GRENOBLE INP Ensimag UGA

Ensimag 2ème année

2021-2022

Rapport Projet BDD

Application Gange

Martin Gangand
Robin Mallecourt
Reda Jaoudar
Safouane Bendou
Yunfe Girault

Contexte de travail

Pour ce projet, nous avons immédiatement partagé les tâches pour ne pas perdre de temps. L'Analyse/Conception a été faite par Yunfe, Martin et Robin, tandis que Safouane et Reda devaient réfléchir à propos de la partie Java ainsi qu'apporter un regard critique sur ce qui était fait lors de la conception et l'analyse .

Une bonne communication entre nous a été la clé pour un travail bien organisé. Nous avons dû organiser des séances supplémentaires de travail pour le projet BDD en étant soit en présentiel soit en distanciel.

Analyse du problème

Pour l'analyse du problème, plusieurs choix ont dû être faits :

- Un produit a nécessairement au moins un couple caractéristique/valeur
- Le client est séparé de l'utilisateur sur le site qui lui est associé (utile pour le droit à l'oubli)
- Un client peut faire des offres mais il peut ne pas en faire
- La colonne AchatReussi vaut 0 si l'offre n'a pas abouti à un achat, et vaut 1 si l'offre faite s'est conclue avec un achat (cette colonne simplifie beaucoup les requêtes SQL pour la recommandation personnalisée)

L'analyse approfondie du problème du site internet Gange nous a amené à avoir au fil des essais plusieurs solutions différentes :

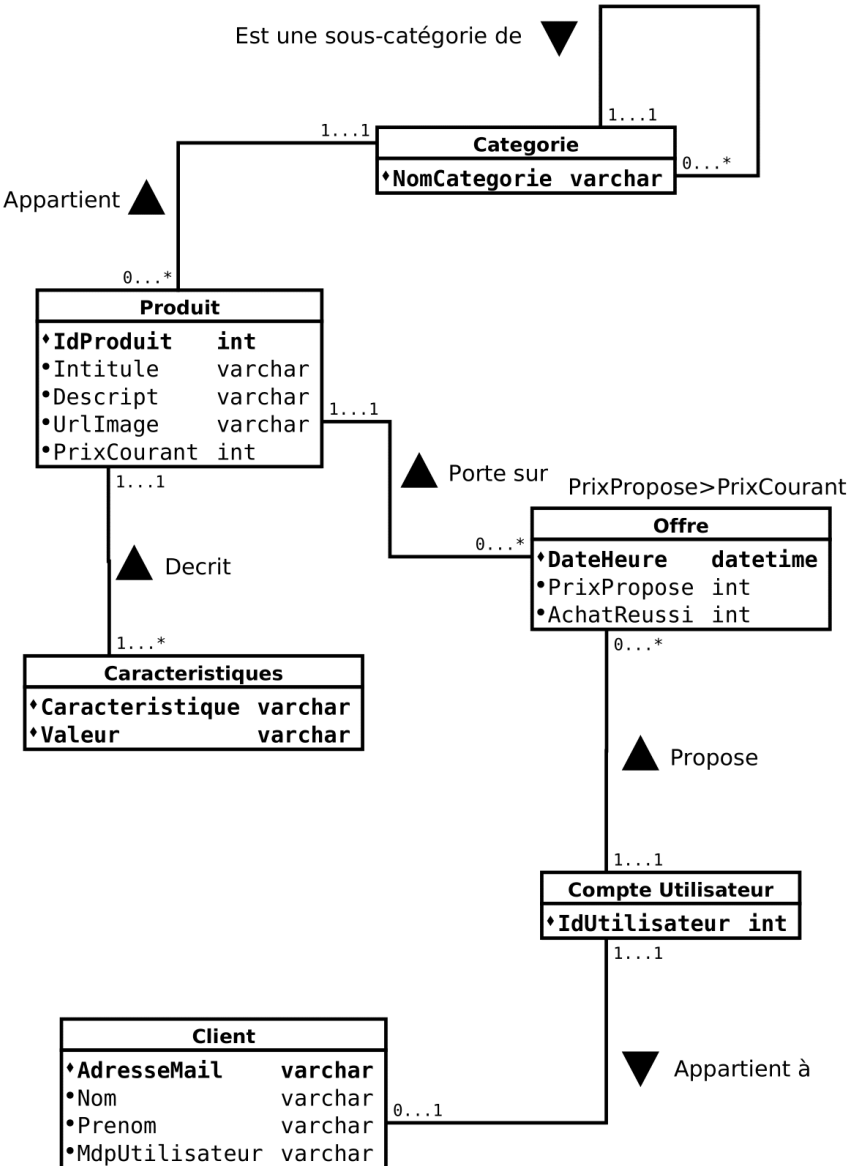
Nous avons représenté les dépendances fonctionnelles, les contraintes de valeur, de multiplicité, et contextuelles dans le tableau suivant.

