

Compte rendu d'activité

Projet IRIT (Polyglot IHM)

COURREJOU Matthieu

BUT 3 Informatique



Table des matières

| | | |
|------|---|---|
| I. | Introduction | 3 |
| II. | Missions | 3 |
| a. | Tâche n°1 : Refonte graphique UI..... | 3 |
| b. | Tâche n°2 : Séparation de l'arborescence de transferts par base d'origine | 5 |
| c. | Tâche n°3 : Création de sous requêtes à partir de l'arbre de transferts | 7 |
| d. | Tâche n°4 : Amélioration de la qualité du code | 8 |
| III. | Conclusion..... | 9 |

I. Introduction

Les polystores sont de systèmes de gestions de données regroupant divers types de stockages, tels que du relationnel, document ou du graphe. De par la nature hétérogène de ces données, il est complexe d'interroger ce type de système. Afin de faciliter cela, l'équipe SIG de l'IRIT a développé une solution reposant sur une vue logique sous forme d'arbre algébrique du polystore. Les utilisateurs interrogent cette vue logique, et le système réécrit les requêtes pour accéder aux données des sous-systèmes, en tenant compte de la répartition des données, et des transferts nécessaires.

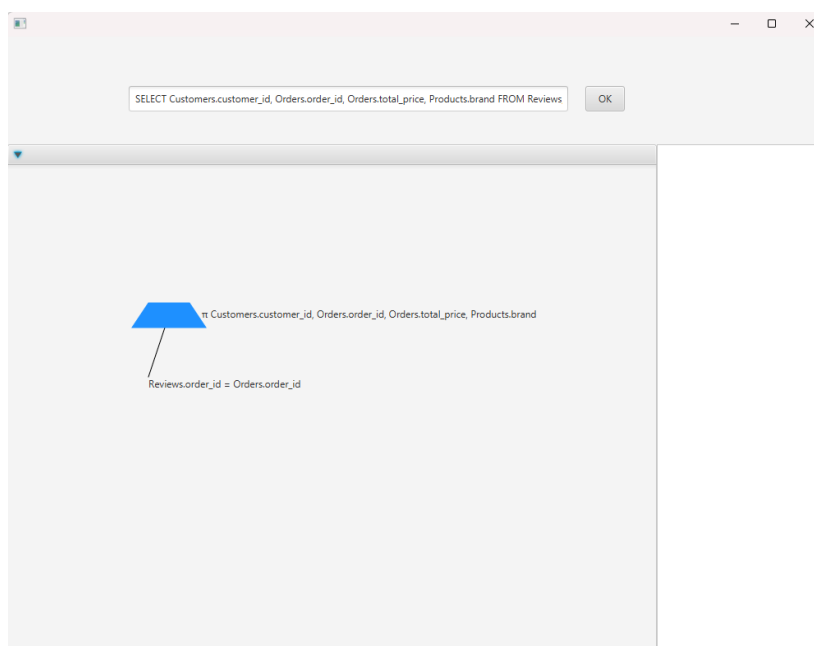
À mon arrivée sur le projet, l'existant sous forme d'IHM était très rudimentaire, et la fonctionnalité de génération d'arbre était en cours de développement. Très vite, des issues furent assignées aux alternants. Pour ma part, la première tâche qui m'a été confiée, a été de faire une refonte graphique de l'application.

II. Missions

a. Tâche n°1 : Refonte graphique UI

Comme indiqué précédemment, l'interface graphique était rudimentaire, seul un seul type d'arbre pouvait être généré, et l'interface ne possédait pas de charte graphique définie.

