



# Documentation du projet BD

Omar OUKHTITE, Mohammed Ziyad CHLIHI, Yassine OUAGGASS, Adil KASSAOUI

## Contents

<b>1</b>	<b>Analyse des contraintes</b>	<b>3</b>
1.1	Propriétés . . . . .	3
1.2	Contraintes de dépendances fonctionnelles . . . . .	3
1.3	Contraintes de valeurs . . . . .	3
1.4	Contraintes de multiplicité . . . . .	4
1.5	Contraintes contextuelles . . . . .	4
<b>2</b>	<b>Conception Entités/Associations</b>	<b>5</b>
<b>3</b>	<b>Traduction en relationnel</b>	<b>5</b>
3.1	Relations obtenues . . . . .	5
3.2	Formes normales des relations . . . . .	6
3.3	Contraintes non implantables directement dans le relationnel . . . . .	6
<b>4</b>	<b>Analyse des fonctionnalités</b>	<b>6</b>
4.1	1ère fonctionnalité . . . . .	6
4.2	2ème fonctionnalité: ajout d'une offre sur un produit mis en vente . . . . .	7
4.3	3ème fonctionnalité : processus du fin d'enchère . . . . .	8
<b>5</b>	<b>Gestion des Transactions et du Temps dans le Code</b>	<b>9</b>
5.1	Utilisation de l'option autocommit . . . . .	9
5.2	Gestion du Temps dans le Code . . . . .	9
<b>6</b>	<b>Bilan</b>	<b>10</b>
6.1	Organisation du Travail . . . . .	10
6.2	Difficultés Rencontrées . . . . .	10
<b>7</b>	<b>Contenu du Répertoire</b>	<b>10</b>
<b>8</b>	<b>Instructions d'Utilisation</b>	<b>11</b>

# 1 Analyse des contraintes

## 1.1 Propriétés

{ NumeroProduit, NomProduit, PrixRevient, Stock,  
NomCategorie, Description,  
NumeroProduit, NomCaracteristique, Valeur,  
NumeroSalle,  
NumeroVente, PrixDepart, Revocable, Montante, PlusieursOffresParUtilisateur, DateDebut, DateFin,  
NumeroOffre, PrixAchat, DateOffre, Quantite, Email, Nom, Prenom, Adresse }

## 1.2 Contraintes de dépendances fonctionnelles

- **Produit** : un produit est indentifié de façon unique par son numéro.

$$\text{NumeroProduit} \rightarrow \{\text{NomProduit}, \text{PrixRevient}, \text{Stock}, \text{NomCategorie}\}$$

- **Catégorie** : une catégorie est identifiée de façon unique par son nom.

$$\text{NomCategorie} \rightarrow \{\text{Description}\}$$

- **Caractéristique** : Une caractéristique est identifiée de manière unique par le couple (NumeroProduit, NomCaracteristique).

$$\{\text{NumeroProduit}, \text{NomCaracteristique}\} \rightarrow \{\text{Valeur}\}$$

- **Salle** : Une salle est identifiée de façon unique par son numéro.

$$\text{NumeroSalle} \rightarrow \text{NomCategorie}$$

- **Vente** : Une vente est identifiée de façon unique par son numéro.

$$\text{NumeroVente} \rightarrow \{\text{PrixDepart}, \text{Revocable}, \text{Montante}, \text{PlusieursOffresParUtilisateur}, \text{DateDebut}, \\ \text{DateFin}, \text{NumeroProduit}, \text{NumeroSalle}\}$$

- **Offre** :

$$\text{NumeroOffre} \rightarrow \{\text{PrixAchat}, \text{DateOffre}, \text{Quantite}, \text{Email}, \text{NumeroVente}\}$$

- **Utilisateur** :

$$\text{Email} \rightarrow \{\text{Nom}, \text{Prenom}, \text{Adresse}\}$$

## 1.3 Contraintes de valeurs

- **Stock (Produit)** :

$$\text{Stock} \geq 0$$

- **Quantité (Offre)** :

$$\text{Quantite} > 0$$

- **Prix (Produit, Vente, Offre)** :

$$\text{PrixRevient}, \text{PrixDepart}, \text{PrixAchat} > 0$$

## 1.4 Contraintes de multiplicité

- **Produit  $\leftrightarrow$  Catégorie :**

$$\text{Produit} \xrightarrow{1,\dots,1} \text{Catégorie}$$

Chaque produit appartient obligatoirement à une et une seule catégorie, mais une catégorie peut contenir plusieurs produits.