Dans ce tableau, nous avons séparé les lignes en les regroupant par futures relations et les noms des relations sont entre « [] ».

Propriétés	DF	Contraintes valeurs	Multiplicité	Contraintes contextuelles
[Produits] IdProduit Intitule PrixCourant Description Image	IdProduit → Intitule, PrixCourant, Description, Image,	IdProduit > 0 PrixCourant >0	IdProduit -|>> IdOffre IdProduit -->> (Caractéristique, Valeur)	
[Offre] DateHeure PrixPropose AchatReussi	DateHeure → PrixPropose, AchatReussi	PrixPropose > 0 AchatReussi ∈ {0, 1}		PrixPropose > PrixCourant
[Utilisateur] IdUtilisateur		IdUtilisateur > 0	IdUtilisateur -|->> IdOffre	
[Client] AdresseEmail, Nom Prenom MdpUtilisateur AdressePostale	AdresseEmail → Nom,			

	Prenom, MdpUtilisateur, AdressePostale			
[Categorie] NomCategorie CategorieMere	NomCategorie → CategorieMere		NomCategorie -|->> IdProduit	
[Caractéristique] NomCaractéristique ValeurCaracteristique	IdProduit → (NomCaractéristique, ValeurCaracteristique)			

Conception Entités/Associations en UML



Les choix faits :

- On suppose qu'un couple (Caractéristique, Valeur) est unique pour un produit
- Un compte utilisateur est lié à **au plus** un client (il n'est pas lié à un client lorsque le client utilise son droit à l'oubli)
- La table catégorie pointe sur elle-même car une catégorie possède **une et une seule mère** (si la catégorie est une catégorie « racine » elle a comme mère elle-même)
- Une catégorie peut n'avoir aucun produit dedans (notamment les catégories mères car les produits sont dans les catégories filles)

Traduction en relationnel

Nous avons finalement les relations suivantes où les attributs en orange sont des Foreign Keys et ceux soulignés sont les Primary Keys :

Produit (IdProduit, Intitule, PrixCourant, Descript, UrlImage, NomCategorie)

Offre (DateHeure, PrixPropose, AchatReussi, IdProduit, IdUtilisateur)

Utilisateur (IdUtilisateur)

Client (AdresseMail, Nom, Prenom, MdpUtilisateur, AdressePostale, IdUtilisateur)

Caractéristique (NomCaracteristique, ValeurCaracteristique, IdProduit)

Categorie (NomCategorie, NomCategorieMere)

Analyse des Formes Normales

Forme Normale 1 (1NF) :
Une relation R est en première forme normale (1FN) si et seulement si tout attribut ne peut contenir que des valeurs atomiques et non Null (Null n'est pas une valeur !)

Ce qui est le cas dans toutes les relations, chaque attribut ne contient que des valeurs atomiques et non Null.
Par exemple, l'ajout de "caractéristique" comme relation au lieu de juste la considérer comme attribut dans la relation produit, est dû au fait que cet attribut ne contient pas de valeurs atomiques.

Forme Normale 2 (2NF) :
Une relation R (et ses DF F) est en deuxième forme normale (2FN) si et seulement si :
1. Elle est en première forme normale (1FN).
2. Tous les attributs non-clés sont pleinement dépendants de chacune des clefs.

Ainsi, on pourra dire qu'une relation ayant une clé formée d'un seul attribut est donc en deuxième forme normale.
On analyse les FN pour la relation Produit et Client par exemple. En fait, la relation Produit est une relation qui a juste un seul attribut clé donc elle est bien de forme 2FN. C'est aussi le cas pour la relation Client, les attributs non-clés (Nom, Prenom, MdpUtilisateur, AdressePostale) dépendent pleinement de l'attribut clé (AdresseMail).

3. Forme Normale 3 (3FN) :
Une relation R (et ses DF F) est en troisième forme normale (3FN) si et seulement si :
1. Elle est 2FN
2. Tout attribut non-clé ne dépend pas d'un ensemble d'attributs qui n'est pas une clef.

Toutes les relations sont 2FN et les attributs non-clés ne dépendent que des attributs clés (Pas de transitivité), donc elles sont 3 FN.
Par exemple, pour la table Client, les attributs non clé (Nom, Prenom, MdpUtilisateur, AdressePostale) dépendent uniquement de la clé AdresseEmail .
Et pour la relation Produit, les attributs non-clés (Intitule, PrixCourant, Descript, Image) ne dépendent que de la clé IdProduit.