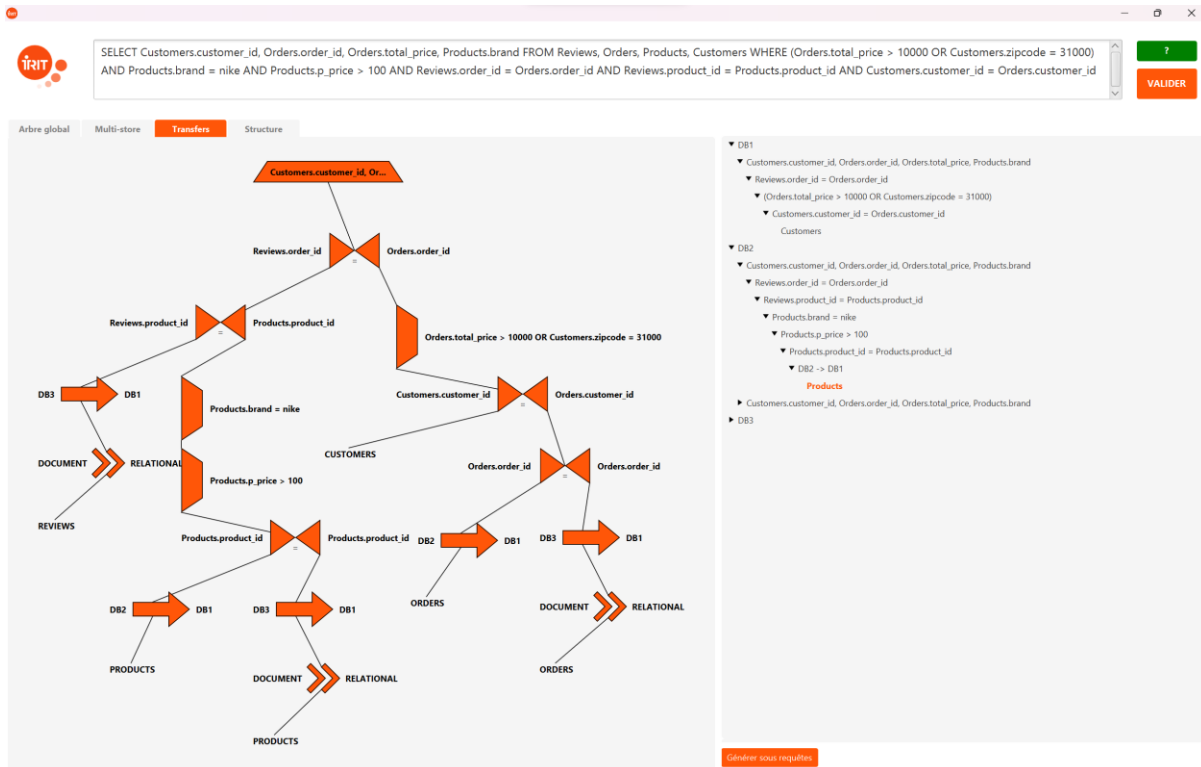
Existante à l'arrivée des alternants :



Remarque : La partie IHM de l'application est développée en Java, en utilisant le framework JavaFX

Afin d'améliorer l'expérience utilisateur, mais aussi pour mieux représenter le client (IRIT), il semblait nécessaire d'améliorer l'aspect visuel. J'ai alors effectué une refonte totale de l'interface utilisateur de l'application.

Résultat de la refonte :



Voici exactement ce que j'ai fait sur cette issue :

- Création de la charte graphique, avec rappel du logo IRIT
- Intégration de la charte graphique en FXML et en CSS
- Ajout des différents onglets afin d'afficher les trois types d'arbres séparément
- Refonte de la structure FXML du projet afin de centrer chaque arbre dans son parent
- Restructuration de l'architecture de dossier, séparant la vue de la logique métier

Conclusion : Pas de grandes difficultés sur cette tâche si ce n'est l'apprentissage de JavaFX que je n'avais jamais utilisé, ainsi que les sélecteurs CSS spécifiques au Framework. Aussi, le centrage des arbres m'a pris un peu de temps, car je n'arrivai pas à trouver la bonne approche en FXML pour arriver au résultat souhaité.

