Documentation projet BD

BOON Yi Jun

DIMASSI Mehdi

MOHD NOOR Nur Hakimi

NAIR Sritesh

TRAORÉ Fiè Victor

I) Analyse

Nous avons commencé d'abord par l'analyse du sujet afin d'identifier les données élémentaires, les dépendances fonctionnelles, les contraintes contextuelles et de multiplicité. On a modélisé l'analyse par le tableau ci-dessous:

DF	**Contraintes valeur**	**Contraintes de multiplicité**
utilisateur → produit	quantité ≥ 0	Un utilisateur peut avoir plusieurs produits
Catégorie,TypeVente → salleVente	prixOffre ≥ 0	Une salle de vente contient plusieurs ventes
idVente → produit,salleVente	prixRevient ≥ 0	Un produit a plusieurs caractéristiques
nomCatégorie → descriptionCatégorie	stockVente ≥ 0	Une vente a un seul produit
idProduit → nomProduit,prixRevient,stockVente	prixDépart > 0	Une vente est dans une seule salle
idProduit → nomCaractéristique		Une salle de vente a une seule catégorie
nomCaractéristique → valeurCaractéristique		Une offre a un seul utilisateur
idVente → prixDépart,offreMultiple, durée,révocable		Un utilisateur peut avoir plusieurs offres
utilisateur,dateOffre,heureOffre → prixOffre,quantité		
email → nom, prénom, adresse		

Table 1: Tableau analyse des dépendances fonctionnelles

Les contraintes contextuelles sont :

- Accepter l'offre si le prix n'est pas inférieur au prix d'enchère.
- Toutes les ventes dans une salle sont du même type.
- Si la vente est en offre unique, respecter la contrainte.
- La quantité demandé est inférieur au stock.

Nous avons ensuite identifiés les entités de la base de données et traduit les DF en schéma entité association suivant :

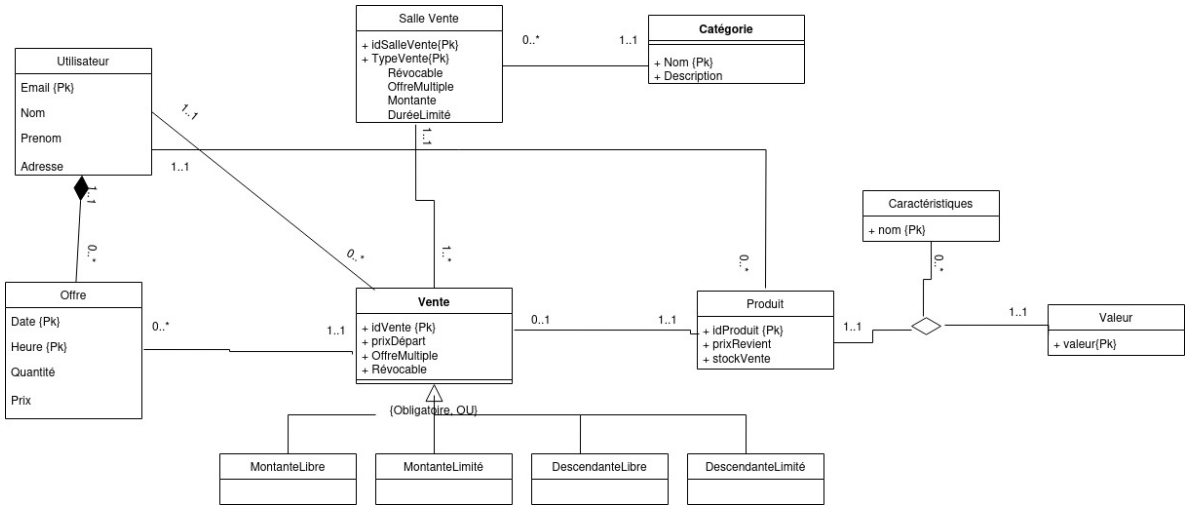

Figure 1: Schéma entité association

NB : On vérifie aussi que tout nos entités sont normalisés afin d'éviter les redondances ce qui est bien le cas.

