SANCHEZ-PORRO
Aurélien 3B

15/10/2023

Compte rendu SAE S5

Projet IRIT (groupe 2)

Table des matières

Présentation du projet

Présentation générale

L'IRIT (Institut de Recherche en Informatique de Toulouse) a développé une application java permettant de générer des arbres algébriques à partir d'une requête SQL sur plusieurs bases de données hétérogènes. Cependant, les arbres créés étaient affichés en texte dans l'invite de commandes.

Le but de la SAE est donc de créer un outil permettant d'afficher les arbres générés d'une façon graphique. Les étudiants en formation initiale ont décidé de faire une application JavaFX pour le front, qui récupérerait les arbres générés par l'application de l'IRIT pour les afficher.

Existant

À l'arrivée des alternants, le squelette de l'application avait été fait avec JavaFX, et permettait d'afficher trois arbres prédéfinis en même temps dans une fenêtre. Pour générer l'arbre, l'outil « fxgraph » à été utilisé.

Pour la récupération d'arbre, le client (M. Peninou) a demandé à ne pas modifier le code de l'IRIT. Les étudiants en formation initiale ont donc créé un « code miroir » pour pouvoir réutiliser les fonctionnalités du code de l'IRIT et pouvoir en ajouter ou les modifier.

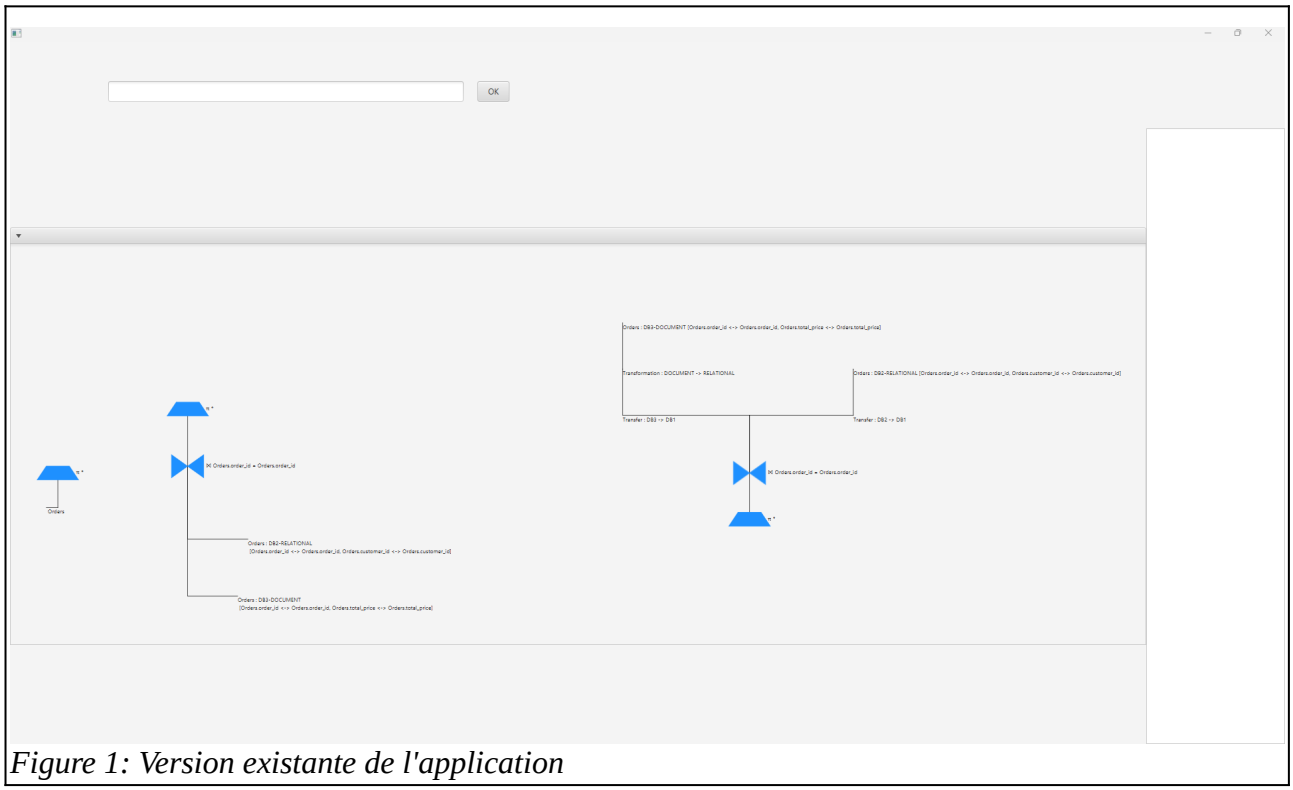

Figure 1: Version existante de l'application

II - Travail effectué

1- Génération de l’arbre

En début de première semaine, j’ai contribué à la User Story « Obtenir un arbre algébrique logique à partir d'une requête SQL » en travaillant sur la génération d’un arbre graphique à partir de l’arbre fourni par l’application de l’IRIT. Chaque élément (sélection, projection, …) de l’arbre généré par l’application de l’IRIT possède, en tant qu’attribut, son (ou ses) enfants. Par exemple, une **sélection** qui vient après une **jointure** entre **deux tables**, a accès à la **jointure**, qui elle même a accès aux **deux tables** jointes.

J’ai donc fait une fonction récursive pour la génération de l’arbre. Cette fonction prends en paramètre l’élément à ajouter à l’arbre, la cellule de la mère de l’élément, et le graphe dans lequel se trouve l’arbre. Elle permet de :

- créer une cellule dans l’arbre pour un élément
- relier cette cellule à la cellule mère
- vérifier si cet élément à un enfant et dans ce cas relancer la fonction avec en paramètres l’enfant, la cellule créée et le graphe.

```
private void makeTree(ETreeNode child, Model model, ICell previousCell) {
    // La projection est la seule cellule sans parent
    if (previousCell == null) {
        // On crée la cellule
        ProjectionCell projection = new ProjectionCell(child.toString());

        // On l'ajoute au model fourni en paramètres
        model.addCell(projection);

        // On verifie si il a des enfants et on réexecute la methode
        if (child.getChild().length > 0) {
            makeTree(child.getChild()[0], model, projection);
        }
    } else {
        switch (child.getClass().getSimpleName()) {
             // Si l’élément est une jointure
