UNIVERSITÉ DE FRANCHE-COMTÉ

Université de Franche-Comté
UFR Sciences et Techniques

Rapport de projet
Licence informatique – 3ème année

Mise en œuvre d'une application web pour interpréter l'algèbre relationnelle

Réalisé par :
Cédric Poncot
Gatien Continsouzas
Nicolas Courvoisier

Tuteur universitaire :
Frédéric Dadeau

Année 2018-2019

Remerciements

Nous tenons à remercier toutes les personnes ayant contribué de près ou de loin au succès de notre projet tutoré et qui nous ont aidés lors de la rédaction de ce rapport.

Tout d'abord, nous adressons nos remerciements à notre enseignant, M. Dadeau de l'Université des Sciences et des Techniques de Besançon qui est également chercheur dans le département d'informatique des systèmes complexes (DISC) à l'institut FEMTO-ST. Notre professeur a su être à notre écoute et nous donner de précieux conseils à suivre pour mener à bien ce projet. Ses conseils pourront être réutilisés dans notre vie future. Nous tenons également à le remercier pour le temps précieux qu'il a su nous accorder lors des différents entretiens que nous avons eu l'occasion d'avoir.

Enfin, nous tenons à remercier toutes les personnes qui nous ont conseillés et relus lors de la rédaction de ce rapport de projet.

Table des matières

Liste des figures

Glossaire

Relation : Une relation R, sur un ensemble de domaines D1,…,DN, est constituée de deux parties qui sont l'en-tête et le corps. Une relation peut être assimilée à un tableau.

En-tête : ensemble fixé d'attributs.

Corps : ensemble de t-uplets.

T-uplet/enregistrement : correspond à une ligne d'une relation.

Attribut : correspond à une colonne d'une relation.

Cardinalité : correspond au nombre de t-uplet d'une relation.

Degré : correspond au nombre d'attributs d'une relation.

Format JSON : JavaScript Object Notation est un format de données textuelles dérivé de la notation des objets du langage JavaScript.

Parser : Mécanisme qui analyse une structure pour en retenir les informations essentielles.

Drag and drop : Mécanisme qui permet de déplacer une relation avec la souris.

LocalStorage : Espace mémoire permettant de stocker des données dans le navigateur.

Introduction

Durant la troisième année de Licence Informatique à l'UFR Sciences et Techniques de Besançon, les étudiants ont un module « projet tutoré » qui se déroule sur la majeure partie de l'année. Ce projet a pour but de simuler un projet d'entreprise pour que les étudiants se rendent compte de l'exigence du métier de développeur. Cela leur permet de mettre en pratique toutes les connaissances apprises durant les trois années de formation.

Dans ce rapport, nous présenterons notre mission principale qui était de créer une application web pour interpréter l'algèbre relationnelle. L'algèbre relationnelle est enseignée dans le module de « base de données » en première année de licence. Cet aspect est souvent mal compris par les étudiants, et leur parait très abstrait car l'enseignement se déroule uniquement sur feuille et qu'il n'y a pas d'outil facile d'usage pour tester les opérateurs. Cette application aura pour but de servir aux étudiants de Licence 1 Informatique pour appréhender plus facilement le fonctionnement des opérateurs de la théorie relationnelle.

Dans un premier temps, nous présenterons l'algèbre relationnelle ainsi que l'étude du projet : l'explication de l'algèbre relationnelle, les contraintes et le cahier des charges, la gestion des tâches et le modèle de données permettant de saisir l'importance du sujet. Une seconde partie sera consacrée à l'explication des différentes étapes de la réalisation du projet en passant par les mécanismes de création des relations, les diverses fonctions de calculs qui satisfont ceux présents dans la théorie relationnelle et en finissant sur les problèmes que l'on a rencontrés durant la phase de développement.

1. Présentation de l'algèbre relationnelle

La théorie relationnelle a été proposé pour la première fois par Edgar Frank Codd, dans l'article : *A relational Model of Data for Large Shared Data Banks*, CACM, Juin 1970. Depuis, il y a eu très peu de modifications.

L'algèbre relationnelle a pour but d'apprendre le fonctionnement des bases de données relationnelles. On peut donc réaliser les mêmes calculs. L'algèbre relationnelle possède un schéma relationnel composé de relations. Ces relations possèdent des propriétés bien spécifiques. Elles sont composées de deux parties : une en-tête qui est un ensemble d'attribut et un corps qui est un ensemble de t-uplets. Une relation peut donc être assimilée à un tableau. Plus précisément, un t-uplet représente une ligne et un attribut est une colonne. Ces attributs doivent donc être uniques. Egalement, il n'y a pas de possibilité de dupliquer les t-uplets, ils sont uniques. Pour identifier les t-uplets, on fait appel à une clé primaire. Ceci est une groupe d'attributs dont la valeur permet d'identifier de manière unique un t-uplet de la relation. Toutes les relations représentent un ensemble.

