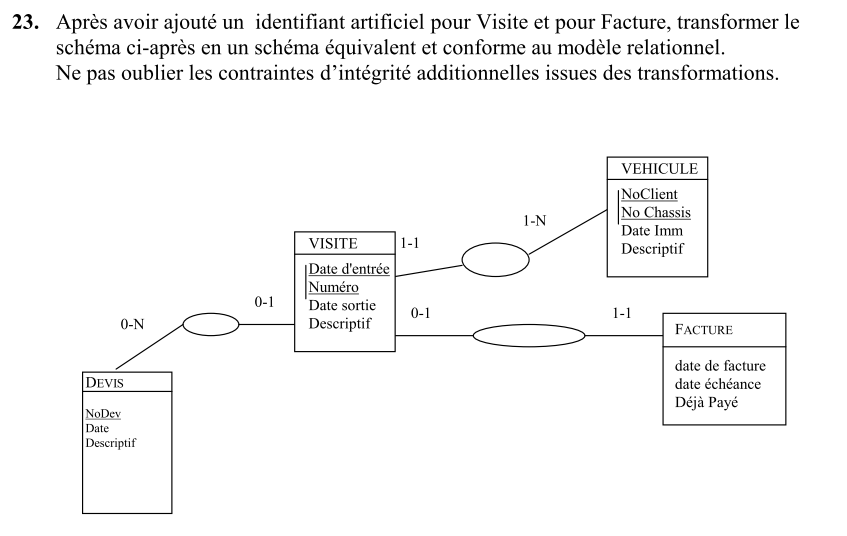
Ce ne sont pas des 0-1 du côté du devis mais des 0-N. Y’a pas de contrainte d’exclusion, mais CA sur le fait qu’un DEVIS doit posséder au moins une des trois lignes.





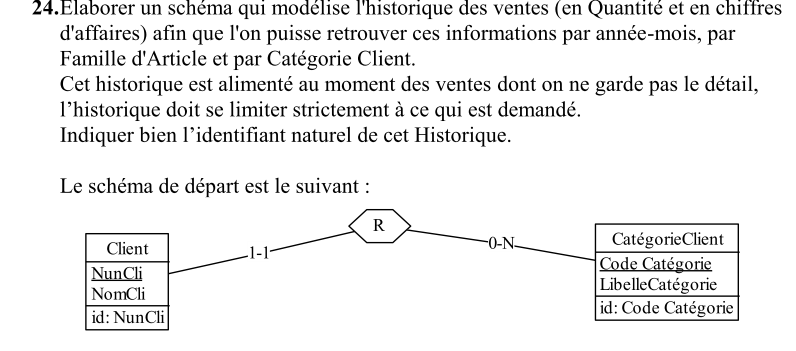
Ne pas oublier la contrainte additionnelle d’exclusion : «un produit peut être soit un produit fini, soit un produit semi fini ». De là en découle le fait que l’id du produit doit être celui du produit fini et du produit semi fini.

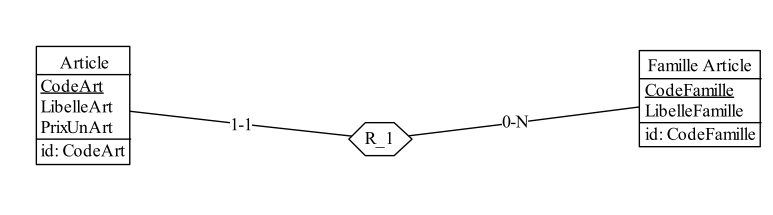
Modélisation des relations « is-a ». Représentation des types spécifiques.