- **Produit  $\leftrightarrow$  Caractéristique :**

$$\text{Produit} \xrightarrow{1,\dots,1} \text{Caractéristique}$$

Chaque produit peut avoir plusieurs caractéristiques.

- **Vente  $\leftrightarrow$  Produit :**

$$\text{Vente} \xrightarrow{1,\dots,1} \text{Produit}$$

Une vente porte sur un et un seul produit.

- **Vente  $\leftrightarrow$  Salle :**

$$\text{Vente} \xrightarrow{1,\dots,1} \text{Salle}$$

Chaque vente a lieu dans une salle unique.

- **Utilisateur  $\leftrightarrow$  Offre :**

$$\text{Utilisateur} \xrightarrow{0,\dots,*} \text{Offre}$$

Un utilisateur peut proposer une ou plusieurs offres (ou aucune), mais chaque offre est liée à un seul utilisateur.

- **Vente  $\leftrightarrow$  Offre :**

$$\text{Vente} \xrightarrow{0,\dots,*} \text{Offre}$$

Une vente peut recevoir une ou plusieurs offres (ou aucune offre), mais chaque offre est associée à une seule vente.

- **Vente  $\leftrightarrow$  VenteDuréeLimitée/VenteDuréeIllimitée :** Une vente doit être liée soit à **VenteDuréeLimitée**, soit à **VenteDuréeIllimitée**, mais pas aux deux.

## 1.5 Contraintes contextuelles

- **Offre uniquement pendant la vente (si limitée) :**

$$\text{DateOffre} \leq \text{DateFin}$$

- **Prix de départ et prix d'achat :**

$$\text{PrixAchat} \geq \text{PrixDepart}$$

Chaque offre doit proposer un prix d'achat supérieur au prix de départ de la vente.

- **Quantité totale des offres d'un même utilisateur :**

$$\sum_{\text{offres d'un utilisateur dans une vente}} \text{Quantite(offre)} \leq \text{Stock}$$

La somme des quantités des offres soumises par un même utilisateur sur une vente doit être inférieure ou égale au stock du produit correspondant.