            case "EJoin":
                // On crée la cellule
                JointureCell jointure = new JointureCell(child.toString());

                // On l'ajoute au model
                this.addCellAndEdge(model, jointure, previousCell);

                // On réexecute la methode pour les deux enfants
                makeTree(((EJoin) child).getLeftChild(), model, jointure);
                makeTree(((EJoin) child).getRightChild(), model, jointure);
                break;
```

Figure 2: Fonction permettant la génération de l'arbre

```
            // Si l'élément est une selection
            case "ESelection":
               char[] charArray = child.toString().toCharArray();
               AtomicReference<String> text = new AtomicReference<>("");
               if (charArray[0] == '(' && charArray[charArray.length-1] == ')')
               {
                   IntStream.range(1, charArray.length-1).forEachOrdered(index
                   -> text.updateAndGet(v -> v + charArray[index]));
               } else {
                   text.updateAndGet(v -> child.toString());
               }
               SelectionCell selection = new
                                    SelectionCell(String.valueOf(text));

               this.addCellAndEdge(model, selection, previousCell);

               if (child.getChild().length > 0) {
                   makeTree(child.getChild()[0], model, selection);
               }
               break;
            // Si l'élément est un transfert
            case "ETransfer":
               TransferCell transfer = new TransferCell(child.toString());

               this.addCellAndEdge(model, transfer, previousCell);

               if (child.getChild().length > 0) {
                   makeTree(child.getChild()[0], model, transfer);
               }
               break;
            // Si l'élément est une transformation
            case "ETransformation":
               TransformCell transform = new TransformCell(child.toString());

               this.addCellAndEdge(model, transform, previousCell);

               if (child.getChild().length > 0) {
                   makeTree(child.getChild()[0], model, transform);
               }
               break;
           // Dans le cas ou ce n'est pas une cellule vérifiée précédemment, on ne crée
            qu'une cellule de texte
           default:
               LabelCell label;
               if (child.getStore() == null){
                   label = new LabelCell(child.toString().toUpperCase(), null);
               }else {
                   String str = ""+child.getStore();
                   String[] tableStr = str.split(" ", 2);
                   label = new LabelCell(child.toString().toUpperCase(), tableStr[0]+"\
                   n"+tableStr[1]);
               }

               this.addCellAndEdge(model, label, previousCell);

               if (child.getChild().length > 0) {
                   makeTree(child.getChild()[0], model, label);
               }
               break;
        }
    }
}
```