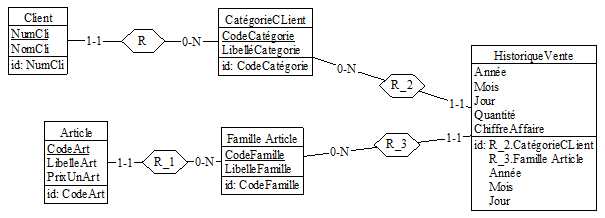


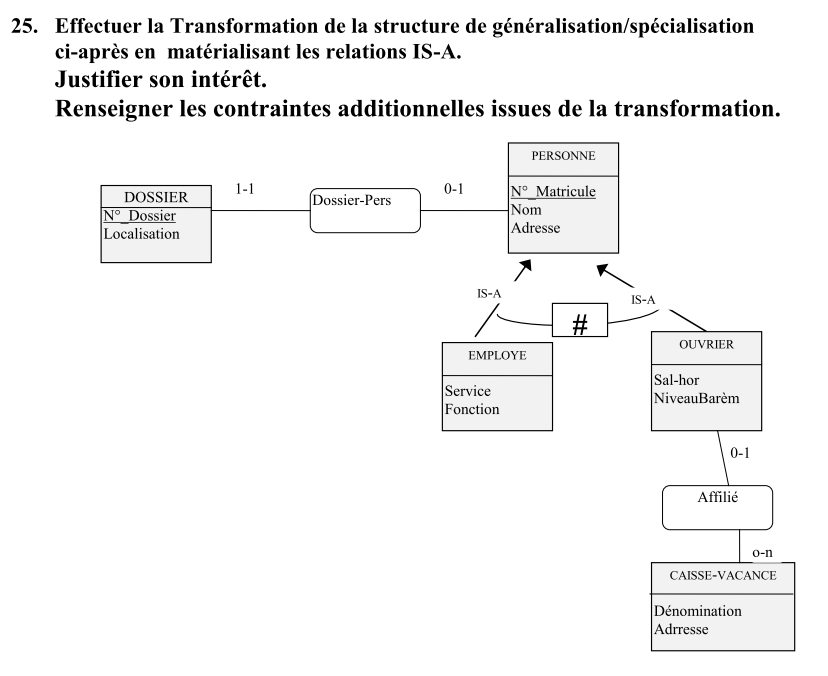


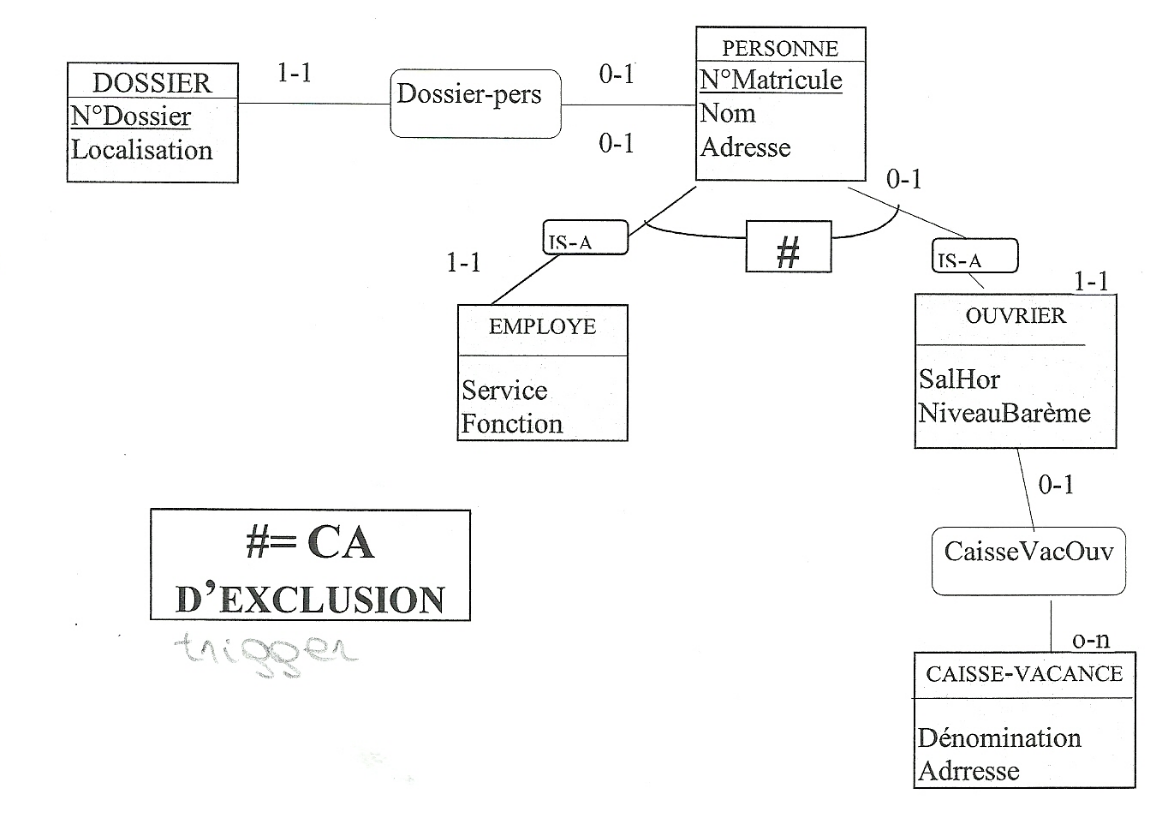
AJOUTER LES CONTRAINTES ADDITIONNELLES :