Nos relations maintenant définies, on peut réaliser divers calculs sur celles-ci. A partir d'une ou deux relations, un opérateur relationnel permet de calculer une nouvelle relation. On peut constater différents types d'opérateurs, la figure 1 ci-présente illustre tous les opérateurs de l'algèbre relationnelle et présente la syntaxe à respecter pour saisir une requête. On remarque les opérateurs ensemblistes qui sont l'union, l'intersection, la

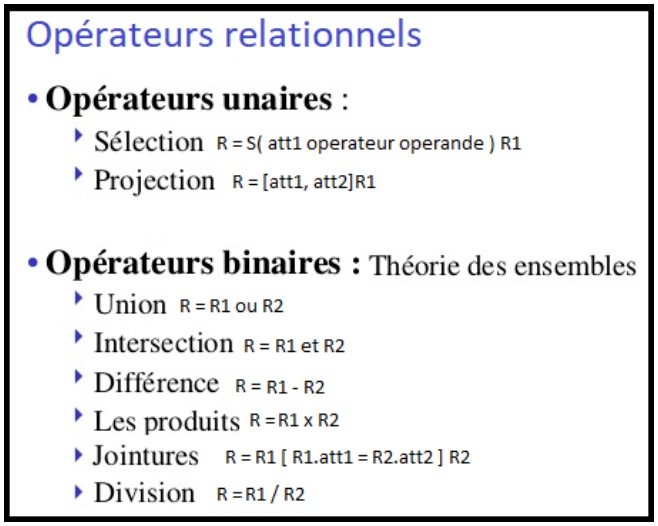

Figure 1 : Les différents opérateurs relationnels

différence et le produit cartésien. Ces opérateurs ont pour but de réaliser un calcul entre deux relations. Par exemple, l'intersection entre deux relations créera une relation avec les t-uplets en commun de ces deux relations. Nous avons aussi des opérateurs relationnels spécifiques : la sélection, la projection et la jointure. La sélection et la projection réalisent des calculs sur une seule relation ; la sélection compare un ou plusieurs attributs avec une valeur et crée une relation résultat. Tandis que la projection permet de définir une relation consistant en l'ensemble de tous les t-uplets de la relation de départ dans laquelle seuls les attributs de projection sont conservés. La jointure permet de relier deux relations entre elles via clé primaire et clé étrangère. Il existe plusieurs types de jointures. La jointure naturelle a pour but de relier deux relations qui possèdent un attribut en commun. L'équijointure de deux relations R1 et R2 est une jointure dont la condition de jointure est l'égalité des valeurs d'attributs att1

et att2, appartenant respectivement aux relations R1 et R2. A l'opposé de la téta-jointure qui est le contraire, la condition de jointure est la différence des valeurs att1 et att2. Et pour finir, nous avons encore les opérateurs dérivés comme la division et la jointure externe. Pour pouvoir être divisées les deux relations doivent avoir en commun au moins un attribut. La division de la relation R1 par une autre relation R2 retourne une relation R3 ainsi que tous les t-uplets de R3 concaténés à R2 apparaissent dans la relation R1. On a : R3 x R2 $\subseteq$ R1.

L'algèbre relationnelle étant définie et expliquée, nous pouvons commencer à réfléchir à une méthode de travail et à la façon dont nous allons mettre en place l'application souhaitée.

2. Etude du projet

Etant quasiment tous issus de la même formation, nous avons pu mettre en pratique nos connaissances et en apprendre de nouvelles puisque le langage de développement utilisé n'avait été appris en cours que par un seul de nous trois. L'algèbre relationnelle ayant été vue par nous tous dans notre cursus, une mise à jour s'imposait pour bien comprendre toutes les fonctionnalités et spécificités de cette algèbre.

A. Contraintes & cahier des charges

Dès le début du projet, nous connaissions les contraintes et le cahier des charges qui étaient donnés avec le sujet. Le sujet était rempli d'informations très importantes qui ont pu nous aiguiller dans la bonne réalisation de l'application souhaitée.

Les contraintes étaient plus technologiques, il était imposé de développer une application web pour que les étudiants de Licence 1 Informatique de Besançon puissent y avoir accès dans le futur. L'application était un site web, donc nos seuls langages autorisés étaient HTML5, CSS3 et JavaScript. Le projet a été principalement développé en JavaScript sans l'utilité d'ajouter un Framework, on parle de VanillaJS. Pour notre tuteur, M. Dadeau, sa principale demande était d'avoir une application fonctionnelle et surtout simple d'utilisation. Avec cette condition, on devait réaliser une application one-page up pour ne pas avoir de redirection. Aussi, ce n'était pas dérangeant d'avoir à réaliser des algorithmes complexes, du moment que l'utilisateur n'ait pas à se demander comment l'application fonctionne. Le mot d'ordre ici était la simplicité pour l'utilisateur : il ne devait jamais se trouver perdu. Un minimum de convivialité était également attendu : c'est pourquoi une partie de la réflexion a été consacrée à simplifier l'utilisation de l'application pour améliorer la navigation de l'utilisateur. Pour nous aider dans le développement, le code devait être orienté objet ce qui aide fortement pour la création de notre modèle de donnée.

