PROJET BD

ANALYSE

Propriétés :

{IdProduit, Intitule, PrixCourant, Description, URL, NomCategorie, NomCatSup, Date, Heure, Prix, IdCompte, Mail, MDP, Nom, Prenom, Adresse, NomCaracteristique, TypeCaracteristique, ValeurCaracteristique}

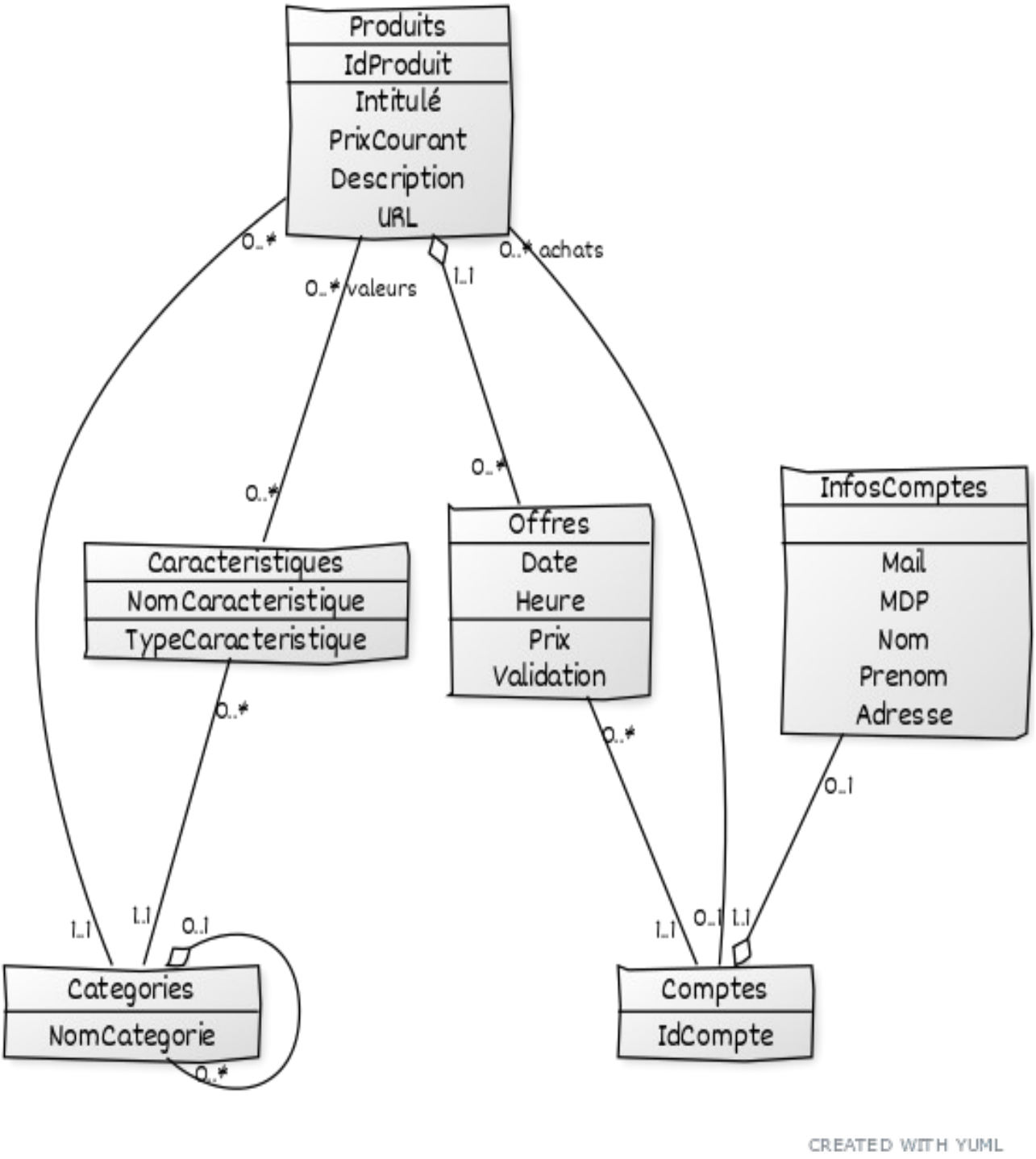
Contraintes :

