Benjamin BEAUCAMP
Muruo WANG

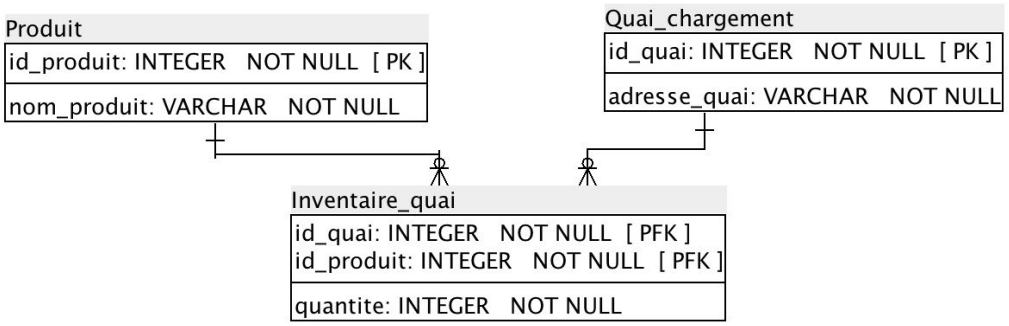
TP2 – Base de données

Introduction :

Dans ce TP, nous allons dans un premier temps mettre en oeuvre des triggers avec **plpgsql** afin d'automatiser la mise à jour des stocks lors de l'ajout, la suppression et la mise à jour d'ordres de livraison. Dans un deuxième temps, nous allons nous connecter à la base de données avec JAVA afin d'effectuer des requêtes pour créer un bordereau de livraison.

1) Gestion du stock

Pour la gestion du stock, nous avions déjà les 3 tables proposées dans le rapport (Produit, QuaiProduit et Quai) mais sous des noms différents : produit, inventaire_quai et quai_chargement. Nous avons décidé de garder nos tables (ainsi que les noms de colonnes).



2) Fonctions TRIGGER avec PL/SQL

Pour insérer les fonctions trigger et les triggers, nous avons utilisé l'interface de PgAdmin, qui évite certaines lignes de code.

a) Trigger sur l'insertion et la modification d'une ligne de livraison

On ajoute un trigger sur les lignes de liste_commande (la table contenant les produits commandés par les clients) qui possède id_ordre comme clé étrangère. En effet, pour associer un produit à une mission, il faut déjà que l'ordre de mission existe donc quand un ordre de mission sera ajouté, aucun produit n'y sera associé : un trigger sur la table ordre_mission ne servirait à rien. On met donc un trigger sur liste_commande pour que les stocks soient mis à jour quand id_ordre d'un produit commandé change (typiquement passe de NULL à une valeur existante d'id_ordre). Ensuite, il suffit d'aller récupérer le numéro de quai associé à l'ordre de mission et d'aller mettre à jour la quantité.

```
DECLARE
numero_quai integer ;
BEGIN
    SELECT id_quai
    FROM ordre_mission
    WHERE ordre_mission.id_ordre = new.id_ordre
    INTO numero_quai ;
    UPDATE inventaire_quai
    SET  quantite =  quantite - new.quantite
    WHERE new.id_produit =  inventaire_quai.id_produit
    AND inventaire_quai.id_quai = numero_quai ;
RETURN  new;
END;
```

b) Trigger sur la suppression d'une ligne de livraison

On ajoute un trigger sur les lignes de ordre_mission qui possèdeiid_quai et id_ordre). Il faut bien penser à mettre les valeurs de id_ordre à NULL dans liste_commande avant de pouvoir supprimer l'ordre de commande (sinon il y a une violation des contraintes d'intégrité à cause des clés étrangères).

```
DECLARE
liste_produit CURSOR IS
SELECT id_produit,quantite  FROM liste_commande
WHERE id_ordre = old.id_ordre ;
BEGIN
FOR produit in liste_produit   LOOP
    UPDATE inventaire_quai
    SET  quantite =  quantite + produit.quantite
    WHERE produit.id_produit =  inventaire_quai.id_produit
    AND inventaire_quai.id_quai = old.id_quai;
    UPDATE liste_commande
    SET id_ordre = NULL
    WHERE produit.id_produit =   id_produit
    AND id_ordre = old.id_ordre ;
END LOOP ;
RETURN new;
END;
```

c) Test des triggers

On crée un nouvel ordre de mission chargeant à Soumoulou (quai n°1). On lui associe une commande de 25 kg de fraises, et le stock est bien mis à jour (on avait 75kg initialement) :

id_produit integer	id_quai integer	quantite integer	nom_produit character varying
6	1	50	Fraises

