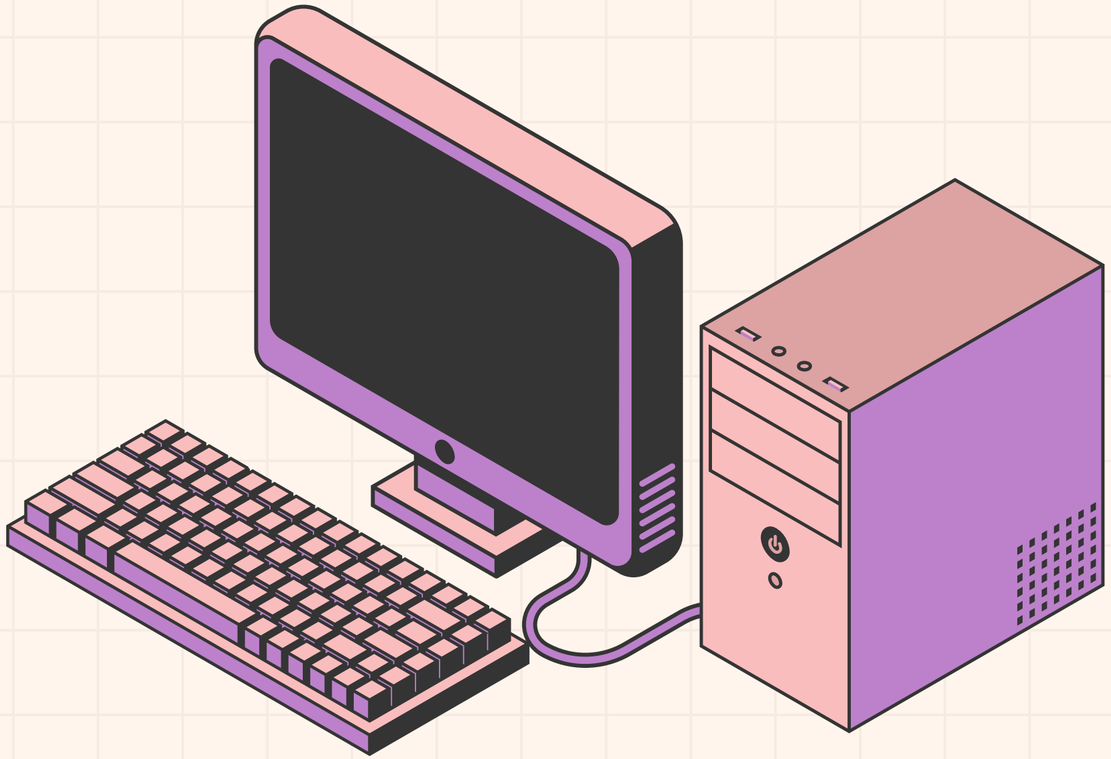
BAIE-ÉLECTRONIQUE

PROJET BASE DE DONNÉES



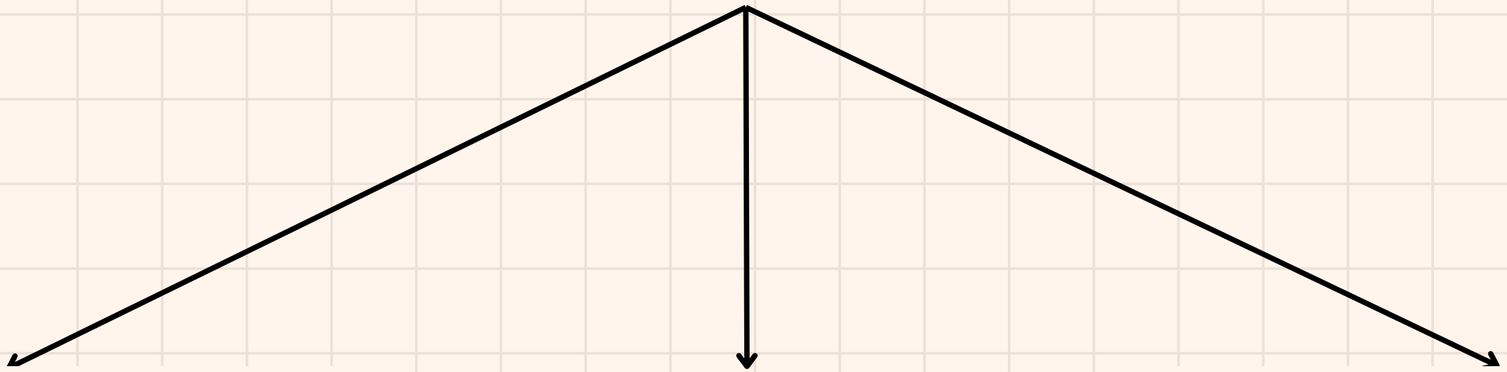
BAZOUANE SARA, BELALLAM SAAD, BENRAHMOUNE MOHAMMED AMINE,
IDBRAYME OMAR , KACIMI SALIM, LE MEUR ALEXIS