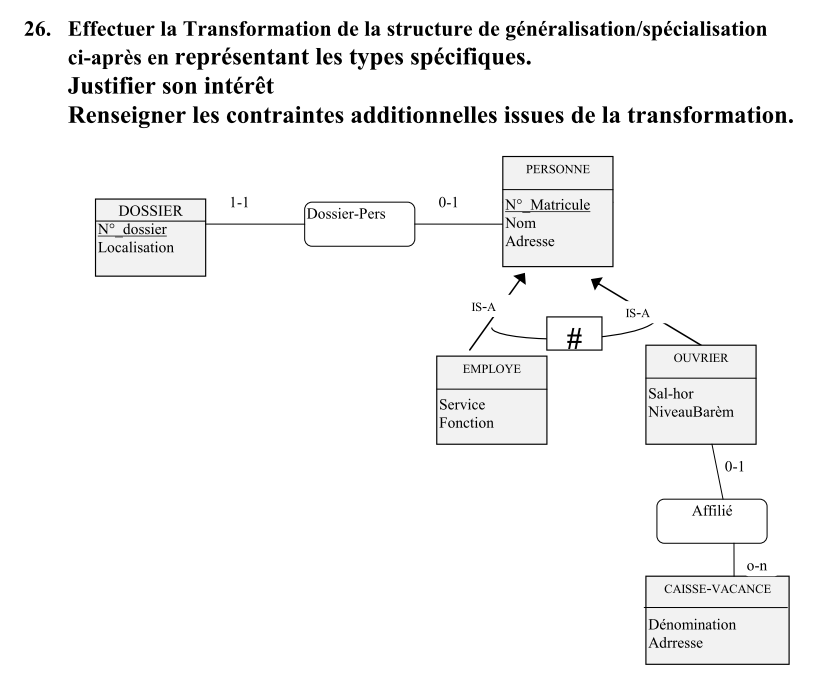


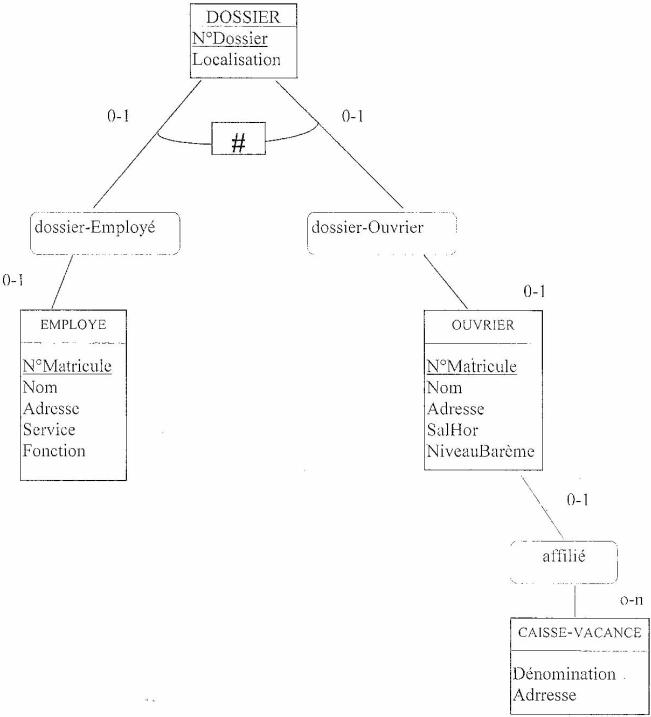




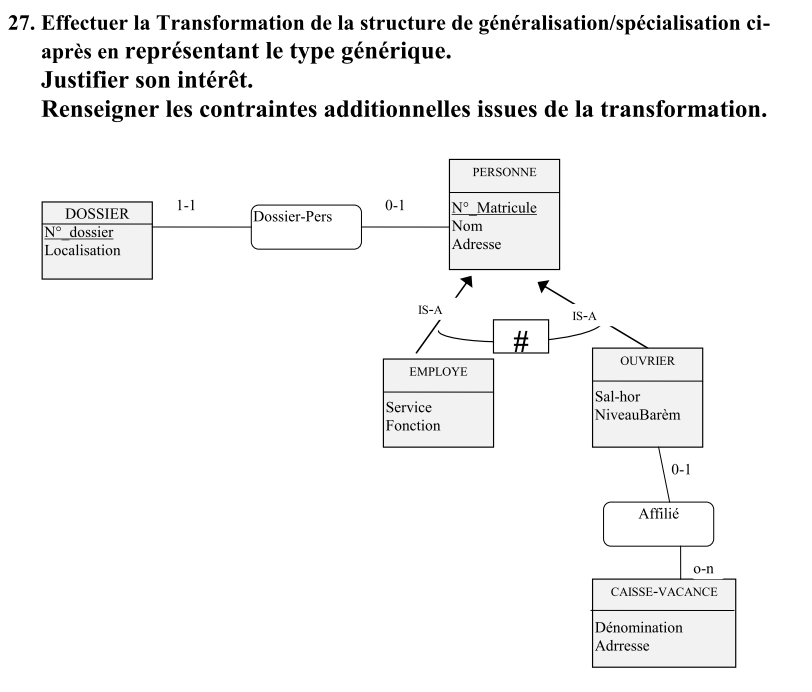
****

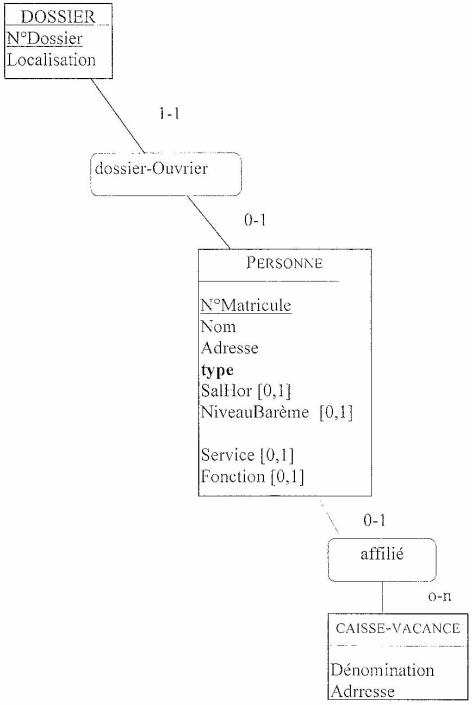
* Intérêts : facilité d’implémentation