Par ailleurs, le cahier des charges demandait de réaliser la majorité des fonctionnalités de l'algèbre relationnelle. Donc, l'application réalisée devra permettre de définir simplement des relations et de saisir son contenu[1]. Ces relations seront visibles à l'écran et pourront être déplaçables. Il y aura tout un ensemble de fonctionnalités sur les relations, comme la

[1] Possibilité d'éditer les attributs, les t-uplets et le nom des relations

possibilité d'ajouter des attributs[2] et des enregistrements[3], mais aussi de pouvoir supprimer les colonnes ou les lignes qui posent problème à l'utilisateur. On pourra aussi ajouter de nouvelles relations et les supprimer. Egalement, puisque les relations peuvent être créées, on pourra réaliser toutes les opérations de la théorie des ensembles[4] mais aussi des opérateurs plus spécifiques tels que la sélection, la projection, les jointures[5] et la division. Ces opérations permettront de réaliser des requêtes saisies par l'utilisateur. A partir de ces requêtes, l'application devra pouvoir générer une nouvelle relation et afficher à l'écran le résultat attendu de manière graphique. Aussi, l'utilisateur doit pouvoir sauvegarder son travail en cours pour pouvoir le reprendre plus tard. Evidemment, il doit aussi pouvoir le recharger quand il revient sur l'application et modifier son travail.

B. Gestion des tâches

Un projet d'une telle envergure était une première pour nous, il fallait d'abord réfléchir à la conception du projet avant de se lancer dans le code.

Dans un premier temps, nous avons pris rendez-vous avec M. Dadeau, celui-ci étant le tuteur du projet, il était nécessaire de savoir exactement ce qu'il attendait de nous. On a tout de suite convenu de réaliser ce projet avec une méthode de travail qui est la méthode agile. Cette méthode avait été vue en cours dans le module « Outil pour la programmation » au premier semestre de la troisième année de licence informatique et nous permettait de nous familiariser avec cette façon de travailler qui est de plus en plus présente dans le monde de l'informatique. Notre tuteur était donc project owner et nous étions développeurs. Le fait de se voir régulièrement nous a permis de découper le projet en sprints et de le rendre dynamique. A chaque rendez-vous, on faisait un point sur les tâches réalisées dans le sprint précédent et une fois toutes les fonctionnalités acceptées par M. Dadeau, on en planifiait de nouvelles pour le prochain sprint. Découper le projet nous a permis de travailler régulièrement sur l'application sans jamais la laisser de côté. Chaque développeur avait ses tâches à réaliser et devait corriger les bugs qui étaient rencontrés par les autres développeurs.

Par ailleurs, pour permettre de travailler en agilité tous ensembles, nous avons mis en place des outils pour pouvoir tous développer à distance simplement. Au début du projet, la première tâche réalisée était de créer un dépôt sur GitHub. La configuration de celui-ci était plutôt simple : M. Dadeau était un watcher pour qu'il puisse suivre le projet dès qu'il le

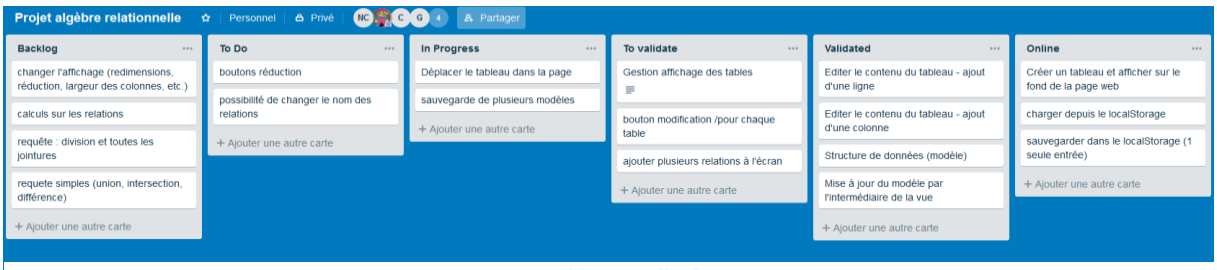

Figure 2 : Tableau Trello du projet

[2] Colonne d'une relation
[3] Ligne d'une relation
[4] Ici, on parle de l'intersection, l'union, la différence et le produit cartésien
[5] Les différentes jointures sont : jointure naturelle, équijointure et téta-jointure