II) Traduction en Relationnel et Conceptions Entités-Associations

Pour passer au relationnel, nous allons procéder comme expliqué dans le cours en créant d'abord:

1) Les tables pour les entités simples

- **Utilisateur** (Email, Nom, Prenom, Adresse)
- **SalleVente** (Id_salle, revocable, offreMultiple, Montante, dureeLimite)
- **Categorie** (nom_categorie, desc_cat)
- **Offre** (date, quantité, prix_offre)
  - `date` représente le couple date-heure dans le schéma
- **Vente** (Id_vente, prix_Départ, offreMultiple, revocable, encours)
- **Produit** (id_produit, nom_produit, prix_revient, stock_vente)
- **Caracteristiques** (nom_caracteristiques)

2) Gestion des Sous-Types d'entités

Dans notre cas, cela concerne les blocs `MontanteLibre`, `MontanteLimité`, `DescendanteLibre`, `DescendanteLimité` du diagramme UML.

Étant donné que ce sont des sous-types de **Vente**, nous avons combiné deux méthodes:

a) **Référence** : Création de deux tables `VenteLibre` et `VenteLimité` dont voici les schémas:

- `VenteLibre` (`id_vente`)
- `VenteLimite` (`id_vente`, `date_fin`) *(avec* ***`date_fin`*** *not NULL)*

Ainsi toutes nos ventes seront reparties dans ces deux tables selon qu'elles soient libre ou à durée limitée.

Ensuite, pour gérer le fait qu'une vente peut être descendante ou montante, nous avons opté pour le cas :

b) **Unification** : Ajout d'un attribut `prix_baisse` à la table `Vente`, qui est NULL si la vente est montante. Sinon, il indique le montant duquel le prix de la vente baisse périodiquement:

- `Vente` (`Id_vente`, `prix_Départ`, `offreMultiple`, `revocable`, `encours`, `prix_baisse`)

NB1: Nous avons utilisé ce même principe d'unification pour gérer le fait qu'une vente puisse être révocable ou multiple. Ainsi, les arguments entiers `revocable` et `offre_multiple` valent 0 ou 1 selon que la vente est non-révocable/non-multiple (0) ou révocable/multiple (1).

Avant de passer aux entités faibles et associations, la table `Vente` est ainsi:

- `Vente` (`Id_vente`, `prix_Départ`, `offreMultiple`, `revocable`, `encours`, `offre_multiple`, `revocable`, `prix_baisse`)

3) Types d'entités faibles

Dans notre conception, seule l'entité `Offre` est une entité faible, car elle ne peut pas être identifiée uniquement par sa clé primaire `date` (plusieurs utilisateurs peuvent passer une offre à la même heure). Ainsi, la table `Offre` devient:

- `Offre` (`date`, `email`, `quantité`, `prix_offre`)

4) Les associations

a) **Associations avec cardinalité 1..1** :

- Ajout de la clé primaire de la table avec la cardinalité 1..1 dans l'autre table. Ainsi:
  - **SalleVente** (`Id_salle`, `revocable`, `offreMultiple`, `Montante`, `dureeLimite`, `nom_categorie`)
  - **Vente** (`Id_vente`, `prix_Départ`, `offreMultiple`, `revocable`, `encours`, `prix_baisse`, `id_salle`, `Email`, `id_produit`)
  - **Offre** (`date`, `email`, `quantité`, `prix_offre`, `id_vente`)

b) **Associations avec cardinalité 0..1** : Gérée dans le cas précédent.

c) **Associations avec cardinalité ?..*** : Entre les tables `Produit` et `Caracteristiques`, un produit peut avoir plusieurs caractéristiques. Nous définissons une table d'association:

- **DescProduit** (`id_produit`, `nom_caractéristique`, `valeur`)

5) Tables finales

Voici la liste des tables finales:

- **Utilisateur** (Email, Nom, Prenom, Adresse)
- **SalleVente** (Id_salle, revocable, offreMultiple, Montante, dureeLimite, nom_categorie)
- **Categorie** (nom_categorie, desc_cat)
- **Offre** (date, email, quantité, prix_offre, id_vente)
- **Vente** (Id_vente, prix_Départ, offreMultiple, revocable, encours, prix_baisse, id_salle, Email, id_produit)
- **Produit** (id_produit, nom_produit, prix_revient, stock_vente, Email)
- **Caracteristiques** (nom_caracteristiques)
- **DescProduit** (id_produit, nom_caractéristique, valeur)

III) ANALYSE DES TRANSACTIONS (réalisations des fonctionnalités en SQL2)

Pour gérer certaines contraintes qu'il fallait vérifier dans les transactions, nous n'avons pas écrit de fichier complet uniquement en SQL qui gère des transactions. Nous gérons les fonctionnalités avec l'application JDBC.