Dépendances fonctionnelles	Contraintes valeur	Contraines multiplicité
IdProduit → Intitule, PrixCourant, Description, URL	PrixCourant > 0	**IdProduit** -|-> IdCompte
IdProduit → NomCategorie	Pas de contrainte	Pas de contrainte
{Date, Heure, IdProduit} → Prix	Prix > PrixCourant	Pas de contrainte
{Date, Heure, IdProduit} → IdCompte	Pas de contrainte	Pas de contrainte
NomCaracteristique → TypeCaracteristique	Pas de contrainte	**{NomCaracteristique, IdProduit}** -|-> ValeurCaracteristique
NomCaracteristique → NomCategorie	Pas de contrainte	Pas de contrainte
NomCategorie contrainte de multiplicité	Pas de contrainte	**NomCategorie** -|-> NomCatSup
IdCompte contrainte de multiplicité	Pas de contrainte	**IdCompte** -|-> Mail, MDP, Nom, Prenom, Adresse

* Contraintes contextuelles pour IdCompte :

- « RGPD : suppression des valeurs dans InfosComptes associées à IdCompte »
- « RGPD : conservation de IdCompte »

Diagramme entité-relationnel :



Passage au relationnel :

Entités faibles

- InfosComptes(PK : IdComptes | Mail, MDP, Nom, Prenom, Adresse)
- Caracteristiques(PK : NomCaracteristique | TypeCaracteristique, FK:NomCategorie)
- Offre(PK : Date, Heure, FK : IdProduit | Prix, FK : IdCompte)

Nouvelles entités créées

- Achats(PK : IdProduit | FK : IdCompte)
- CaracteristiquesProduit(PK : IdCaracteristique, IdProduit | ValeurCaracteristique)
- HeritageCategories(PK : NomCategorie | NomCatSup)

Les dépendances relationnelles et contraintes de multiplicité restent les même.

Formes normales :

FN 1 :

- immédiat ici

FN 2 :

- FN 1
- pour Offres : tout attribut dépend bien de {Date, Heure, IdProduit} et non de la date ou du produit seulement
- pour CaracteristiquesProduit : la valeur dépend à la fois du produit et de la caractéristique demandée
- pour les autres relations, on a directement la forme normale

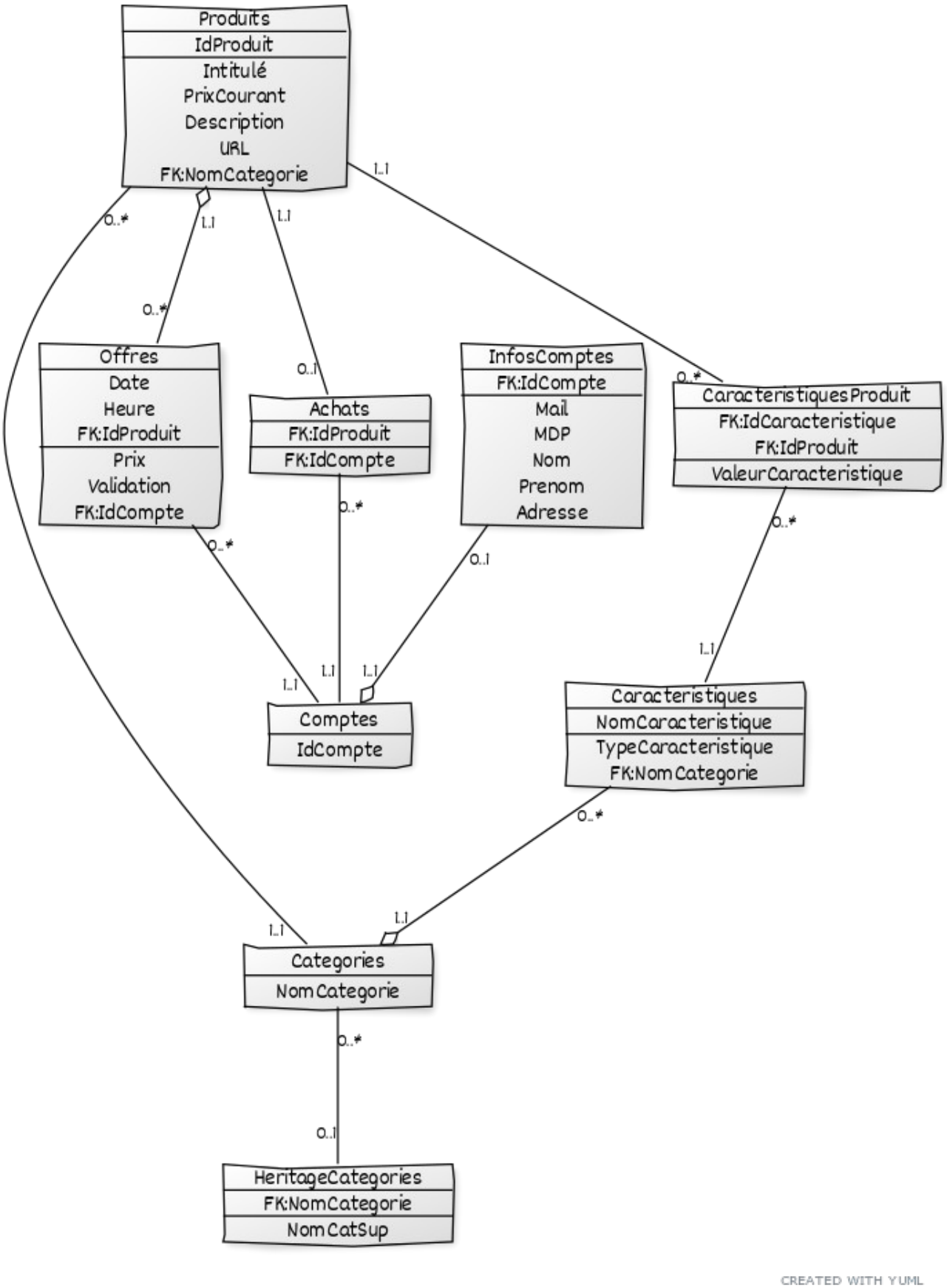
FN 3 :