souhaite et ajouter les trois développeurs pour que chacun d'entre nous puisse mettre son travail en commun. Egalement, on a très vite réalisé un système de double branche pour pouvoir mettre en place un site en ligne de l'application développée. Cela permettait de montrer nos avancées et tâches réalisées à chaque rendez-vous avec le project owner. Aussi, à chaque réunion sur le projet, on définissait les tâches à suivre pour le sprint suivant. Sous les conseils de M. Dadeau, nous avons mis en place un tableau de suivi des tâches sur le site Trello. Ce tableau permet de suivre les différentes étapes de développement d'une tâche. Elles peuvent être dans différents états : backlog, to do, in progress, to validate, validated, on line. Chaque tâche est déplacée en fonction de son état dans le projet. L'avantage d'un tel système est de choisir une tâche disponible parmi celles encore non-réalisées, cela permet de ne pas réaliser de travail redondant entre les développeurs et de voir l'avancée des travaux. Egalement, ce tableau organise le travail entre les personnes et permet de savoir sur quoi les autres travaillent, fait important si l'on souhaite donner un peu d'aide à quelqu'un d'autre ou tout simplement travailler à plusieurs sur une tâche.

Ces outils sont primordiaux pour notre avenir dans le monde de l'informatique. Ils sont de plus en plus utilisés dans les équipes de développement pour leurs vertus organisationnelles et dynamiques. Ce projet nous forme donc au travail en équipe avec tous les requis d'un développement d'une application : apprentissage du travail de groupe ainsi que l'apprentissage de la restitution et l'explication des travaux réalisés.

C. Modèle de données

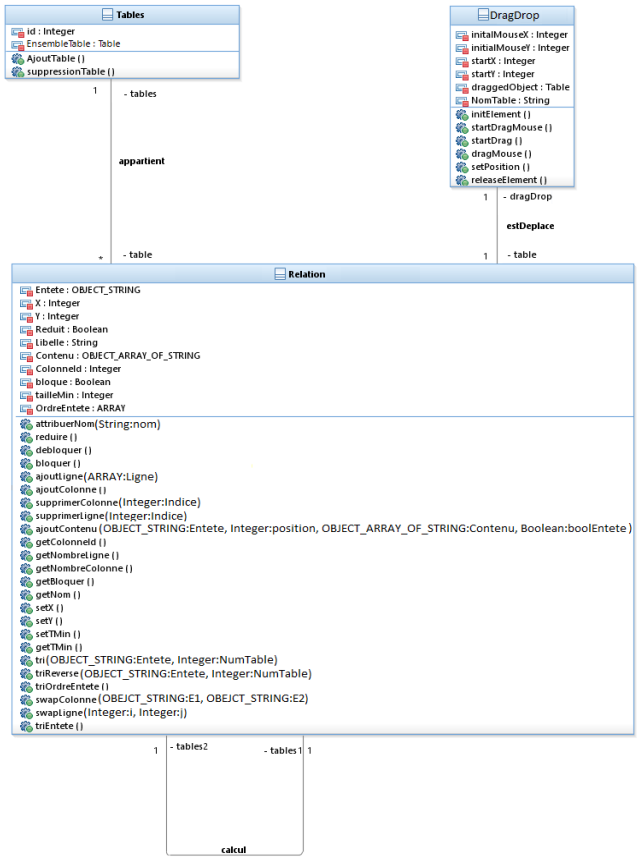

Figure 3 : Diagramme de classe du projet

Au travers de notre cursus universitaire, nous avons à de nombreuses reprises étudié comment stocker des informations. C'est pourquoi, nous proposons ce modèle pour nos relations.

Au fil du projet, notre modèle a beaucoup évolué. La figure 3 illustre le modèle de données de notre application. Tout d'abord, une relation se définit avec un entête et un contenu rempli en fonction de cet entête. Pour modéliser l'entête, nous avons un objet dynamique avec une clé et une valeur pour le premier attribut de l'entête de base pour chaque relation. Cet objet étant dynamique grandit dès que l'utilisateur ajoute une colonne à la relation. Si l'utilisateur ajoute C colonnes à sa relation, l'objet modélisant l'entête aura donc C[6] clés avec valeur. Chaque clé a pour nom « E » concaténé avec l'indice de la colonne. Prenons