Pour une transaction, nous la *committons* dès qu'elle a réussi, ce qui permet des transactions en concurrence sans mettre en péril notre application.

La transaction commence en désactivant l'*auto-commit* avec `this.con.setAutoCommit(false)`. Ainsi, toutes les opérations sont regroupées et aucun changement n'est appliqué à la base de données tant qu'elles ne réussissent pas entièrement. En cas de succès, les modifications sont validées avec `commit`; si une erreur survient à une étape, aucune des modifications précédentes ne sera appliquée.

Réalisation des fonctionnalités

1. Mise en place de salle et sélection de produit

1. **Vérification de l'existence de la salle de vente**
   Lorsque l'utilisateur souhaite mettre un produit en vente, le système commence par vérifier si une salle de vente existe déjà pour la catégorie correspondante. Cette vérification est effectuée en interrogeant la base de données via une transaction. Si aucune salle n'est trouvée :
   - Une nouvelle catégorie est créée (si elle n'existe pas déjà) via une opération d'insertion encapsulée dans la transaction.
   - Une nouvelle salle de vente est ensuite créée et insérée dans la base de données, toujours dans le cadre de la même transaction.

   En cas d'échec à cette étape, toutes les modifications sont annulées pour préserver la cohérence des données.

2. **Validation de la propriété du produit**
   Une fois la salle de vente vérifiée ou créée, le système s'assure que l'utilisateur est bien le propriétaire du produit avant de continuer. Si l'utilisateur n'est pas le propriétaire, l'opération est interrompue, et aucune modification n'est apportée à la base de données.

3. **Enregistrement de la vente**
   Si toutes les vérifications précédentes sont satisfaites, le système passe à l'enregistrement de la vente dans la base de données.
   - Si la vente est de type libre, un enregistrement est ajouté également à la table `VenteLibre`.

- Si la vente est limitée, un enregistrement est ajouté à la table `VenteLimite`, accompagné des dates de début et de fin.

4. **Validation ou annulation de la transaction**
   Si toutes les opérations réussissent, un `COMMIT` valide définitivement les modifications apportées à la base de données. En cas d'erreur (par exemple, si la salle ne peut pas être créée ou si les informations de la vente sont incorrectes), un `ROLLBACK` est automatiquement déclenché, annulant toutes les modifications effectuées jusqu'à ce point.

2. Offre utilisateur

Gestion des enchères montantes:

1. **Validation du prix proposé**
   Une validation est effectuée pour s'assurer que le prix proposé est supérieur :

   - Au prix de départ.
   - À l'enchère actuelle la plus élevée (si applicable).

   En cas de prix invalide, l'utilisateur doit ressaisir son offre.

2. **Vérification de l'unicité de l'offre**
   Si l'enchère n'accepte pas plusieurs offres par utilisateur (`offre_multiple = 0`), une vérification est effectuée pour s'assurer que l'utilisateur n'a pas déjà fait d'offre.

3. **Placement de l'offre**
   Si l'enchère est toujours active (non expirée), l'offre est placée en appelant la méthode `placeMontanteBid`. En cas d'échec (par exemple, si l'enchère est terminée), l'utilisateur en est informé, et aucune modification n'est effectuée.

Gestion des enchères descendantes:
Le prix du produit diminue toutes les minutes jusqu'à ce qu'il soit acheté par un utilisateur. Le produit peut être acheté à tout moment au prix actuel.

3. Processus de fin et désignation du vainqueur

Pour l'enchère *montante*, l'enchère est terminée à cause de la limite du temps (temps limite) ou lorsqu'une offre reste valable pendant 10 minutes (temps libre).
Pour l'enchère *descendante*, l'enchère est terminée à cause de la limite du temps ou lorsqu'un utilisateur achète le produit au prix proposé (temps libre).