INTRODUCTION

Dépendances fonctionnelles

- NOM_CATEGORIE → DESCRIPTION
- ID_SALLE_VENTE → NOM_CATEGORIE
- ID_PRODUIT → PRIX_REVIENT, NOM_CATEGORIE, STOCK
- (ID_PRODUIT, NOM_CARACTÉRISTIQUE) → VALEUR
- MAIL → NOM, PRENOM, ADRESSE
- ID_VENTE → MONTANTE, LIMITE_OFFRE, REVOCABLE, PRIX_DEPART, DATE_FIN, PRIX_DESCENTE, STOCK_LOT, ID_SALLE_VENTE, ID_PRODUIT, MAIL
- ID_OFFRE → MAIL, ID_VENTE, DATE_OFFRE, PRIX_ACHAT, QUANTITE

Contraintes identifiées



CONTRAINTES DE VALEUR

— UNE CATÉGORIE EST DÉFINIE PAR UN NOM UNIQUE ET UNE DESCRIPTION.

— LES PRIX DES PRODUITS DOIVENT ÊTRE SUPÉRIEURS OU ÉGAUX À 0.

— LE STOCK D'UN PRODUIT EST UNE VALEUR ENTIÈRE POSITIVE.

— LA QUANTITÉ D'UNE OFFRE DOIT ÊTRE COMPRISE ENTRE 1 ET LE STOCK DISPONIBLE POUR LE PRODUIT CORRESPONDANT.

CONTRAINTES DE MULTIPLICITÉS

— UN PRODUIT APPARTIENT À UNE SEULE CATÉGORIE.

— UNE CATÉGORIE PEUT CONTENIR PLUSIEURS PRODUITS.

— UNE SALLE DE VENTE CONTIENT PLUSIEURS VENTES (AUMOINS UNE).

— UNE VENTE EST ASSOCIÉE À UN SEUL PRODUIT ET À UNE SEULE SALLE DE VENTE.

— UNE VENTE PEUT ÊTRE ASSOCIÉE À PLUSIEURS OFFRES.

— UNE OFFRE PEUT CONCERNER UN SOUS-LOT SPÉCIFIQUE OU UN PRODUIT COMPLET.

CONTRAINTES CONTEXTUELLES

— UNE OFFRE EST INVALIDE SI ELLE NE RESPECTE PAS LE SENS DE LA VENTE (MONTANTE OU DESCENDANTE).

— UNE VENTE RÉVOCABLE PEUT ÊTRE ANNULÉE SI LE PRIX DE REVIENT N'EST PAS ATTEINT.