Figure 2 (suite): Fonction permettant la génération de l'arbre

2- Ajouts dans l'application et gestion du Github

Après avoir fait la génération de l'arbre, j'ai créé les classes permettant d'afficher les transferts (transferCell.java) et les transformations (transformCell.java). Contrairement aux jointures, sélections et projections, ces cellules n'avaient pas d'équivalent dans un arbres algébrique. Le design pour le transfert a été simple a créer, contrairement à celui de la transformations qui a eu 3 versions.

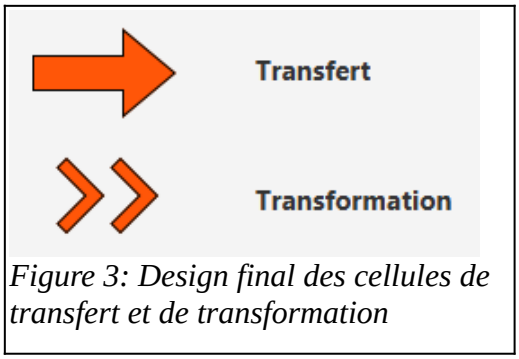

Figure 3: Design final des cellules de transfert et de transformation

En fin de première semaine, j'ai pu merge les différentes branches du repository GitHub afin de faire une release V1. Lors du merge des branches, plusieurs conflits sont apparus. Je me suis occupé de résoudre ces conflit, mais cela a créé des erreurs dans le code que je me suis empressé de corriger. J'ai après pu sortir une release « V1 », et supprimer les branches merged ou inutilisées.

Durant la seconde semaine, un rendez-vous a été organisé avec M. Peninou pour lui faire une démonstration de la V1 de l'application. Dans son retour d'expérience, il nous a fait part de plusieurs conseils pour améliorer l'application et son code. Un des conseils était de créer une variable globale pour gérer facilement la couleur des cellules. Cette tache a été incorporée dans l'issue « Améliorations IHM mineures » Nous avions déjà un fichier CSS pour gérer le style de l'application. J'ai donc ajouté dans ce fichier une section pour les objets de style «color-polygon», et importé le CSS dans les fichiers FXML.

```
.color-polygon {
    -fx-fill: -fx-main-color;
    -fx-stroke: black;
    -fx-stroke-type: inside;
}
```

Figure 4: partie du fichier CSS gérant la couleur et les contours des cellules

```
<AnchorPane prefHeight="400.0" prefWidth="600.0" xmlns="http://javafx.com/javafx/17.0.2-ea"
xmlns:fx="http://javafx.com/fxml/1">
   <Group fx:id="transferGroup" layoutX="300.0" layoutY="170.0">
      <!-- Import du CSS -->
      <stylesheets>
         <URL value="@../../css/styles.css"/>
      </stylesheets>
      <!-- Création du polygone, dont attribution de "color-polygon" -->
      <Polygon fx:id="transferPolygon" styleClass="color-polygon" points="-10.0, 15.0, -10.0,
               35.0, 40.0, 35.0, 40.0, 50.0, 70.0, 25.0, 40.0, 0.0, 40.0, 15.0"/>
      <Label fx:id="transferLeftLabel" alignment="CENTER_RIGHT" contentDisplay="RIGHT"
             layoutX="-17.0" layoutY="17.0" text="left" textAlignment="RIGHT" />
      <Label fx:id="transferRightLabel" layoutX="75.0" layoutY="17.0" text="right" />
   </Group>
</AnchorPane>
```