* Contraintes additionnelles : CA d’exclusion

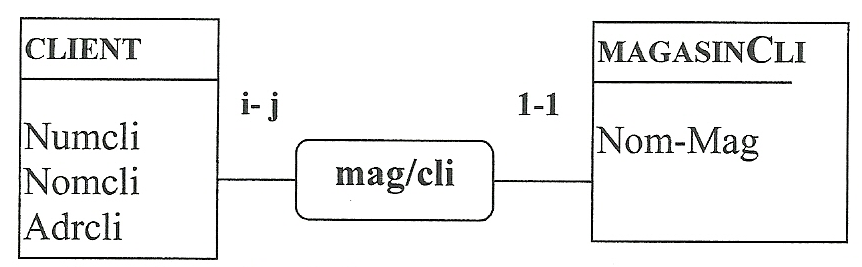
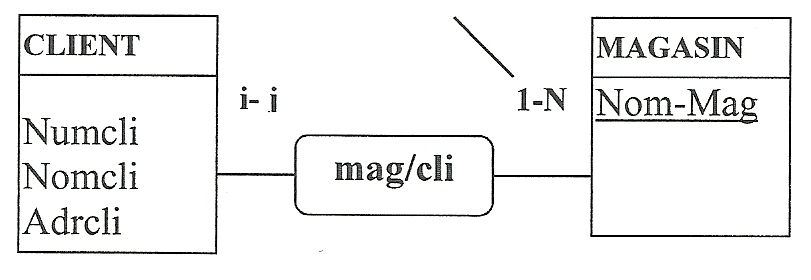
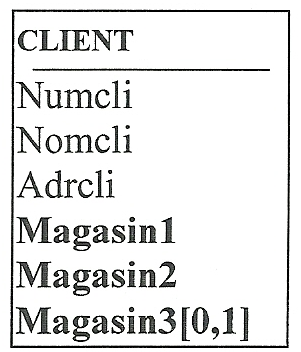
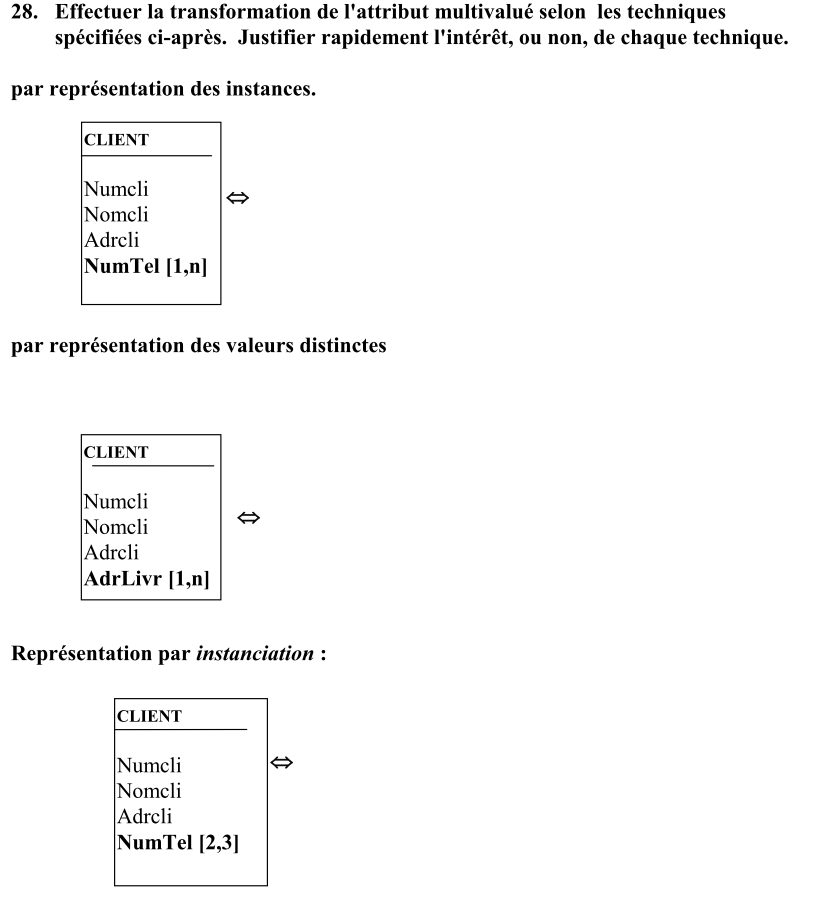