1. **Récupération des ventes actives**
   La méthode `checkAndEndAuctions` est appelée pour vérifier toutes les enchères en cours (`encours = 1`) et gérer leur fin si elles sont terminées.

2. **Vérification de l'état de l'enchère**
   La méthode `isAuctionEnded(venteId)` est utilisée pour vérifier si l'enchère associée à `venteId` est terminée.Si l'enchère est terminée, la méthode handleAuctionEnd(venteId) est appelée pour gérer les étapes nécessaires à sa clôture.

3. **Gestion de la fin de l'enchère**
   Dans la méthode `handleAuctionEnd(venteId)` :

   - Une requête SQL sélectionne la meilleure offre pour cette vente: (prix le plus élevé) et récupère les informations sur le gagnant (email), le prix offert (prixOffre), et la quantité achetée (quantite)
   - Le stock du produit correspondant est mis à jour: Le stock du produit correspondant à cette vente est diminué de la quantité achetée. Cette mise à jour est effectuée par une requête SQL qui ajuste la valeur de stockVente dans la table Produit.

- La vente est marquée comme terminée: Le statut de la vente est mis à jour dans la table Vente (encours passe à 0), indiquant que l'enchère est clôturée.

4. **Validation de la transaction**
   Après avoir traité toutes les ventes actives, `this.con.commit()` est appelé pour confirmer les changements. En cas d'erreur, un `ROLLBACK` est déclenché.

Si ces étapes réussissent, la méthode retourne `true`, indiquant que l'enchère a été gérée avec succès.

4. Validation de la transaction

Après avoir traité toutes les ventes actives, `this.con.commit()` est appelé pour confirmer les changements dans la base de données. Si une erreur se produit pendant le processus, une exception est levée, et les changements ne sont pas validés, garantissant ainsi l'intégrité des données.

IV. Bilan du projet

Organisation

Nous avons travaillé ensemble sur les premières séances. Après plusieurs relectures du sujet, nous avons entamé l'analyse du problème en identifiant les dépendances fonctionnelles ainsi que les contraintes de valeur et de multiplicité.

Dès la troisième séance, nous avions terminé la réalisation du diagramme UML et commencé à réfléchir au passage au SQL.

Par la suite :

- Un étudiant s'occupait de comprendre `JDBC`,
- Les autres implémentaient la base de données en SQL.

Dans les dernières séances :

- Trois étudiants ont travaillé sur l'application de démonstration et les transactions (`SQL` + `Java`) pour réaliser les fonctionnalités.
- Les deux autres ont débuté la rédaction du rapport et du diaporama de présentation.

Difficultés rencontrées

Nous n'avons pas eu une idée fini directement,

On échangeaient régulièrement pour se mettre d'accord sur les fonctionnalités demandées par le sujet et comment les réalisés techniquement en java.

Pour anecdote, nous avons passée toute une séances, parmi les dernières, à être d'accord sur comment on devait relier une salle de vente à une vente, (Ehh ouais ... a t-on 16 ou plus types de salles toutes prêtes et on y mets les ventes correspondantes au fur et à mesure? Chaque vente engendre t-il la création d'une nouvelle salle? , regroupe t-on les ventes par Salle de catégorie sans tenir compte du type de vente?...)

Au final, Nous avons décider de créer les salles au besoin.

Autre exemple: dans notre première version du code SQL, nous avions mis dans les tables où il fallait un attribut temps deux paramètres date et heure. Ensuite, Il était difficile de gérer cela en SQL, donc nous avons juste pris un attribut date de type TIMESTAMP.

Également, nous avons beaucoup discuté sur l'implémentation concrète des sous classes possibles pour une vente (révocable?, offreMultiple?, libre?, montante?). Il y a 2^4 choix à gérer et nous avons mélanger des solutions implémentation comme cela nous semblaient mieux (voir section passage au relationnel).

Bilan final

Nous sommes fiers de notre travail et pensons avoir respecté le cahier des charges du projet. Le manque de détails techniques dans l'énoncé nous a mis au défi, mais cela a également stimulé notre capacité à collaborer efficacement.