- **Vente limitée dans le temps :**

$$\text{DateFin} > \text{DateDebut}$$

## 2 Conception Entités/Associations

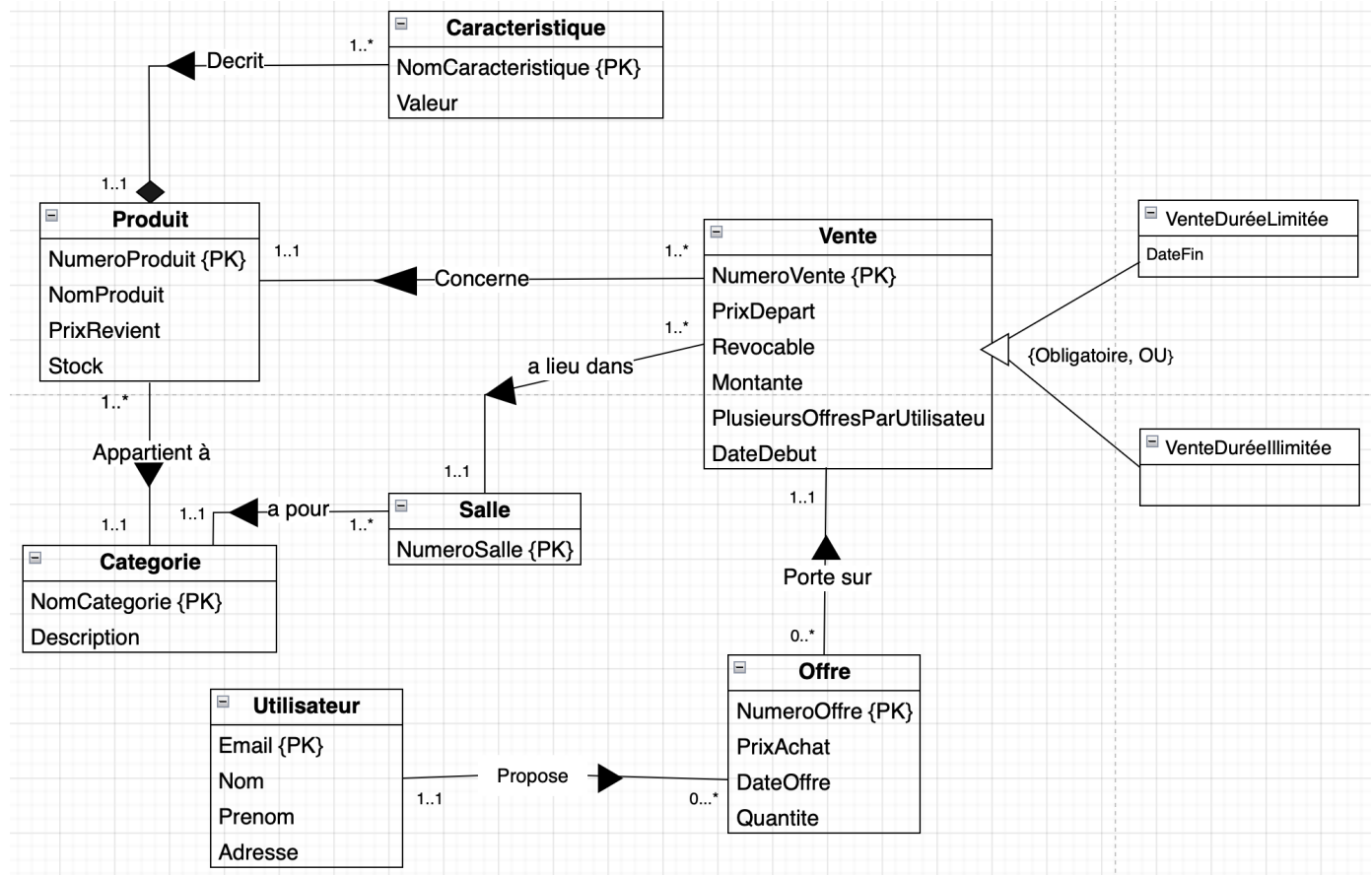


Figure 1: Diagramme UML des entités et associations

- La relation **Caractéristique** est considérée comme une entité faible par rapport à **Produit**, car **NomCaractéristique** ne suffit pas, à lui seul, pour identifier la valeur associée. L'identification nécessite également une référence au produit correspondant.
- Les prix (**PrixRevient**, **PrixAchat**, **PrixDepart**) ne représentent pas un prix unitaire, mais plutôt le coût total correspondant à l'achat de l'ensemble de la quantité concernée.

Après élaboration du schéma Entités/Associations, on remarque bien que ce schéma est cohérent avec l'analyse faite, et respecte bien les contraintes.

## 3 Traduction en relationnel

### 3.1 Relations obtenues

- **Catégorie** (NomCategorie, Description)
- **Produit** (NumeroProduit, NomProduit, PrixRevient, Stock, NomCategorie)  
NomCategorie référence Catégorie.
- **Caractéristique** (NumeroProduit, NomCaracteristique, Valeur )  
NumeroProduit référence Produit.

- **Salle** ( NumeroSalle, NomCategorie )  
NomCategorie référence Catégorie.
- **Vente** ( NumeroVente, PrixDepart, Revocable, Montante, PlusieursOffresParUtilisateur, DateDebut, NumeroProduit, NumeroSalle )  
NumeroProduit référence Produit, NumeroSalle référence Salle.
- **VenteDuréeLimitée** ( NumeroVente, DateFin )  
NumeroVente référence Vente.
- **VenteDuréeIllimitée** ( NumeroVente )  
NumeroVente référence Vente.
- **Utilisateur** ( Email, Nom, Prenom, Adresse )
- **Offre** ( NumeroOffre, PrixAchat, Date, Quantite, Email, NumeroVente )  
Email référence utilisateur, NumeroVente référence Vente.  
NumeroVente référence Vente.