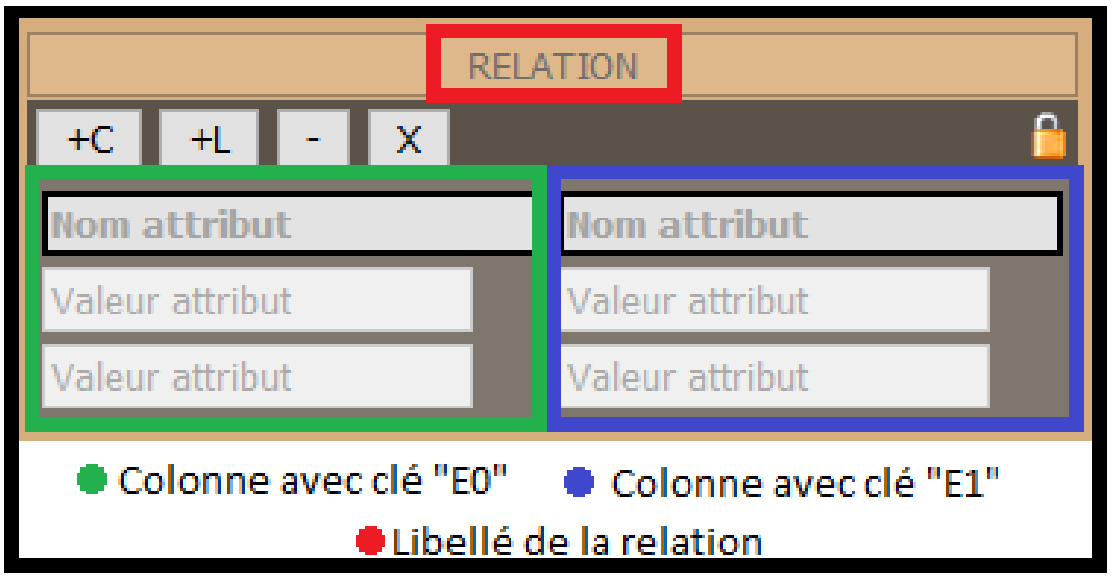

Figure 4 : Schéma d'une relation

un exemple, si la clé d'un attribut est « E5 », celle-ci fait référence à la colonne d'indice 6. Dans cette relation, il y aura six colonnes puisque le premier indice est 0. Le contenu de la relation est identifié avec les mêmes clés dynamiques que l'entête. Si l'entête a pour clé « E0 », le contenu de cet entête est trouvable dans la variable « Contenu » avec la clé du même nom « E0 ». Ici, la valeur sera un tableau composé d'une taille égale au nombre de lignes de la relation. Et, chaque indice de ce tableau sera la valeur du contenu de la ligne. La figure 4 ci-jointe illustre l'exemple suivant : s'il y a deux colonnes et deux lignes dans une relation, la valeur de la case trouvée en colonne deux et ligne deux, se trouvera grâce à la clé « E1 » à l'indice 1 du tableau. Aussi, notre classe pour modéliser une relation possède une position X et Y, avec un libellé, une taille minimum et une variable réduite qui définit l'état[7] de la relation. Sur cette figure la relation est visible. L'ordre de l'entête est un tableau des clés de l'entête. Dès qu'une relation est créée, l'ordre est « E0 », « E1 » jusqu'à « EN ». Cela permet de trier les colonnes par entête pour réaliser certains calculs, ce choix a été fait pour simplifier les opérations suivantes : la différence, l'intersection et l'union. Aussi, différentes fonctions ont été mises en place dans cette classe, pour pouvoir modifier les valeurs de nos variables comme par exemple, ajouter ou supprimer des lignes.

[6] Il y aura bien C clé et non C+1, le premier indice commence à 0 avec clé « E0 »
[7] Les états d'une relation : réduite ou visible

D'autre part, il existe d'autres classes dans notre modèle toutes aussi importantes les unes que les autres. Nous avons une classe qui regroupe l'ensemble des relations. Celle-ci est composée d'une variable qui est le nombre de relations dans notre modèle et d'un tableau de relations. D'ailleurs, c'est ce tableau qui est passé au format JSON et qui est envoyé dans le localstorage[8] pour la sauvegarde du modèle à l'écran. Et nous avons une classe dragAndDrop qui permet de déplacer les relations. Une relation appartient à l'ensemble des relations. Et une relation peut être appelée pour réaliser un calcul avec une autre relation. Dans ce cas, une nouvelle relation est créée donc une relation est ajoutée dans l'ensemble des relations.

La réflexion sur le modèle étant réalisée, nous pouvons passer à la partie développement du projet.

3. Développement de l'application web

Le développement d'une application web est vite chronophage en fonction de ce que l'on souhaite y faire pour améliorer l'ergonomie du site. Nous commencerons par présenter les différentes phases de ce projet, avant d'expliquer les problèmes que l'on a rencontrés ainsi que les solutions apportées.

A. Différentes phases du projet

Un projet d'une telle ampleur prend du temps et pour arriver à un résultat accepté par le client, nous avons dû passer par différentes étapes expliquées dans les paragraphes suivants.

I. Découverte & apprentissage de JavaScript

Dans un premier temps, pour la plupart d'entre nous, le langage utilisé nous était inconnu, on a donc commencé par apprendre la JavaScript. A l'UFR des Sciences et des Techniques de Besançon, ce langage est vu au deuxième semestre de la troisième année de licence informatique. Or, ce projet débute environ à la moitié du premier semestre de cette même année. M. Dadeau, étant l'enseignant réalisant et donnant les cours de JavaScript, il nous a donné en avance accès à ses cours. Les premières semaines ont été de la lecture de cours et de la création de petits algorithmes pour se former à ce nouveau langage. On s'est très vite rendu compte de la puissance de ce langage ; notamment grâce à son pouvoir de tout rendre dynamique et tout modifier. C'est un outil intéressant. Mais on s'est aussi rendu compte de ses faiblesses, comme le fait que le langage est non-typé.