- FN 2
- 1 seule clef pour beaucoup de relations et tous les attributs dépendent ici pleinement et directement des clefs lorsqu'il y en a plusieurs ({Date, Heure, IdProduit})

Boyce-Codd-Kent :

- FN 3
- ici, toute dépendance fonctionnelle contient effectivement une clef, toujours à gauche

Schéma relationnel de GANGE :



Difficultés rencontrées et choix (analyse) :

Voici une description succinte des différentes questions auxquelles nous avons apporté une solution dans notre projet. Tout d'abord, le choix des entités *Produits*, *Offres*, *Comptes* et *Categories* découlait directement du sujet, *Offres* dépendant de *Produits*. Mais il fallait pouvoir, pour un produit donné, appartenant à une catégorie, lui assigner des valeurs propres à cette catégorie. La solution choisie a été de créer une entité faible *Caracteristiques* liée à *Produits* et *Categories* (voir schéma 1), dont la relation avec *Produits* contient les valeurs des caractéristiques d'un produit. On créera alors une nouvelle entité *CaracteristiquesProduit* lors de la traduction qui, à un produit et une caractéristique donnée assignera une valeur.

La question des catégories et sous-catégories se posait ensuite. Nous avons pour cela mis l'entité *Categories* en relation avec elle-même, une catégorie pouvant (ou pas) être une sous-catégorie d'une autre catégorie. Ce choix mènera lors de la traduction en relationnel à la création d'une nouvelle entité *HeritageCategories* liée à *Categories*. Deux possibilités semblaient alors possibles : pour une sous-catégories, lorsque l'on définissait ses caractéristiques, soit on recopiait toutes les caractéristiques de la catégorie mère auxquelles on ajoutait les caractéristiques propres à la sous-catégorie, ce qui représente un gain de temps, mais plus de mémoire allouée (redondance) ; soit on n'ajoutait que les caractéristiques propres à la sous-catégorie et, lors de la recherche de caractéristiques, on remontait aux caractéristiques de la catégorie supérieure, et ce ainsi de suite jusqu'à avoir remonté toute l'arborescence des catégories. C'est cette solution que nous avons décidé d'adopter afin d'éviter toute redondance.

Pour le droit à l'oubli, il fallait pouvoir supprimer les informations sur un client tout en gardant un client type dans notre modèle. Pour cela, il fallait pouvoir supprimer les attributs du compte tout en gardant un *IdCompte*. Nous avons alors choisi de créer une entité faible, *InfosComptes* liée à *Comptes*, tel que la suppression des données entraine la suppression des valeurs des attributs dans la relation *InfosComptes*, mais la conservation de l'identifiant dans *Comptes*.

Implémentation des fonctionnalités :

Système de recommandations :

Pour implémenter le système de recommandations demandées par le Client, on se base sur deux requêtes SQL pour les deux sources de recommandations spécifiées. Les transactions employées ici sont avec les paramètres par défaut du JDBC, c’est-à-dire que l’autocommit est activé car il n’est pas nécessaire de le contraindre ici.