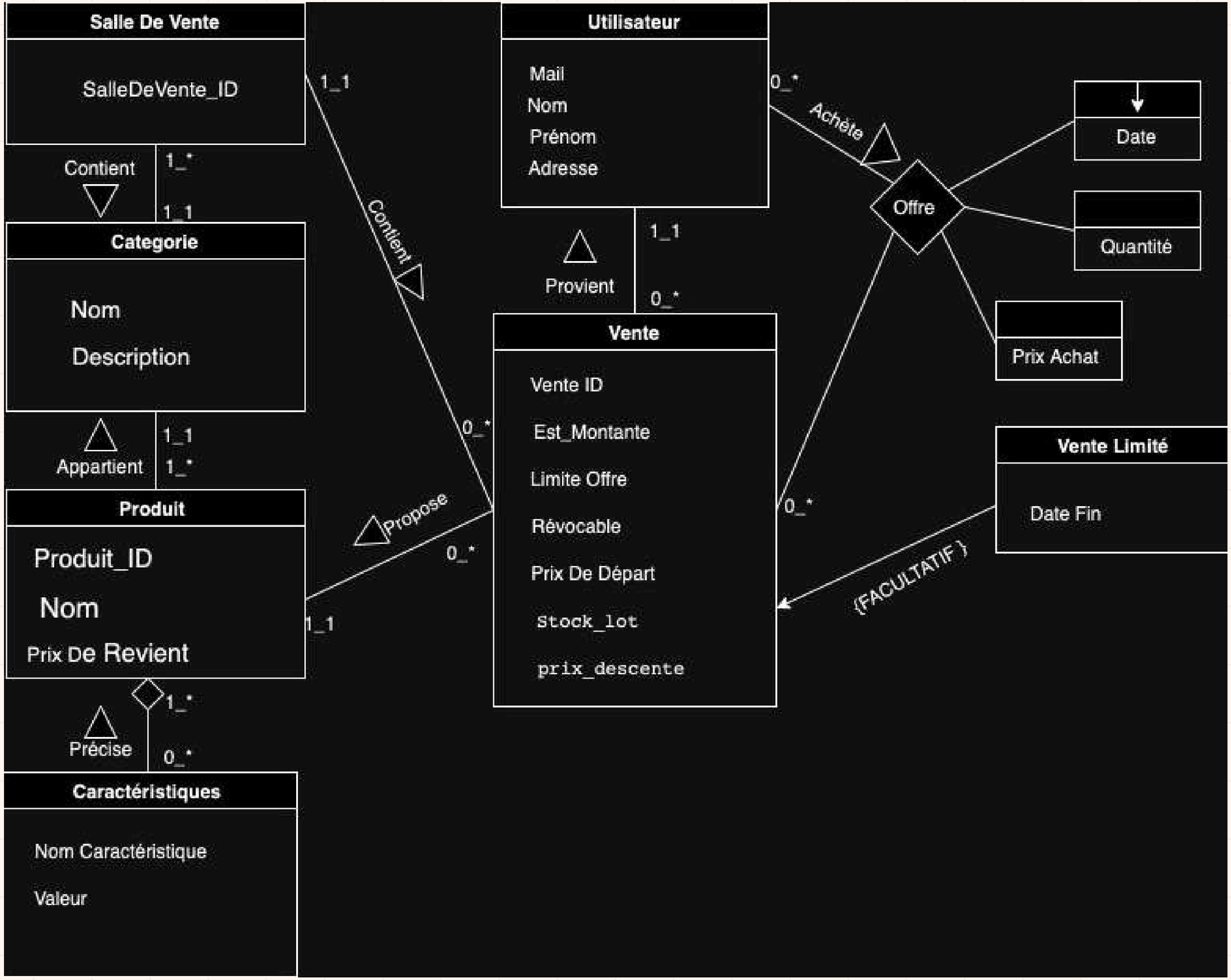
— SI LIMITE_OFFRE EST ACTIVÉ, UN UTILISATEUR NE PEUT FAIRE QU'UNE SEULE OFFRE.

— PAR DÉFAUT, LES VENTES SONT MONTANTES, NON RÉVOCABLES, SANS LIMITATION DE TEMPS, ET PERMETTENT PLUSIEURS OFFRES.

— SI UNE DATE DE FIN EST DÉFINIE, ELLE DOIT ÊTRE SUPÉRIEURE À LA DATE ACTUELLE.