Egalement, à l'Université on apprend beaucoup de choses et l'algèbre relationnelle a été vu en première année de licence informatique, c'est-à-dire il y a deux ans. M. Dadeau réalise, aussi, le cours d'algèbre relationnelle dans le module de « base de données », il nous a donc refourni son cours. Durant ce premier sprint, où l'apprentissage et/ou remise à niveau

[8] Ce mécanisme est expliqué au paragraphe 3A5

est de mise, nous avons relu le cours d'algèbre relationnelle pour se remémorer les spécificités du sujet et bien comprendre comment fonctionnent les opérateurs.

Dans un second temps, nous avons commencé à travailler sur un début de conception du site. On s'est posé toutes les questions que l'on peut voir émerger quand on débute un projet. Nous avons commencé à réfléchir sous quelle forme nous allions gérer nos tables, comment les concevoir dans notre modèle. On s'est aussi demandé à quoi le site allait ressembler, c'est-à-dire comment l'utilisateur devrait utiliser le site et les mécanismes de sûreté à mettre en place pour ne pas qu'il fasse tout ce qu'il veut. Une fois ces réflexions faites, on pouvait enfin commencer à développer des fonctionnalités.

II. Création des relations

Les bases étant posées, on pouvait dans un premier temps commencer par s'occuper de représenter les relations, autant d'un point de vue graphique que dans le modèle.

Tout d'abord, on s'est concentré sur le fait de pouvoir afficher une table graphiquement sur la page web. Nous nous sommes donc contentés d'utiliser les tables avec les balises tr et th du langage HTML5. La première ligne représente l'entête d'une relation, nous avons tout de suite mis en place un style différent du contenu pour que l'entête soit

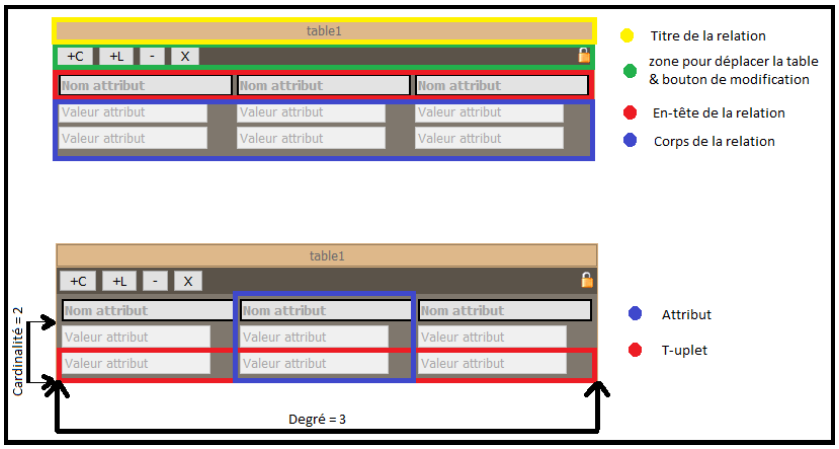

Figure 5 : Différentes sections d'une relation

distingué plus facilement et ajouter de la convivialité. Une fois que l'on pouvait afficher simplement une relation, on a automatisé la création de celles-ci avec le bouton « ajouter tables ». Ce processus étant automatisé, nous devions pouvoir modifier les relations simplement, nous avons ajouté des fonctionnalités d'ajout de lignes et de colonnes.

Ensuite, pouvoir sauvegarder son travail en cours est indispensable. Sous les conseils de M. Dadeau, nous avons utilisé le localstorage des navigateurs web. Cette espace permet de sauvegarder des données sans délai d'expiration. Via une fonction de sauvegarde et de rechargement, nous accédons très facilement à nos relations. Pour sauvegarder, nous avons un objet Tables, qui est composé d'un tableau des tables affichées sur la fenêtre et c'est cet objet Tables qui est passé en chaine de caractères pour pouvoir sauvegarder dans le localstorage. On peut voir grâce au schéma ci-contre comment interagir avec le mécanisme de chargement des modèles. Maintenant, grâce à un système de gestion des données sauvegardées, l'utilisateur peut enregistrer plusieurs modèles sur lesquels il travaillait. Il peut identifier chaque modèle avec un nom différent de ceux déjà enregistrés dans le localstorage.