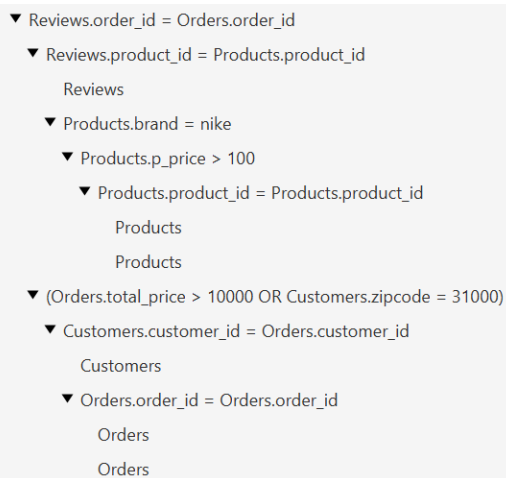
Lien de l'issue : <https://github.com/AurelienSP/SAE-S5-IRIT-G2/issues/29>

b. Tâche n°2 : Séparation de l'arborescence de transferts par base d'origine

L'application dispose de trois vues sous forme d'arbres. Une de ces vues est celle de transferts, indiquant quelles données sont transférées depuis leur base d'origine vers la base d'arrivée. Le client souhaitait pouvoir séparer les arborescences par base d'origine des données.

Vu que nous disposions déjà d'une vue d'arborescence, qui se contentait de représenter l'arbre sans le modifier, il m'a suffi de changer l'algorithme de création de l'arborescence, séparant ainsi les branches de l'arbre par base d'origine.

Arborescence de transfert avant l'issue :



```
▼ Reviews.order_id = Orders.order_id
  ▼ Reviews.product_id = Products.product_id
    Reviews
  ▼ Products.brand = nike
    ▼ Products.p_price > 100
      ▼ Products.product_id = Products.product_id
        Products
        Products
  ▼ (Orders.total_price > 10000 OR Customers.zipcode = 31000)
    ▼ Customers.customer_id = Orders.customer_id
      Customers
    ▼ Orders.order_id = Orders.order_id
      Orders
      Orders
```

Remarque : Ici l'arborescence a exactement la même structure, et les branches ne sont pas séparées pas base de données d'origine.

Résultat final :

```
▼ DB1
  ▼ Customers.customer_id, Orders.order_id, Orders.total_price, Products.brand
    ▼ Reviews.order_id = Orders.order_id
      ▼ (Orders.total_price > 10000 OR Customers.zipcode = 31000)
        ▼ Customers.customer_id = Orders.customer_id
          Customers
  ▼ DB2
    ▼ Customers.customer_id, Orders.order_id, Orders.total_price, Products.brand
      ▼ Reviews.order_id = Orders.order_id
        ▼ Reviews.product_id = Products.product_id
          ▼ Products.brand = nike
            ▼ Products.p_price > 100
              ▼ Products.product_id = Products.product_id
                ▼ DB2 -> DB1
                  Products
    ▼ Customers.customer_id, Orders.order_id, Orders.total_price, Products.brand
      ▼ Reviews.order_id = Orders.order_id
        ▼ (Orders.total_price > 10000 OR Customers.zipcode = 31000)
          ▼ Customers.customer_id = Orders.customer_id
            ▼ Orders.order_id = Orders.order_id
              ▼ DB2 -> DB1
                Orders
  ▼ DB3
    ▼ Customers.customer_id, Orders.order_id, Orders.total_price, Products.brand
      ▼ Reviews.order_id = Orders.order_id
        ▼ Reviews.product_id = Products.product_id
          ▼ DB3 -> DB1
            ▼ DOCUMENT -> RELATIONAL
              Reviews
    ► Customers.customer_id, Orders.order_id, Orders.total_price, Products.brand
    ► Customers.customer_id, Orders.order_id, Orders.total_price, Products.brand
```

Remarque : Les arborescences sont affichées à l'aide d'un composant JavaFX TreeView. Chaque branche de l'arbre est classée à partir de sa base de données d'origine

Voici exactement ce que j'ai effectué sur cette issue :

- Rajout d'une logique spécifique de génération d'arbre lors de la sélection de l'onglet de transfert.
- Création d'un algorithme de parcours de l'arbre à partir des feuilles, remontant vers la racine.