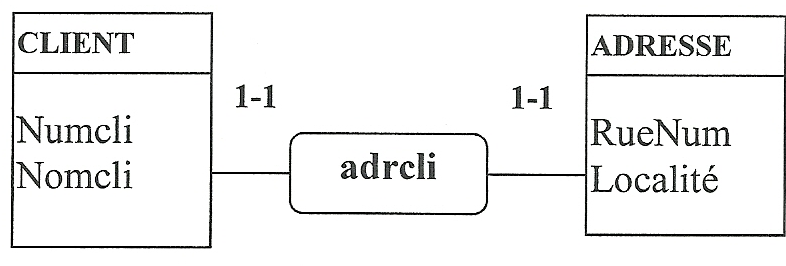
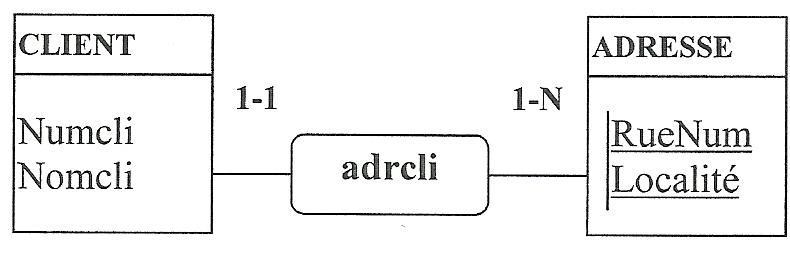
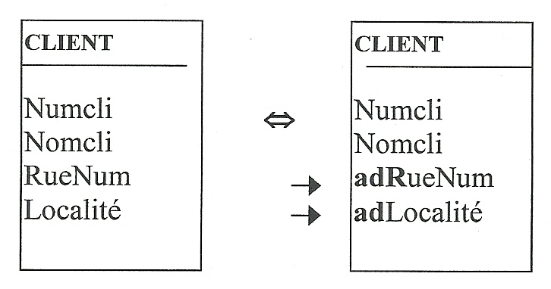
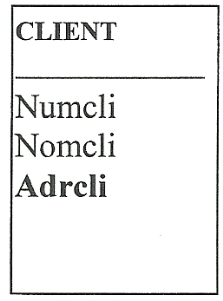