Le nom de modèle saisi par l'utilisateur est le nom de la clé pour pouvoir sauvegarder les données dans le localstorage, aucune clé ne peut être égale à cause des risques de conflit ; vérifier les données saisies est donc important. Egalement, le nom du modèle n'est pas totalement libre, l'utilisateur n'a pas le droit de mettre de blanc[9] dans le nom, des caractères spéciaux[10] et celui-ci n'a pas le droit de commencer par un chiffre ou un nombre. Le nom doit forcément commencer par une ou plusieurs lettres et peut finir par un nombre. Evidemment, l'utilisateur n'a pas le droit de mettre un nom vide. Aussi, pour pouvoir sauvegarder toutes les relations de notre modèle, on prend toutes les tables avec toutes leurs propriétés et on les passe au format JSON. Cela permet de simplement compacter des objets JavaScript en utilisant la fonction JSON.stringify[11] pour sauvegarder et la fonction JSON.parse[12] pour recharger et réafficher les relations souhaitées. Pour recharger les modèles, l'utilisateur possède un menu. Ce menu est illustré avec la figure 6 qui nous montre les différents boutons

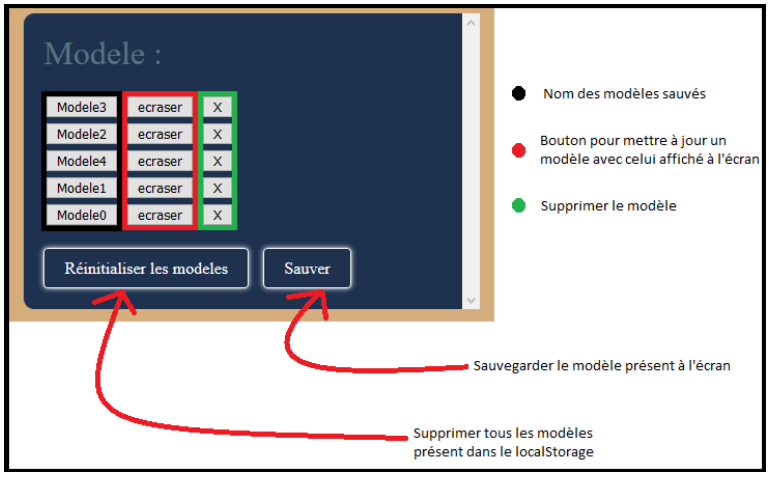

Figure 6 47 : Différentes sections pour sauvegarder et charger un modèle

pour interagir avec le système de sauvegarde. L'encadrement en noir nous permet de voir la liste des modèles sauvegardés. On a ajouté des boutons écraser pour pouvoir mettre à jour un modèle et une croix pour supprimer le modèle. Le nom des modèles est aussi un bouton ; un clic sur ce bouton permet de recharger le modèle voulu. Une fois rechargé, nous n'avons que les objets Table, il faut maintenant les afficher sur la page web. Pour cela, une fonction lit le contenu d'un objet et permet d'afficher les relations à l'écran.

A ce niveau-là, l'utilisateur peut donc créer des relations, en éditant l'entête et le contenu de celle-ci. Il peut également sauvegarder son travail et le recharger pour revenir plus tard et pouvoir poursuivre ses travaux.

III. Opérateurs de calculs simples

L'utilisateur pouvant maintenant réaliser des relations et simuler des situations d'algèbre relationnelle, nous devions commencer à implémenter des opérateurs de calculs simples, notamment les opérateurs de la théorie des ensembles.

[9] Les blancs, en informatique, sont les espaces, retour à la ligne et tabulations
[10] Tout ce qui n'est pas des chiffres ou des lettres
[11] Convertit une valeur JavaScript en chaîne JSON
[12] Analyse une chaine de caractères JSON et construit la valeur JavaScript ou l'objet décrit par cette chaîne

Dans un premier temps, nous avons décidé de commencer par implémenter ceux que l'on jugeait les plus simples. Nous avons donc développé l'union, l'intersection et la différence. Ces opérateurs nous ont parus simples car pour réaliser les calculs de ces opérateurs, nous devons juste comparer les lignes des relations choisies par l'utilisateur. La figure 7 ci-dessous illustre un schéma de code de la fonction union. Pour comparer les lignes, nous avons

```
vérification que les relations ont le mêmes nombres d'attributs
tri sur les entêtes de relations, pour qu'elles aient le même ordre
vérification que les entêtes aient le même ordre en comparant l'attributs des relations

création de la nouvelle entête en parcourant l'entête d'une relation
Ajout de tous les t-uplets de la première relation

Ajout des t-uplets de la seconde relation

POUR ( j=0; j<nbr2; j++)
        ligne2 = recupèreLigne(relation2, j)
        passage au format JSON de ligne2
        estDoublon = faux
        POUR ( i=0; i<nbr1; i++)
                ligne1 = recupèreLigne(relation1, i)
                passage au format JSON de ligne1

                SI (ligne1 == ligne2)
                        estDoublon = vrai
                FIN SI

        FIN POUR
        SI (!estDoublon)
        ALORS ajout de la ligne j de la seconde relation à la table union de ces deux relations
        FIN SI
FIN POUR

Retourne la nouvelle table
```