### 3.2 Formes normales des relations

Les relations obtenues respectent les conditions de la **Troisième Forme Normale Boyce-Codd-Kent (3FNBCK)**:

- **1FN** : Tout attribut ne contient que des valeurs atomiques et non nulles.
- **2FN** : Tous les attributs non-clés sont pleinement dépendants de chacune des clés.
- **3FN** : Tout attribut non-clé ne dépend pas d'un ensemble d'attributs qui n'est pas une clé.
- **3FNBCK** : Pour toute dépendance fonctionnelle  $X \rightarrow Y$  non triviale,  $X$  contient une clé de la relation.

### 3.3 Contraintes non implantables directement dans le relationnel

- **Contraintes contextuelles sur les dates et heures :**

$$\text{DateFin} > \text{DateDebut}$$

- **Prix et quantités :**

$$\text{PrixAchat} \geq \text{PrixDepart}, \quad \text{Quantite} > 0$$

- **Somme des quantités des offres d'un utilisateur :**

$$\sum_{\text{offres d'un utilisateur dans une vente}} \text{Quantite(offre)} \leq \text{Stock}$$

## 4 Analyse des fonctionnalités

### 4.1 1ère fonctionnalité

Cette fonctionnalité permet de créer une salle de vente et d'ajouter des produits à cette salle pour une vente. Le processus est divisé en plusieurs étapes :

- **Choix de la catégorie** : L'utilisateur sélectionne une catégorie de produits parmi celles disponibles. Si aucune catégorie ne contient de produits, le processus d'ajout de produits est interrompu.

- **Création de la salle de vente** : Une fois la catégorie choisie, une nouvelle salle de vente est créée avec un numéro unique et la catégorie sélectionnée.
- **Ajout de produits à la salle de vente** : L'utilisateur peut sélectionner des produits disponibles dans la catégorie choisie pour les ajouter à la salle de vente. Des informations comme le prix de départ, la possibilité de révoquer l'enchère, et d'autres conditions sont saisies.
- **Durée de la vente** : L'utilisateur peut également définir si la vente a une durée limitée ou illimitée. Si la vente est limitée, une date de fin est spécifiée.

```

1  INSERT INTO Salle (NumeroSalle, NomCategorie) VALUES (?, ?);
2  INSERT INTO Vente (NumeroVente, NumeroSalle, NumeroProduit,
3  PrixDepart, Revocable, Montante, PLUSIEURSOFFRESPARUTILISATEUR, DateDebut)
4  VALUES (?, ?, ?, ?, ?, ?, ?, ?, CURRENT_TIMESTAMP);
5
6  -- Si vente de duree limitee
7  INSERT INTO VENTEDUREELIMITEE (NumeroVente, DateFin) VALUES (?, ?);
8
9  -- Si vente de duree illimitee
10 INSERT INTO VENTEDUREEILLIMITEE (NumeroVente) VALUES (?);

```

## 4.2 2ème fonctionnalité: ajout d'une offre sur un produit mis en vente

Le code implémente la logique de création d'une offre dans un système de vente aux enchères. Il suit les étapes suivantes :

1. **Vérification de la possibilité de faire plusieurs offres par utilisateur** : La méthode `plusieursOffres` vérifie si l'utilisateur peut faire plusieurs offres sur une même vente, en consultant la colonne `PLUSIEURSOFFRESPARUTILISATEUR` dans la table `Vente`.
2. **Vérification du statut de la vente** : La méthode `statutVente` détermine si la vente est toujours en cours ou si elle est terminée, selon la présence d'une date de fin dans la table `VenteDureeLimitee` ou du temps écoulé depuis la dernière offre dans la table `Offre`.
3. **Récupération des salles et des produits disponibles** : L'utilisateur choisit une salle et une vente. Le code vérifie si la vente est valide et si l'utilisateur peut participer.
4. **Vérification des offres existantes** : Si l'utilisateur a déjà fait une offre pour cette vente, il ne pourra pas en faire une nouvelle, sauf si la vente permet plusieurs offres par utilisateur.
5. **Création de l'offre** : Si l'utilisateur choisit de faire une offre, le code vérifie si l'offre est valide (prix d'achat supérieur à la meilleure offre existante et quantité dans les limites du stock). L'offre est ensuite insérée dans la base de données.