On supprime ensuite cet ordre de commande, et la quantité de fraises est bien restituée :

	id_produit integer	id_quai integer	quantite integer	nom_produit character varying
1	6	1	75	Fraises

3) JAVA

On commence par charger le driver avec un premier bloc try/catch :

```
public class BordereauDeLivraison {

    public static void main(String[] argv) {
        try {
            Class.forName("org.postgresql.Driver");

        } catch (java.lang.ClassNotFoundException e) {
            System.err.println("ClassNotFoundException :" + e.getMessage());
            // Logger.getLogger(BordereauDeLivraison.class.getName()).log(Level.SEVERE, null, ex);

        }
```

Dans un deuxième bloc try/catch, on exécute deux requêtes :

- la première pour l'affichage du haut du bordereau (nom du chauffeur, adresse du quai de chargement et date de chargement)
- la deuxième pour l'affichage de toutes les marchandises à charger

Finalement, on libère le driver avant de terminer la fonction.

```java
// connexion à la base
try {
    Connection connect = DriverManager.getConnection("jdbc:postgresql://127.0.0.1:5432/TP1", "postgres", [redacted]);

    int id_ordre = 1;
    // écrire des requêtes  1
    String query = "select * from ordre_mission join quai_chargement using(id_quai) join chauffeur using(id_chauffeur) where id_ordre = ?";
    PreparedStatement stmt = connect.prepareStatement(query);
    stmt.setInt(1, id_ordre);
    ResultSet res = stmt.executeQuery();
    while (res.next()) {
        System.out.println("Nom de chauffeur     :    " + res.getString("nom_chauffeur"));
        System.out.println("Prenom de chauffeur  :    " + res.getString("prenom_chauffeur"));
        System.out.println("Quai de chargement   :    " + res.getString("adresse_quai"));
        System.out.println("Date de chargement   :    " + res.getString("date_heure_chargement"));
    }
    stmt.close();

    // écrire des requêtes  2
    String query1 = "select nom_entreprise,  adresse_depot , nom_produit, quantite, date_livraison from liste_commande \n"
            + "join commande using(id_commande)\n"
            + "join produit using(id_produit)\n"
            + "join entreprise using(siret)\n"
            + "join depot using(id_depot)\n"
            + "where id_ordre = ? ";
    PreparedStatement stmt1 = connect.prepareStatement(query1);
    stmt1.setInt(1, id_ordre);
    ResultSet res1 = stmt1.executeQuery();
    while (res1.next()) {
        System.out.println();
        System.out.println("Nom de l'entreprise    :    " + res1.getString("nom_entreprise"));
        System.out.println("Adresse de depôt       :    " + res1.getString("adresse_depot"));
        System.out.println("date livraison         :    " + res1.getString("date_livraison"));
        System.out.println("Nom de produit         :    " + res1.getString("nom_produit"));
        System.out.println("Quantité de produit    :    " + res1.getString("quantite"));
    }
    stmt.close();

    connect.close();
    Driver theDriver = DriverManager.getDriver("jdbc:postgresql://127.0.0.1:5432/TP1");
    DriverManager.deregisterDriver(theDriver);

} catch (SQLException ex) {
    System.err.println("SQLException :  " + ex.getMessage());
}
```

On obtient le résultat suivant dans la console :

```
Nom de chauffeur     :    DENT
Prenom de chauffeur  :    Arthur
Quai de chargement   :    5 allee Beltegeuse    Soumoulou
Date de chargement   :    2019-11-15 06:00:00

Nom de l'entreprise    :    Scoiatel
Adresse de depôt       :    5 allee des remouleurs,Pau
date livraison         :    2019-11-15
Nom de produit         :    Melons
Quantité de produit    :    300

Nom de l'entreprise    :    Kalkstein
Adresse de depôt       :    7 impasse des 3 saules,Strasbourg
date livraison         :    2019-11-18
Nom de produit         :    Peches
Quantité de produit    :    300

Nom de l'entreprise    :    Scoiatel
Adresse de depôt       :    5 allee des remouleurs,Pau
date livraison         :    2019-11-15
Nom de produit         :    Pommes de terre
Quantité de produit    :    1000
```

Conclusion :

Dans ce TP, on a étudié comment générer des Trigger en **plpgsql** avec pgAdmin. En utilisant un driver “org.postgresql.Driver”, ça nous permet de connecter la base avec l’IDE Netbeans et JAVA.Maven nous permet de télécharger le Jar facilement en éditant dans le fichier pom.xml.