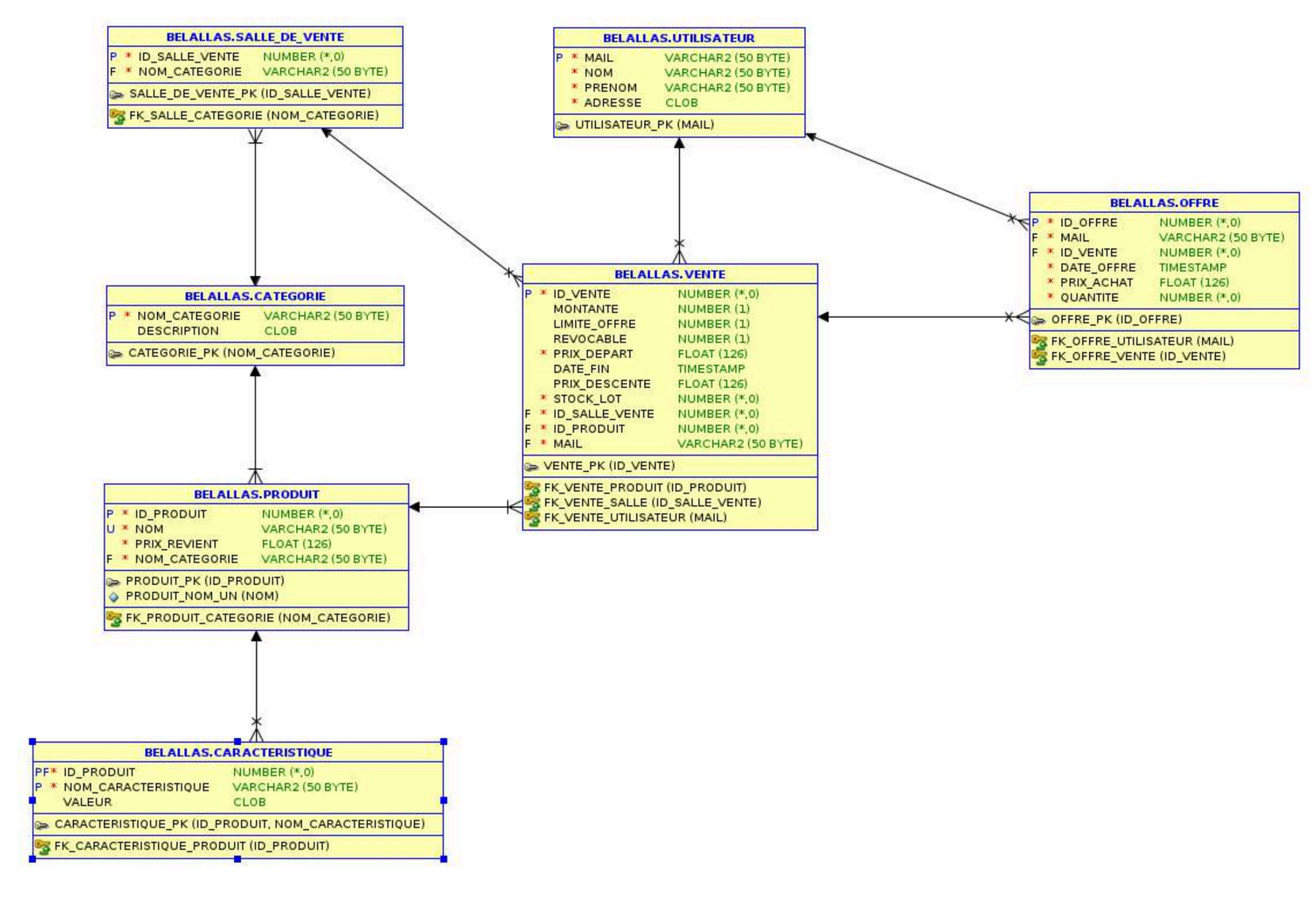
—LE PRIX PROPOSÉ DOIT ÊTRE STRICTEMENT SUPÉRIEUR AU PRIX DE DÉPART.

CONCEPTION UML (ENTITÉS/ASSOCIATIONS)


Salle De Vente
SalleDeVente_ID
Contient
1_*
1_1
Categorie
Nom
Description
Appartient
1_1
1_*
Produit
Produit_ID
Nom
Prix De Revient
Précise
1_*
0_*
Caractéristiques
Nom Caractéristique
Valeur
1_1
Contient
0_*
Propose
0_*
1_1
Utilisateur
Mail
Nom
Prénom
Adresse
Provient
1_1
0_*
Vente
Vente ID
Est_Montante
Limite Offre
Révocable
Prix De Départ
Stock_lot
prix_descente
0_*
Achète
Offre
Date
Quantité
Prix Achat
Vente Limité
Date Fin
{FACULTATIF}

Passage du modèle Entité/Association vers un schéma Relationnel

Schéma Relationnel de la Base de Données



GESTION DES TRANSACTIONS

CONTRÔLE DES TRANSACTIONS

Les transactions permettent de regrouper plusieurs opérations SQL en une seule unité cohérente, assurant l'intégrité des données même en cas d'échec. Dans notre projet, nous avons désactivé l'option autocommit pour garantir un contrôle explicite sur les transactions. Cela nous permet de valider ou d'annuler manuellement les modifications en fonction du succès ou de l'échec des opérations.

MODE SERIALIZABLE

Dans notre projet, nous avons configuré les transactions en mode d'isolation Serializable, le niveau le plus strict. Ce mode garantit que les transactions s'exécutent de manière totalement isolée, évitant les problèmes comme les lectures fantômes ou les écritures concurrentes. Bien que cela puisse légèrement affecter les performances, il assure une cohérence maximale des données, ce qui est essentiel dans un système d'enchères où la concurrence est élevée.