Mode d'emploi

NB : Nous avons implémenté le projet sur le compte Oracle de l'étudiant avec le login `nairs`.

1. Après vous être connecté à Oracle avec ce login, placez-vous dans le dossier racine du projet :
   - Lancez `start main.sql` pour créer les tables.
   - (Facultatif) Peupler la base de données avec `start data.sql`.
2. Dans un autre terminal :
   - Accédez au dossier `app`.
   - Compilez l'application avec `javac TerminalApp.java`.
   - Exécutez `java TerminalApp`.

L'application propose des menus intuitifs pour l'utilisation des fonctionnalités. Voici comment vérifier leur bon fonctionnement :

a) Fonctionnalité 1 : Mise en place de salle de vente et choix de produits disponibles pour créer une vente

- Si vous avez lancé le fichier `data.sql`, des produits existent déjà pour l'utilisateur `Alice` :
  1. Choisissez l'option `1)` au lancement de l'application.
  2. Entrez l'email : `alice@example.com`.
  3. Choisissez ensuite l'option `1)` pour créer une vente. La liste de vos produits disponibles s'affiche, il faut alors choisir l'ID du produit à mettre en vente.
  4. Par exemple, choisissez l'ID `1` pour mettre les `Laptop` en vente.
  5. Répondez aux questions concernant les détails de votre vente :
     - Prix de départ : 700
     - Montante ou descendante : `1 (Montante)`
     - Offres multiples ou non : `1 (Oui)`
     - Vente révocable ou non : `1 (Oui)`
     - Vente libre ou limitée : `1 (Libre)`
     - Entrez une catégorie pour votre produit (Si elle n'existe pas, elle sera créée).
     - Description de la catégorie (facultatif) : tapez `Entrée` sans rien.
- Votre vente est automatiquement assignée à une salle de vente selon les consignes. Vous revenez ensuite à la page d'accueil.
- Vérifiez que votre vente a été créée :
  1. Choisissez l'option `5)`.
  2. Une liste s'affiche avec les caractéristiques de vos ventes, incluant l'ID de la salle.
- Note : Comme la vente est révocable, il est proposé de la retirer si nécessaire.
- Si vous n'avez pas lancé `data.sql`, procédez ainsi :
  1. Choisissez l'option `2)` pour créer un compte.
  2. Connectez-vous avec votre login.
  3. Ajoutez un produit (la procédure est intuitive).
  4. Référez-vous ensuite à la première section pour mettre le produit en vente.

b) Fonctionnalité 2 : Enchère par un utilisateur (`Offre`)

Deux cas de figure :

- Si vous avez lancé `data.sql`((finalement il faut le lancer ce fichier pour avoir la vie facile. Nous l'avons pas fait pour vous laisser la liberté !!!) :
  1. Connectez-vous avec l'utilisateur `Bob` (`bob@example.com`).
  2. Choisissez l'option `2)` pour faire une offre.
  3. Une catégorie vous est demandée, choisissez par exemple `1)` pour voir les offres.
  4. Une liste de ventes avec des ID s'affiche. Sélectionnez l'ID du produit convoité (par exemple, `20` pour l'offre d'Alice).
  5. Indiquez la quantité désirée en fonction du stock disponible (par exemple, `4` pour l'offre d'Alice).
  6. Entrez un prix supérieur au prix de départ (par exemple, `700`).
- Vous revenez ensuite à la page d'accueil. Choisissez l'option `4)` pour voir vos offres avec un indicateur vous précisant si vous êtes gagnant potentiel.

Note : Au bout d'une minute (au lieu de dix comme indiqué dans l'énoncé), les enchères prennent fin. Si vous êtes vendeur, vous pouvez voir vos ventes terminées en choisissant l'option `7)` dans le menu d'accueil.

c) Fonctionnalité 3 : Désignation du vainqueur

- **Si vous êtes acheteur :** Choisissez l'option `4)` pour savoir si vous avez gagné.
- **Si vous êtes vendeur :** Choisissez l'option `7)` pour voir vos ventes terminées et le montant gagné.