1. Recommandations basées sur le nombre d’offres n’ayant pas conclu a un achat du client

```
SELECT PRODUITS.Categorie FROM OFFRES
JOIN PRODUITS ON OFFRES.IdProduit = PRODUITS.IdProduit
WHERE IdCompte=%d AND NOT EXISTS
(SELECT * FROM ACHATS
WHERE ACHATS.IdCompte=PRODUITS.IdProduit AND
ACHATS.IdProduit=OFFRES.IdCompte)
GROUP BY PRODUITS.Categorie
ORDER BY COUNT(PRODUITS.IdProduit) DESC
OFFSET 0 ROWS FETCH NEXT %d ROWS ONLY
```

On va donc récupérer les catégories des produits pour lesquels l'idCompte actuel a fait des offres mais n'a pas conclu l'achat. Le résultat est trié par nombre d'offre par catégorie décroissant et on limite directement le nombre de catégories retournées (à l'aide d'une constante définie dans le *JDBC MAX_PERSONNAL_CAT_RECOMMANDATIONS)*.

2. Recommandations basées sur le nombre d'offres totales par catégories

SELECT PRODUITS.Categorie FROM OFFRES
FULL JOIN PRODUITS ON OFFRES.IdProduit = PRODUITS.IdProduit
GROUP BY PRODUITS.Categorie
ORDER BY COUNT(PRODUITS.IdProduit) DESC

Sur le même principe que précédemment, on trie par nombre d'offres totales par catégories (sans filtrer pour l'idCompte cette fois). Seule particularité, on utilise ici un FULL JOIN pour récupérer des catégories même si aucune offre n'a encore été placée dessus (ce qui n'aurait pas pu être le cas avec un JOIN simple).

Récupération caractéristiques

Par construction du schéma relationnel, on peut directement récupérer les caractéristiques sur la table CAR_PRODUITS à l'aide de la requête suivante :

SELECT CAR_PRODUITS.NomCar, CAR_PRODUITS.ValCar, PRODUITS.Intitule FROM CAR_PRODUITS
JOIN PRODUITS ON CAR_PRODUITS.IdProduit = PRODUITS.IdProduit
WHERE CAR_PRODUITS.IdProduit = %d

Droit à la suppression des données personnelles
Pour exercer le droit RGPD de l'utilisateur, notre implémentation utilise une simple requête supprimant le n_uplet lié à l'identifiant du compte concerné dans la relation INFOS_COMPTES :
DELETE FROM INFOS_COMPTES WHERE IDINFOSCOMPTE = idAccount (formaté au préalable).

Petite particularité : pour s'assurer de la bonne suppression des informations personnelles, on va alors vérifier la non-présence de l'identifiant compte avec : *SELECT COUNT(*) FROM INFOS_COMPTES WHERE IDINFOSCOMPTE= idAccount*.

Si le COUNT(*) vaut 0 c'est qu'on a bien effectué notre suppression.

Connexion à la base de données :

Cette transaction n'est pas très compliquée, il suffit de vérifier dans la table des utilisateurs, qu'il existe bien un utilisateur avec le mail et mdp donnés :

```
SELECT * FROM INFOS_COMPTES WHERE Mail = '%s' AND MdP = '%s'
```

Affichage des produits des catégories :

L'affichage des produits est associé à plusieurs contraintes. Il faut que les produits soient triés par ordre décroissant du nombre d'offres, et alphabétique sur leurs noms. Mais étant donné que nous avons un fonctionnement avec une forme de récursion sur les catégories,

nous souhaitions implémenter une solution pour récupérer tous les produits associés à la catégorie actuelle mais également ceux associés à ses catégories filles (et petites filles etc). Pour cela, il est possible d'effectuer une requête SQL récursive, mais ne maîtrisant pas ces notions, nous avons préféré effectuer la récursion en Java. Il faut également que les produits n'aient pas été achetés.

```
select PRODUITS.IdProduit, PRODUITS.Intitule from PRODUITS JOIN OFFRES on
PRODUITS.IdProduit = OFFRES.IdProduit WHERE PRODUITS.categorie = '%s'
AND NOT EXISTS (SELECT * FROM ACHATS WHERE ACHATS.IdProduit=PRODUITS.IdProduit)
GROUP BY PRODUITS.IdProduit,PRODUITS.Intitule
ORDER BY COUNT(*) DESC,PRODUITS.Intitule
```

Cette requête effectue ce qui est demandé pour une catégorie. Nous sommes ainsi capables d'obtenir l'ordre souhaité. Cependant, en introduisant la récursion, nous perdons cet ordre (car la récursion s'effectue côté Java, si nous le faisions côté SQL, nous pourrions ordonner après). Une autre solution possible aurait été de récupérer la liste des catégories en Java récursivement, puis d'effectuer un `WHERE PRODUITS.categorie IN (liste_catégories)`. Cette solution n'est pas viable à plus grande échelle. La meilleure solution aurait donc été la récursion en SQL que nous ne maîtrisons pas.