4. Forme normale de Boyce-Codd-Kent (3FNBCK) :
Une relation R (et ses DF F) est en forme normale de Boyce-Codd-Kent (3FNBCK) si et seulement si :
1. Elle est 3FN
2. Pour toute DF ($X \rightarrow Y$) non triviale (i.e. $Y \not\subseteq X$), X contient une clef de R.

En fait, toutes les relations sont 3FN et tous les attributs non-clés ne sont pas source d'une dépendance fonctionnelle, donc ils sont 3FNBCK.
À titre d'exemple, IdProduit → Intitule, PrixCourant, Description, Image. En effet, tous les attributs non-clés de la table Produit dépendent uniquement de l'Idproduit.
De même pour la relation Client, AdresseEmail → Nom, Prenom, MdpUtilisateur, AdresseUtilisateur.

PS :

La contrainte « PrixPropose > PrixCourant » n'a pas été implantée en SQL car nécessitait l'utilisation de Trigger dans les Transactions.

Analyse des fonctionnalités

Il est nécessaire de maintenir une cohérence dans notre base de données. Prenons un exemple concret pour bien comprendre. Lorsque deux clients A et B sont connectés sur Gange, il est possible qu'ils souhaitent faire une offre sur un même produit. Supposons que A propose une offre de 20 € sur une lampe torche dont le prix courant est de 15 €. Peu après B veut faire une offre sur ce même produit, il propose alors une offre à 18 €. Il est important de prendre en compte l'offre de A, de sorte que dès que B fait son offre il doit être averti que le prix qu'il propose est inférieur au prix courant du produit.

Il est alors intéressant d'utiliser les transactions et d'utiliser "set autocommit off". C'est alors à nous de placer des "commit" pour signaler les changements à la base de données aux bons endroits. Dans notre exemple, une proposition d'offre se fait en 2 temps. Nous ajoutons d'abord l'offre avec le prix proposé dans la table OFFRE, puis nous mettons à jour le prix courant du produit dans la table PRODUIT (si le prix est accepté évidemment). Nous plaçons donc un commit après avoir exécuté ces 2 requêtes. Ainsi, lorsque B propose son offre, la proposition de A a été prise en compte et donc on signale à B que son offre n'est pas valide.

Mode d'emploi du démonstrateur

Tout d'abord, en lançant l'application Gange, l'utilisateur a le choix de se connecter à son compte ou de s'en créer un s'il n'en a pas. Lorsqu'il est connecté, un menu s'affiche avec les actions qu'il peut effectuer, il doit alors taper le numéro correspondant dans le terminal. Parmi les possibilités, l'utilisateur peut parcourir les catégories et les sous-catégories. Il peut aussi choisir d'afficher les recommandations générales des catégories ou ses recommandations personnalisées, la fiche d'un produit (qui inclut les caractéristiques), ainsi que la liste des produits qui sont encore disponibles à la vente. Ensuite, il peut choisir de proposer une offre, d'effacer son compte de la base de données, et enfin de fermer l'application. Ainsi nous couvrons les aspects essentiels d'une application de gestion de vente de produit.

Nous avons également créé un compte administrateur avec des droits supplémentaires. L'adresse mail est "admin" et le mot de passe aussi. L'administrateur peut choisir d'afficher

les recommandations personnalisées de n'importe quel utilisateur, et lorsqu'il choisit le droit à l'oubli il peut supprimer le compte utilisateur qu'il souhaite.

Bilan du projet

Le projet s'est bien passé car les tâches ont été réparties dès le début. Il y avait cependant un point qui aurait pu être amélioré : nous n'avons pas créer de Git pour le code Java, ainsi il était plus pénible de merge le code Java quand nous étions plusieurs dessus (ou bien même pour partager le code).

Les points difficiles ont été surtout de devoir changer (plusieurs fois en cours de route) l'analyse/conception du problème quand nous nous sommes rendu compte qu'il manquait certaines choses comme la colonne AchatReussi, ou bien qu'il aurait fallu faire les choses différemment. Les requêtes SQL ont donc dû être refaites ainsi que le diagramme en UML.

Cependant, ce projet a été très formateur car il nous a appris beaucoup de nouvelles choses. Nous avons développé notre autonomie car nous devions trouver des informations qui n'étaient pas forcément données. L'utilisation de bases de de données en java était nouveau pour nous mais cela s'est révélé très intéressant et très pratique !