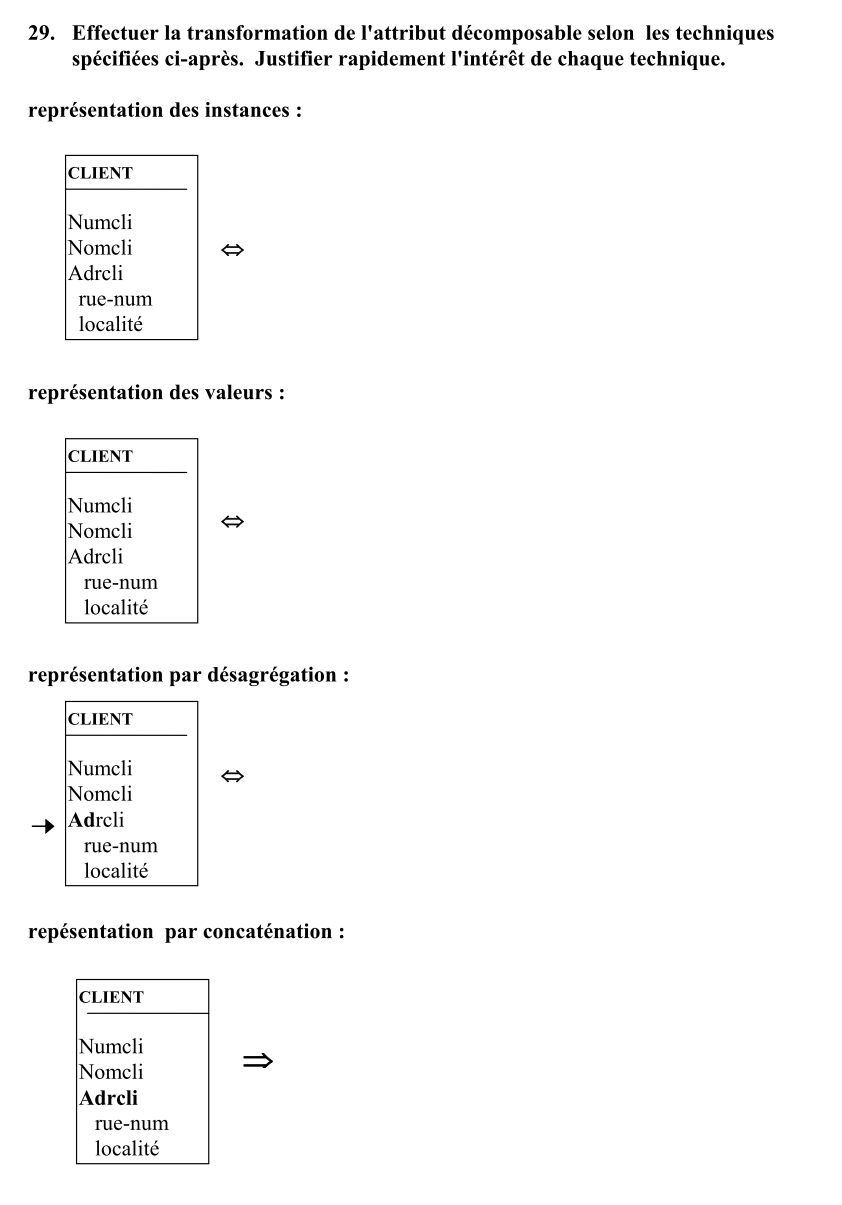
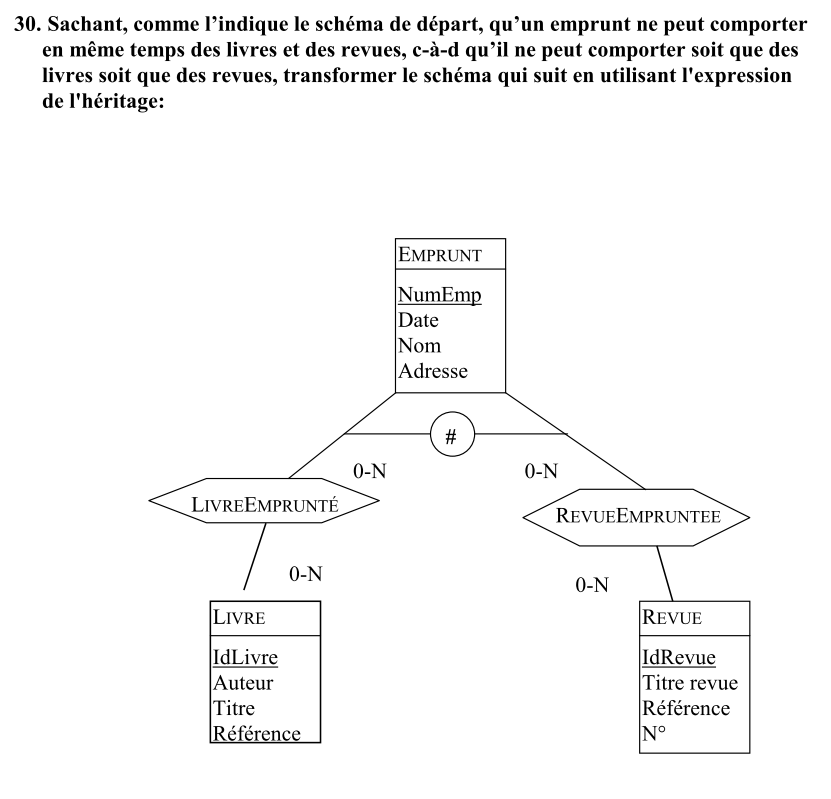
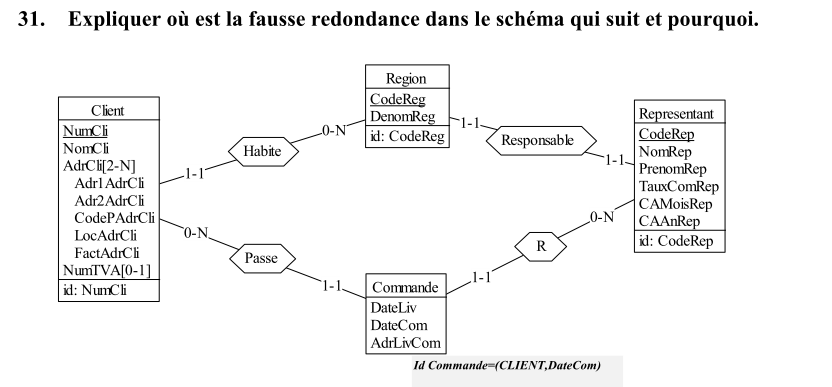
****