### Requêtes SQL Exécutées

```

1  -- Verification des offres existantes
2  SELECT * FROM Offre
3  WHERE EMAIL = ? AND NUMEROVENTE = ?
4
5  -- Recupration de la meilleure offre pour une vente donnee
6  SELECT * FROM utilisateur u
7  JOIN offre o ON u.email = o.email
8  WHERE o.numerovente = ?
9  AND u.email IN (
10     SELECT o1.email FROM offre o1
11     WHERE o1.numerovente = ?
12     GROUP BY o1.email
13     HAVING SUM(o1.prixachat) = (
14         SELECT MAX(total) FROM (
15             SELECT SUM(o2.prixachat) AS total

```

```

12         FROM offre o2
13         WHERE o2.numerovente = ?
14         GROUP BY o2.email
15     ) max_totals
16 )
17 )

```

```

1  -- Insertion d'une nouvelle offre
2  INSERT INTO Offre (NUMEROOFFRE, PRIXACHAT, DATEOFFRE, QUANTITE, EMAIL, NUMEROVENTE)
3  VALUES (?, ?, CURRENT_TIMESTAMP, ?, ?, ?)

```

### 4.3 3ème fonctionnalité : processus du fin d'enchère

Le processus se déroule en plusieurs étapes :

1. **Vérification de la vente** : La vente est vérifiée pour déterminer si elle est en cours ou terminée. Cela dépend de plusieurs critères comme la date de fin de la vente, la présence de plusieurs offres par utilisateur, et la durée de la vente.
2. **Vérification des offres** : Le système vérifie si un utilisateur a déjà fait une offre sur la vente en question. Si plusieurs offres par utilisateur sont autorisées, l'utilisateur peut soumettre une nouvelle offre.
3. **Sélection du gagnant** : Le gagnant est déterminé par la somme totale des prix des offres d'un utilisateur sur une vente. Si plusieurs utilisateurs ont soumis des offres, celui avec la somme la plus élevée est déclaré gagnant.
4. **Affichage du meilleur prix** : Le prix total des offres est calculé pour chaque utilisateur et comparé pour déterminer le gagnant de l'enchère.

#### Requêtes SQL Exécutées

- Vérification de la possibilité de plusieurs offres La première requête détermine si la vente permet plusieurs offres par utilisateur.

```

1  SELECT PLUSIEURSOFFRESPARUTILISATEUR
2  FROM Vente
3  WHERE NUMEROVENTE = ?

```

Cette requête vérifie si la vente (NUMEROVENTE) permet plusieurs offres par utilisateur. Si la colonne PLUSIEURSOFFRESPARUTILISATEUR contient la valeur 'Y', cela signifie que plusieurs offres sont autorisées.

- Vérification de l'état de la vente (En cours ou terminée) La requête suivante est utilisée pour vérifier si la vente est toujours en cours ou si elle est terminée. Cela dépend de la présence d'une date de fin spécifique.

```

1  SELECT DATEFIN
2  FROM VenteDureeLimitee
3  WHERE NUMEROVENTE = ?

```

Si la vente a une durée limitée, cette requête récupère la DATEFIN pour vérifier si la vente est toujours en cours.

- Vérification de la dernière offre faite sur la vente La dernière offre pour une vente spécifique est récupérée avec la requête suivante :



```

1 SELECT *
2 FROM Offre
3 WHERE NUMEROVENTE = ?
4 AND DATEOFFRE = (
5     SELECT MAX(DATEOFFRE)
6     FROM Offre
7     WHERE NUMEROVENTE = ?
8 )

```

Cette requête récupère la dernière offre faite sur une vente donnée, en cherchant l'offre avec la date la plus récente.