PREPARED STATEMENTS

Les Prepared Statements sont utilisés dans toutes nos transactions pour sécuriser les interactions avec la base de données. Ils protègent contre les attaques par injection SQL en séparant les commandes SQL des données utilisateur. De plus, ils améliorent les performances en permettant la réutilisation de requêtes précompilées, rendant le système à la fois plus rapide et plus sûr.

Transactions en Action : Illustration par le Code

```java
public void start(Stage primaryStage) {
    try {
        dbConnection = DriverManager.getConnection(
                SQLRequests.SQL_SERVER_LOCATION, SQLRequests.SQL_USER, SQLRequests.SQL_PASSWORD);
        dbConnection.setAutoCommit(false);
        dbConnection.setTransactionIsolation(Connection.TRANSACTION_SERIALIZABLE);
    }
    catch (SQLException e) {
        e.printStackTrace();
    }
    primaryStage.setTitle("Baie-électronique");
    mainScene = new Scene(new HomeComponent());
    primaryStage.setScene(mainScene);
    primaryStage.show();
}
```

Main.java

```java
"""";

public static final String SQL_INSERT_NEW_SALE = """  2 usages
INSERT INTO Vente (montante, limite_offre, revocable, prix_depart,
date_fin, prix_descente, Stock_lot, ID_salle_vente, ID_produit, mail)
VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
""";

public static final String SQL_INSERT_NEW_PRODUCT = """  1 usage
INSERT INTO Produit (prix_revient, Nom_categorie, stock) VALUES (?, ?, ?)
""";

public static final String SQL_RETRIEVE_OFFERS_BY_INTEREST = """  1 usage
SELECT * FROM Offre WHERE ID_Vente = ? ORDER BY prix_achat/quantite DESC
```

SQLRequests.java

```java
public static boolean pushNewSale(Sale sale) {  1 usage  Alexis L. +1
    Connection connection = Main.getDbConnection();
    try {
        if(sale instanceof LimitedSale) {
            PreparedStatement statement = connection.prepareStatement(SQLRequests.SQL_INSERT_NEW_SALE);
            statement.setBoolean( parameterIndex: 1, sale.isIncrease());
            statement.setBoolean( parameterIndex: 2, sale.isLimitedOffers());
            statement.setBoolean( parameterIndex: 3, sale.isCancelable());
            statement.setFloat( parameterIndex: 4, sale.getPrixDepart()); // Utilisation de `getStartingPric
            statement.setDate( parameterIndex: 5, ((LimitedSale) sale).getEndDate());
            statement.setFloat( parameterIndex: 6, sale.getPriceDecreaseAmount());
            statement.setInt( parameterIndex: 7, sale.getQuantity()); // Utilisation de `getQuantity`
            statement.setInt( parameterIndex: 8, sale.getSaleRoomId());
            statement.setInt( parameterIndex: 9, sale.getProductId());// ID de la salle de vente
            statement.setString( parameterIndex: 10, sale.getOwner());
            statement.execute();
            connection.commit();
            statement.close();
        }
        else {
            PreparedStatement statement = connection.prepareStatement(SQLRequests.SQL_INSERT_NEW_SALE);
            statement.setBoolean( parameterIndex: 1, sale.isIncrease());
```

```java
        }
        else {
            PreparedStatement statement = connection.prepareStatement(SQLRequests.SQL_INSERT_NEW_SALE);
            statement.setBoolean( parameterIndex: 1, sale.isIncrease());
            statement.setBoolean( parameterIndex: 2, sale.isLimitedOffers());
            statement.setBoolean( parameterIndex: 3, sale.isCancelable());
            statement.setFloat( parameterIndex: 4, sale.getPrixDepart()); // Utilisation de `getStartingPric
            statement.setDate( parameterIndex: 5, x: null);
            statement.setFloat( parameterIndex: 6, sale.getPriceDecreaseAmount());
            statement.setInt( parameterIndex: 7, sale.getQuantity()); // Utilisation de `getQuantity`
            statement.setInt( parameterIndex: 8, sale.getSaleRoomId());
            statement.setInt( parameterIndex: 9, sale.getProductId());// ID de la salle de vente
            statement.setString( parameterIndex: 10, sale.getOwner());
            statement.execute();
            connection.commit();
            statement.close();
        }
        return true;
    } catch (SQLException e) {
        handleSQLException(connection, e);
    }
    return false;
```

Requests.java

Démonstration :