* Intérêt : représentation relativement proche de la représentation de l’héritage de départ.
* Contraintes additionnelles :
  + Le numéro de matricule doit être unique entre ouvrier et employé
  + Ajouter une contrainte d’exclusion : « soit … soit … »

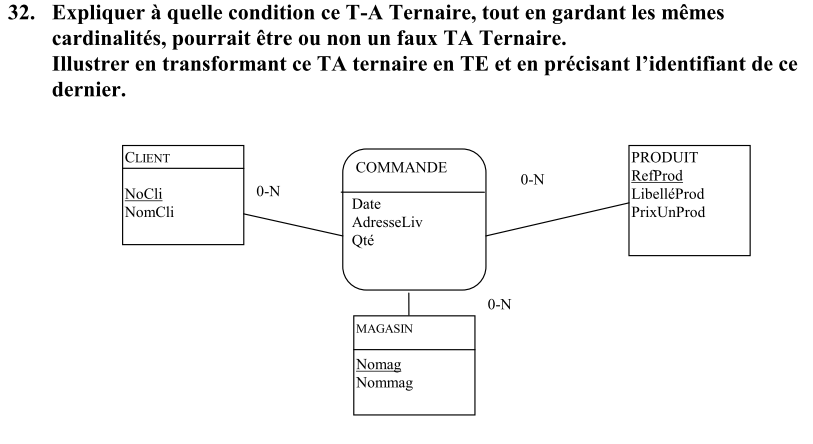
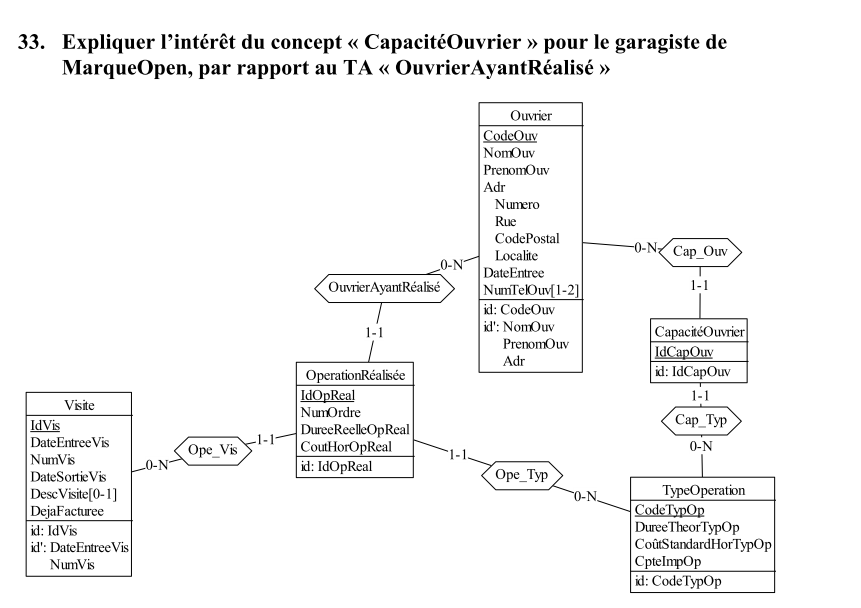