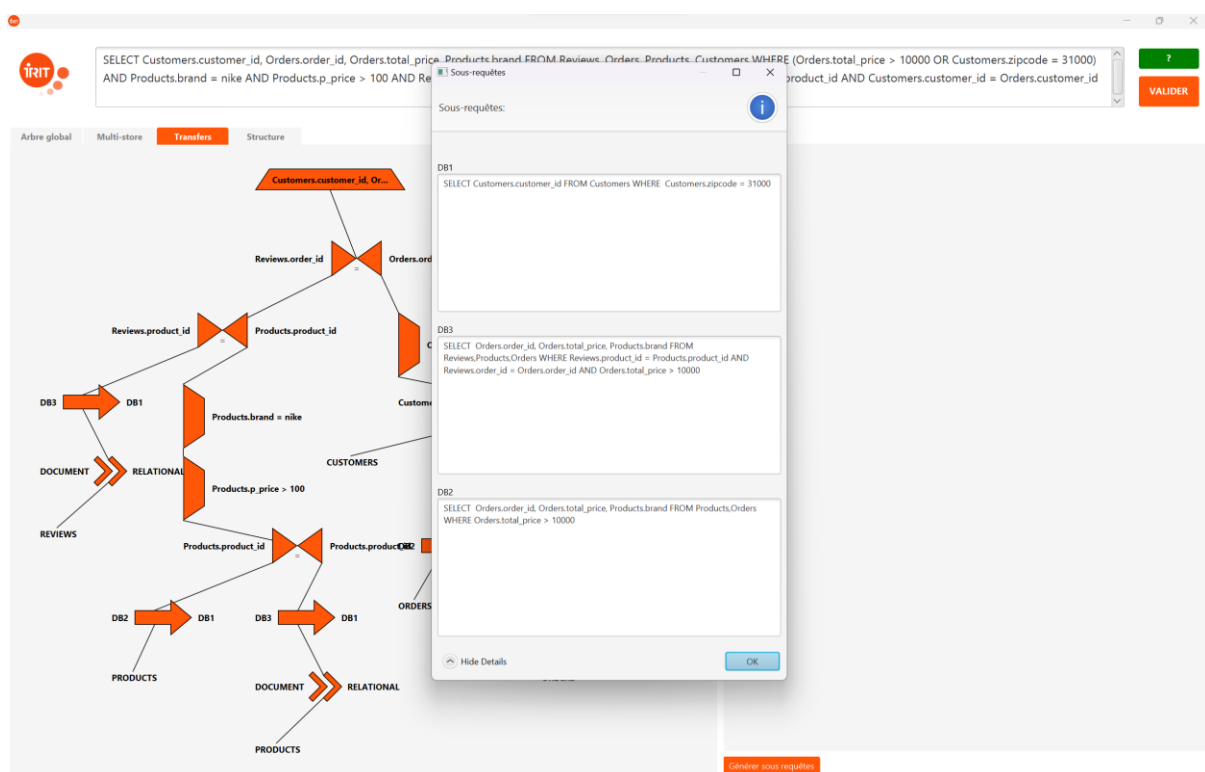
Conclusion : Les difficultés rencontrées sur cette tâche étaient surtout d'ordre algorithmique, car il n'était possible de trouver la base de données source d'une branche que sur son nœud feuille, donc pour pouvoir ajouter une branche dans la catégorie correspondant à sa base de données sur la TreeView (arborescence), il me fallait parcourir l'arbre jusqu'à la feuille, et ensuite créer la TreeView de l'arbre en le remontant depuis la feuille vers la racine.

Lien de l'issue : <https://github.com/AurelienSP/SAE-S5-IRIT-G2/issues/38>

c. Tâche n°3 : Création de sous requêtes à partir de l'arbre de transferts

En tant qu'utilisateur, je veux cliquer sur les éléments de l'arbre logique afin de générer des sous-requêtes. En effet, le client souhaitait pouvoir générer des sous-requêtes, afin de pouvoir requêter chaque base individuellement. Nous générons une sous-requête pour chaque base de données sollicitée par la requête globale.

Résultat de la tâche :



Travail effectué sur la tâche :

- Création d'une classe avec différentes méthodes permettant la conversion d'une branche de l'arbre en une requête SQL fonctionnelle.
- Création d'un bouton et d'une vue permettant l'affichage d'une fenêtre de dialogue avec les sous-requêtes accessibles en copier/coller.