- Sélection du gagnant de l'enchère La requête suivante permet de déterminer l'utilisateur ayant fait la meilleure offre, en fonction du montant total des prix d'achat.

```

1 SELECT *
2 FROM utilisateur u
3 JOIN offre o ON u.email = o.email
4 WHERE o.numerovente = ?
5 AND u.email IN (
6     SELECT o1.email
7     FROM offre o1
8     WHERE o1.numerovente = ?
9     GROUP BY o1.email
10    HAVING SUM(o1.prixachat) = (
11        SELECT MAX(total)
12        FROM (
13            SELECT SUM(o2.prixachat) AS total
14            FROM offre o2
15            WHERE o2.numerovente = ?
16            GROUP BY o2.email
17        ) max_totals
18    )
19 )

```

Cette requête permet de déterminer l'utilisateur dont la somme des prix d'achat (**prixachat**) est la plus élevée. Si plusieurs utilisateurs ont la même somme maximale, ils sont tous considérés comme gagnants.

- 5. Insertion d'une nouvelle offre Si un utilisateur souhaite faire une nouvelle offre, celle-ci est insérée dans la base de données avec la requête suivante :

```

1 INSERT INTO Offre (NUMEROOFFRE, PRIXACHAT, DATEOFFRE, QUANTITE, EMAIL, NUMEROVENTE)
2 VALUES (?, ?, CURRENT_TIMESTAMP, ?, ?, ?)

```

Cette requête insère une nouvelle offre dans la table **Offre**, avec les informations suivantes : **NUMEROOFFRE**, **PRIXACHAT**, **DATEOFFRE**, **QUANTITE**, **EMAIL** de l'utilisateur, et le numéro de la vente (**NUMEROVENTE**).

## 5 Gestion des Transactions et du Temps dans le Code

### 5.1 Utilisation de l'option autocommit

Dans notre projet, nous avons fait usage de l'option **autocommit** pour gérer les transactions dans la base de données, permettant ainsi d'effectuer une validation automatique (*commit*) après chaque instruction SQL.

### 5.2 Gestion du Temps dans le Code

La gestion du temps dans ce code joue un rôle central pour déterminer le statut des ventes.

- **Ventes limitées dans le temps** : La différence entre la date de fin (**DATEFIN**) et l'heure actuelle (**System.currentTimeMillis()**) est calculée afin de vérifier si la vente est encore active ou terminée.

- **Ventes descendantes** : Le prix initial est ajusté dynamiquement en fonction du temps écoulé depuis la date de début (`DATEDEBUT`), garantissant une diminution progressive des prix.
- **Ventes montantes** : La méthode `dateDerniereOffre` extrait la date de la dernière offre et compare le temps écoulé à un seuil de 10 minutes pour décider si la vente doit se clôturer.

Cette logique garantit une gestion précise et réactive des contraintes temporelles propres à chaque type de vente, tout en offrant des retours clairs sur les délais restants ou les raisons de la fin des ventes.

## 6 Bilan

Le projet a consisté à créer une base de données avec JDBC et à implémenter des requêtes SQL. L'organisation du travail a été structurée en plusieurs étapes, avec une collaboration fluide au sein de l'équipe.

### 6.1 Organisation du Travail

Le travail a été réparti de la manière suivante :

- L'équipe a d'abord travaillé collectivement sur l'analyse et le schéma entité-association.
- Une partie de l'équipe s'est concentrée sur le code Java et la gestion JDBC.
- Une autre partie a pris en charge la création de la base de données SQL et l'écriture des requêtes nécessaires.
- Par la suite, l'équipe a réajusté ses tâches : certains ont rejoint la partie JDBC et d'autres ont préparé la documentation et les slides.

GitHub a été utilisé pour gérer le code et suivre les évolutions du projet.

### 6.2 Difficultés Rencontrées

Une des difficultés majeures du projet a été la gestion du temps dans le code, car le temps est un paramètre clé dans les actions effectuées dans la base de données. L'intégration de la gestion temporelle avec les actions SQL a nécessité une réflexion approfondie, notamment pour faire des comparaisons entre les différentes dates et la date actuelle au moment de l'exécution du programme.