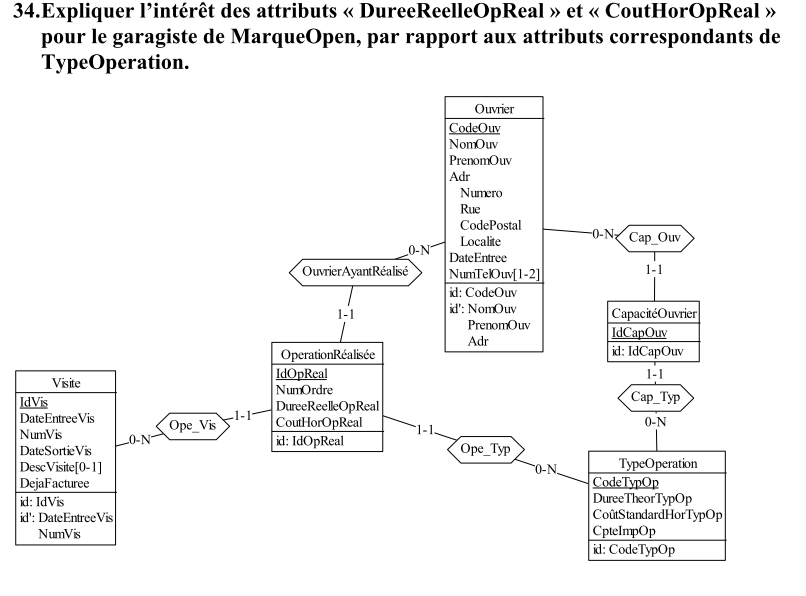
* Intérêt : représentation qui permet de n’avoir plus qu’une seule entité ***PERSONNE*** (ouvrier ou employé selon l’indication du type)
* Avantage :
  + - * + Moins de tables
        + Plus performant
        + MAIS plus de contraintes
      * Contraintes additionnelles :
        + Si l’attribut ***TYPE*** équivaut à OUVRIER alors les colonnes SalHor et NiveauBarème doivent être remplies. Un T.E « CAISSE-VACANCE » sera également crée
        + Si l’attribut ***TYPE*** équivaut à EMPLOYE alors les colonnes Service et Fonction devront être remplis.

****

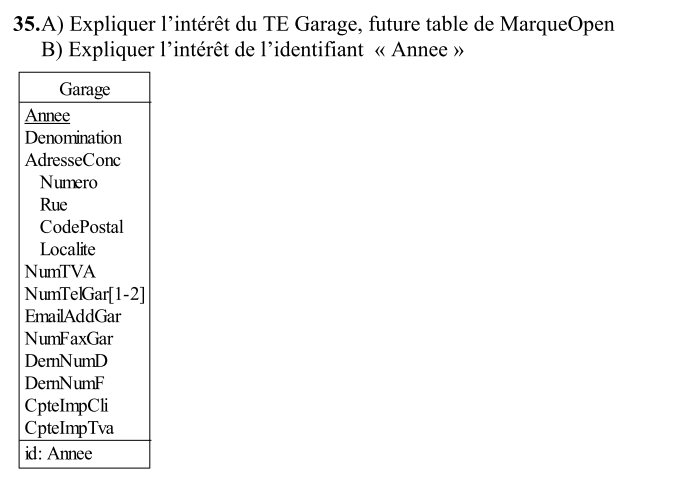
* ****
* Pour la dernière solution : « concaténation », parfois bien utile pour la performance mais non réversible !
* 
* 
* 

***CF QUESTION 3***

* 
* id(Commande) = Client, Produit, Magasin
* Il apparaît plus clairement après transformation que le concept de Commande est proche du concept de Ligne commande, c’est un cas particulier où la commande n’a qu’une seule ligne.
* Dans cette interprétation, il s’agissait d’une vraie ternaire, mais il faut remarquer que le schéma ternaire de départ pouvait cacher un concept de commande comportant plusieurs lignes. Il ne s’agirait plus alors d’une vraie ternaire.
* 
* Impose que l’ouvrier choisi pour une opération requiert ce niveau d’expertise, et qu’il ne soit pas incompétent dans la tâche qu’on lui incombe. Retrouver les opérations que l’ouvrier est capable de faire. Une CA doit être vérifiée lorsqu’on assigne une opération à un Ouvrier.



* DureeReelleOpReal : Durée réelle, effective de l’opération réalisée
* CoutHorOpReal : Le « vrai » cout de l’opération, ce qu’elle a vraiment couté quand le client à payer.
* Le prix et la durée réelle peuvent être différente du cout / durée théorique (une opération pour changer une roue prend théoriquement, 30 min mais le mécanicien l’a réalisé en 25min, ce qui deviens la durée réellement passé à changer la roue. Son cout est de 15€ théoriquement mais le mécanicien lui en demande réellement 10€).



1. Pouvoir consulter les données du garage
2. Gardez les paramètres du garage d’une année à l’autre. Une année identifie un état du garage, ex : « en 2012, le garage à changer d’adresse », « en 2014, le garage à changer d’adresse mail ».

Si l'état change on ajoute une nouvelle entrée avec l'année courante du changement et permettre de faire l'historique des changements dans le garage.