Conclusion : Cette tâche était réellement la plus difficile que j'ai pu faire au cours de ce projet, car il me fallait trouver comment stocker les informations récupérées sur chaque nœud d'une branche, en retirant les jointures et conditions pointant sur des tables d'autres bases de données, pour ensuite recréer une requête correcte. J'ai pu renforcer mes compétences en matière de array streams, ainsi qu'avec l'utilisation de HashMaps en Java.

Lien de l'issue : <https://github.com/AurelienSP/SAE-S5-IRIT-G2/issues/25>

d. Tâche n°4 : Amélioration de la qualité du code

Étant donné que de nombreuses personnes travaillaient sur le projet, et que la plus grosse partie du code se situait dans un seul contrôleur, il devenait compliqué de s'y retrouver. Aussi, la majeure partie de la logique métier de l'application se situait dans une seule grande méthode de plusieurs centaines de lignes. C'est pourquoi il m'a été demandé d'améliorer la propreté du code.

Exemple d'amélioration :

```
/**
 * Recrée la TreeView en fonction de l'arbre affiché sur le panneau sélectionné.
 *
 * @param newTab Le panneau sélectionné
 */
private void renderTabTreeView(Tab newTab) {
    // Sauvegarde le panneau sélectionné pour utilisations ultérieures
    this.selectedTab = newTab;
    // Re-création de la TreeView en fonction de l'arbre correspondant au panneau sélectionné
    if (this.computedTrees != null){
        switch (newTab.getId()) {
            case "globalTreeTab":
                this.subRequestButton.setDisable(true);
                this.renderTreeView(this.computedTrees[0], separateDB: false);
                break;
            case "multiStoreTreeTab":
                this.subRequestButton.setDisable(true);
                this.renderTreeView(this.computedTrees[1], separateDB: false);
                break;
            case "transferTreeTab":
                this.subRequestButton.setDisable(false);
                this.renderTreeView(this.computedTrees[2], separateDB: true);
                break;
            case "structureTab":
                this.subRequestButton.setDisable(true);
                this.tvNode.setRoot(null);
                break;
        }
    }
}
```

Remarque : L'IDE que j'ai utilisé pour ce projet est JetBrains IntelliJ.

Travail effectué sur la tâche :

- Factorisation de la méthode principale du logiciel en plusieurs méthodes réutilisables.
- Réduction de la duplication de code à l'aide de classes utilitaires ou de méthodes dans le contrôleur.
- Écriture de la Java Doc sur la plupart des méthodes
- Ajout de commentaires sur des sections de code complexes
- Ajout de switches ou d'opérateurs ternaires pour remplacer des if/else trop longs ou superflu

Conclusion : Bien que je sois plutôt satisfait du résultat obtenu à la fin de cette tâche, il me semble que d'autres améliorations sont encore à faire sur la code base afin d'atteindre un niveau de propreté de code permettant une lisibilité et compréhension du code optimale. J'ai malheureusement manqué d'un peu de temps pour faire ces dernières améliorations, et je fais totalement confiance à mes collègues pour continuer cette tâche.

Lien de l'issue : <https://github.com/AurelienSP/SAE-S5-IRIT-G2/issues/27>

III. Conclusion

Pour conclure, je suis satisfait de l'avancement du projet, car la grande majorité des fonctionnalités demandées par le client sont implémentées. De plus, j'ai trouvé le sujet très intéressant, car il représentait un réel défi en matière d'algorithmie, notamment de par l'utilisation d'arbre algébrique, mais aussi de par la notion de polystore avec tout ce que cela implique (E.g : transferts). Je déplore cependant un manque d'implication de certains membres du projet, ne participant que très peu à l'effort du groupe.

Concernant la suite du projet, plusieurs choses restent à faire : suite à une réunion avec le client, ce dernier nous a transmis quelques recettes, qui doivent être implémentées dans le projet. De plus, il faudra étoffer la documentation qui, pour l'instant, reste très sommaire. Une fois ces deux étapes validées, il me semble que le projet sera abouti, à voir cependant si fusionner notre projet et celui du groupe concurrent permettra d'avoir un projet encore plus satisfaisant pour le client.