Affichage de la fiche complète d'un produit :

Deux parties sont nécessaires à cela : la récupération des informations stockées dans la table Produit, ainsi que les caractéristiques qui lui sont associées. Il n'y a pas de problème particulier avec des requêtes.

```
SELECT * FROM PRODUITS WHERE IdProduit = '%d'
SELECT CAR_PRODUITS.NomCar, CAR_PRODUITS.ValCar, PRODUITS.Intitule FROM
CAR_PRODUITS JOIN PRODUITS ON CAR_PRODUITS.IdProduit = PRODUITS.IdProduit WHERE
CAR_PRODUITS.IdProduit = %d
```

Structure du démonstrateur :

Le démonstrateur s'organise en Menus. Ces derniers sont des sortes de pages du démonstrateur qui permettent certains types d'actions. Dans notre cas, il y en a pour trois choses : se connecter, parcourir les catégories et enchérir sur un produit.

Commandes générales et connexion :

- exit : permet de fermer la connexion avec la base de données
- con : permet de se connecter en temps qu'utilisateur de Gange ou administrateur
- decon : déconnexion
- ret : retourne au menu précédant

Commandes du catalogue (parcours par catégories) :

- del : permet d'exercer son droit à l'oubli en supprimant son compte et toutes les informations associées
- show <nameProduct> : Affiche la fiche descriptive du produit
- cc <Category> : change de catégorie (et affiche les produits associés par conséquent)
- parent : remonte à la catégorie parente
- achats : affiche les achats remportés par l'utilisateur connecté
- cp <idProduct> : choisit le produit afin d'effectuer une enchère dessus

Infos produit et enchères :

- bid : permet d'obtenir le prix courant du produit et d'enchérir dessus.

Effectuer des enchères :

Il s'agit de différentes requêtes assez basiques à combiner. Notamment, il faut vérifier que l'enchère proposée est correcte. Puis, il faut pouvoir insérer une nouvelle offre :

```
INSERT INTO OFFRES(IdProduit, IdCompte, DateHeure, Montant) VALUES (?, ?, ?, %.2f)
```

Puis mettre à jour le prix courant du produit :

```
UPDATE PRODUITS SET PRIXCOURANT = '%s'
WHERE IDPRODUIT = '%s'
```

Ainsi que si l'achat est effectué, ajouter une ligne à sa table :

```
INSERT INTO ACHATS (IdProduit, IdCompte) VALUES (%s, %s)
```

Gestion de la concurrence sur les enchères :

Sur les enchères, on ne souhaite pas autoriser les lectures non répétables. En effet, il serait frustrant pour un utilisateur de voir un produit à un certain prix, et au moment de l'enchère, se voir refuser d'effectuer une offre car un autre utilisateur l'a fait avant. Nous imposons donc que les transactions soient sérialisables. Lorsqu'un utilisateur commence une enchère sur un produit, la transaction est changée de mode. Cette gestion est nécessaire et l'autocommit doit être désactivé, ce qui permet d'ailleurs d'effectuer des rollback en cas de problème sur la requête.

Bilan du projet :

Pour conclure, le rendu fourni avec cette documentation répond bien aux demandes du projet, que ce soit pour les systèmes d'offres / achats, les recommandations et la RGPD. L'organisation pour y arriver à été la suivant : la première semaine a été consacrée à l'analyse et la modélisation du problème par tout le groupe, puisque cette partie s'est avérée indispensable pour tout le reste du projet. Par la suite, deux membres du groupes se sont consacrées au passage en relationnel et à l'implantation de la base de données, tandis que deux autres se sont occupés de l'analyse et l'implémentation des fonctionalités, et enfin, une dernière personne s'est chargée de la partie en Java (même si nous nous tenions informés de nos avancements plutôt que de faire une mise en commun à la toute fin). Cette organisation ainsi que les efforts apportés par chacun en plus des séances de TD ont permis de mener à bien ce projet tout en respectant la contrainte du temps, et permettant de pouvoir poser les bonnes questions au client / expert pour continuer dans cette lancée. Enfin, c'est grace à l'investissement sérieux et régulier de chacun que ce projet a pu être réalisé dans les meilleurs conditions.