Figure 5: Code FXML de la cellule de Transfert

III – Résultat

À La fin des deux semaines, nous avons grandement pu avancer le projet et, après avoir essayé la V1 de l'application, M. Peninou avait l'air satisfait de notre travail. Mes camarades ont pu faire une refonte graphique de l'application, ainsi que l'ajout de nouvelles fonctionnalités. Les arbres sont générés à partir de la requête entrée, et sont affichées dans différentes fenêtres. Je suis satisfait du travail que nous avons pu fournir durant ces deux semaines.

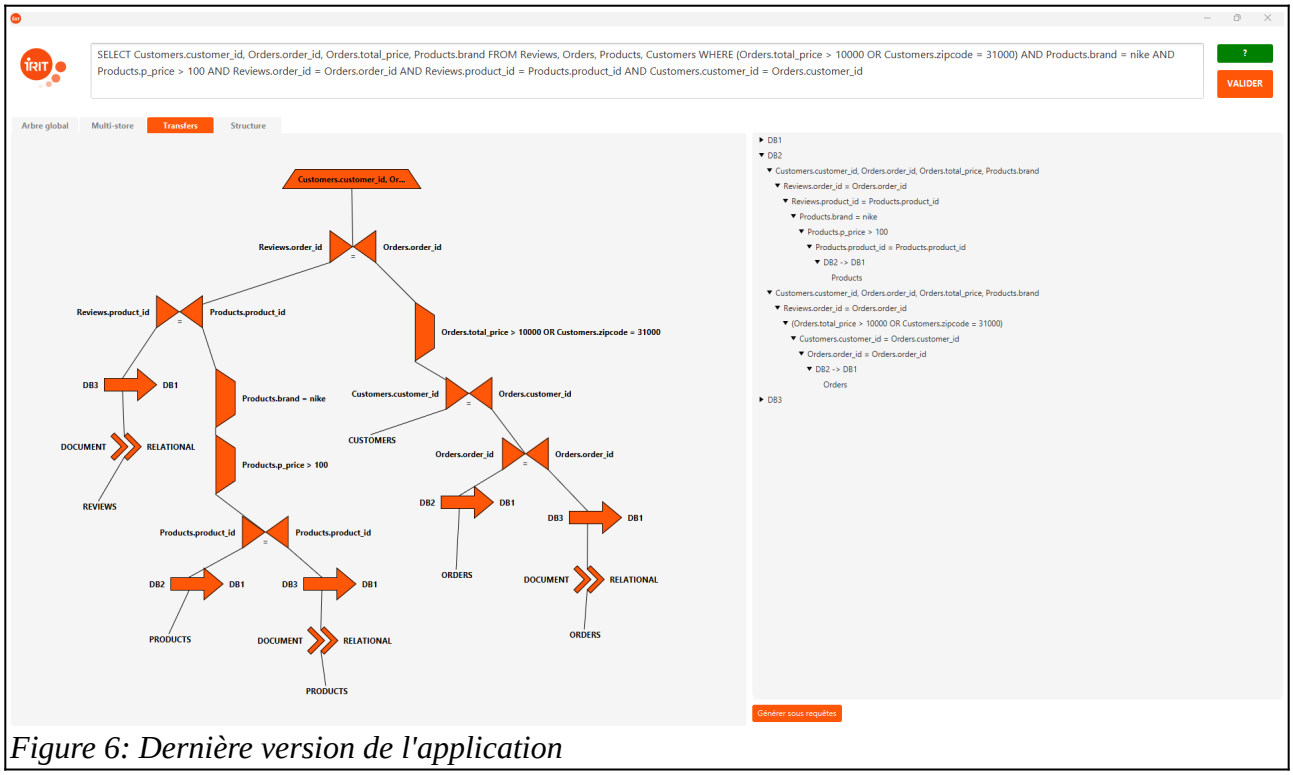

Figure 6: Dernière version de l'application