## 7 Contenu du Répertoire

Le projet est structuré de manière organisée pour faciliter son développement, son exécution et sa maintenance. Voici une présentation des fichiers et répertoires essentiels :

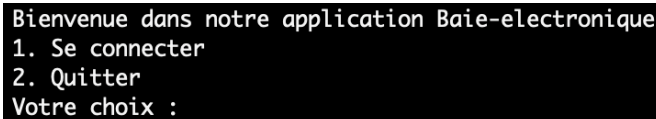
- **Makefile** : Ce fichier permet de simplifier la compilation et l'exécution du projet via des commandes prédéfinies.
- **src/** : Contient les fichiers source Java du projet. Les fichiers importants incluent :
  - `Main.java` : Point d'entrée du programme.
  - `DatabaseConnection.java` : Implémente la connexion avec la base de données.
  - `Acheter.java`, `CreerSalle.java`, `ConsulterResultats.java` : Implémentent respectivement les fonctionnalités d'achat, de création de salle, et de consultation des résultats.
- **bin/** : Ce répertoire contient les fichiers `.class` générés après la compilation.
- **E/A** : Regroupe les documents liés au diagramme Entités/Associations, notamment :
  - `E/A.png` : Une capture du diagramme entités-associations.

- `SchémaEntitésAssociations.drawio` : Le fichier source pour le diagramme entité-association.
- **lib/** : Contient la bibliothèque `ojdbc6.jar`, nécessaire pour l'interaction avec la base de données Oracle.
- **sql/** : Ce répertoire regroupe les scripts SQL utilisés pour la configuration et la gestion de la base de données :
  - `creertables.sql` : Crée les tables nécessaires.
  - `ajoutdedonnees.sql` : Ajoute les données initiales aux tables.
  - `reinitialiser.sql` : Réinitialise les données pour un nouveau départ (supprime les tables, les crée une autre fois et les peuple).

## 8 Instructions d'Utilisation

Voici les étapes pour accéder à la base de données, accompagnées d'exemples en ligne de commande :

1. **Naviguez dans le répertoire du projet** : Assurez-vous de vous placer dans le bon répertoire avant de commencer.
2. **Lancez la commande make** : Cela compile les fichiers nécessaires.
3. **Exécutez `make exeMain`** : Cela démarre le programme principal.

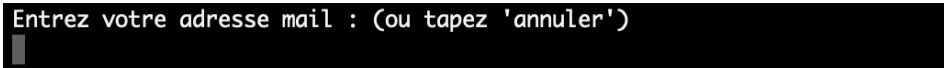


```

Bienvenue dans notre application Baie-electronique
1. Se connecter
2. Quitter
Votre choix :
  
```

Figure 2: Lancement de l'application

4. **Entrez votre email** : (ou quittez si vous voulez) Une fois le programme lancé, il vous sera demandé d'entrer votre email.



```

Entrez votre adresse mail : (ou tapez 'annuler')
  
```

Figure 3: Renseignement de mail

Voilà les adresses emails à utiliser pour pouvoir se connecter : `omar.oukhtite@example.com`, `ziyad.chlihi@example.com`, `yassine.ouaggass@example.com`, `adil.kassaoui@example.com`, `arsene.lupin@example.com` (oui, même Arsène Lupin s'intéresse à nos ventes, mais il a sûrement un plan derrière la tête !).

5. **Accédez aux données dans la base de données** : Après validation, vous accéderez au menu de l'application.

```
Bienvenue John Doe

Que souhaitez vous faire ?
1. Creer salle de vente
2. Acheter
3. Consulter les resultats
4. Quitter
Votre choix :
```

Figure 4: Menu de l'application

Et voilà ! Une fois dans le menu de l'application, vous pouvez :

- 1 : créer une salle de vente et y ajouter des ventes, selon leur catégorie ;
- 2 : enchérir sur un produit déjà disponible dans une salle de vente ;
- 3 : consulter les résultats des ventes auxquelles vous avez participé ;
- 4 : quitter l'application (mais pourquoi partir si tôt? On a encore plein de bonnes affaires!).