Figure 7 : Schéma de la fonction union de relations

commencé par réaliser une fonction qui récupère la ligne courante de la relation et retourne celle-ci dans un tableau. Et, nous avons implémenté une double boucle qui parcourt toutes les lignes des deux relations avec une condition qui les compare. Le premier encadrement en rouge montre cette condition. Pour comparer les lignes, nous nous sommes simplifiés la tâche en mettant chaque ligne au format JSON, cette astuce nous donne l'avantage de comparer deux chaines de caractères plutôt que de comparer deux tableaux qui est plus lourd niveau implémentation et complexité. En réalité, les trois fonctions pour créer l'intersection, l'union et la différence entre deux relations sont très similaires à une condition près. Ici, le second encadrement nous permet de voir si la ligne est dans les deux relations, si c'est le cas on ajoute qu'une ligne pour éviter la duplication d'informations. L'intersection prend les lignes en commun donc elle teste l'égalité entre les deux lignes des deux relations. Egalement, l'union prend toutes les lignes des deux relations sans prendre les doublons. Et la différence retire les lignes qui sont en commun dans les deux relations. Ces fonctions sont donc très proches l'une de l'autre. La figure 8, illustre des résultats de requêtes réalisées avec les opérateurs intersection, différence et union.

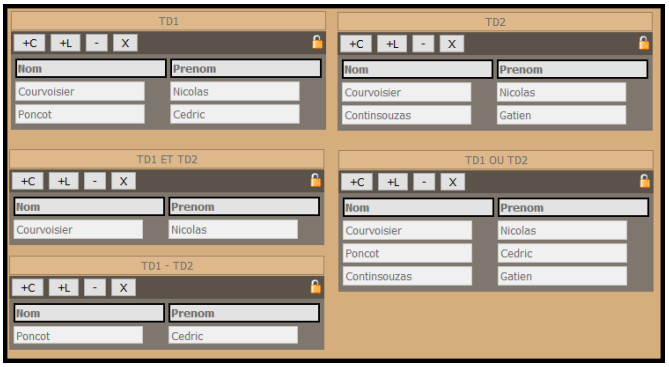

Figure 8 : Résultat des calculs union, intersection et différence

Dans un second temps, il était temps de se concentrer sur les autres opérateurs simples, tels que le produit cartésien, la sélection et la projection. Le produit cartésien en algèbre relationnelle entre deux relations consiste à réaliser toutes les combinaisons possibles entre les t-uplets des deux relations. Pour implémenter ceci, on commence par déterminer la cardinalité[13] des deux relations. Ensuite, on réalise deux boucles imbriquées l'une dans l'autre avec un nombre de tours de la cardinalité des relations respectives et on concatène chaque ligne de la première relation avec toutes celles de la seconde relation. Et, on obtient simplement le produit cartésien de nos deux relations. Cependant, pour réaliser ce produit, il est important de vérifier le contenu des entêtes des deux relations. Une relation doit avoir une entête d'attribut unique. Il se peut que deux relations possèdent des noms d'entête commun entre deux relations. Donc au calcul du produit, il est important de vérifier que les entêtes de la nouvelle relation sont bien uniques. Si ce n'est pas le cas, une fonction supprimera les doublons grâce à un renommage des entêtes concernés. Pour cela, la fonction ajoutera un caractère au début du nom de l'attribut. La figure 9, ci-jointe représente un

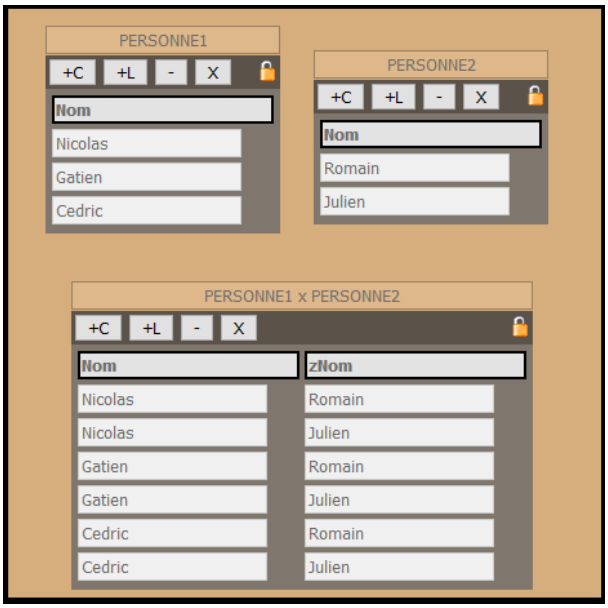

Figure 74 : Résultat d'un produit cartésien

résultat de produit cartésien entre deux relations. Nous pouvons constater que les deux tables possèdent un attribut « nom » en commun. Et à la création de la nouvelle relation, nous avons ajouté le caractère 'z' au début de l'attribut. Ce mécanisme est capital pour éviter tout conflit si l'utilisateur souhaite réaliser d'autres calculs avec cette relation. Par ailleurs, la projection consiste juste à sélectionner les attributs d'une relation choisie par l'utilisateur. On reçoit un tableau contenant les attributs à garder dans la fonction projection et on parcourt les colonnes de la relation pour créer la nouvelle relation et l'afficher.

Par ailleurs, la sélection est une fonction importante pour choisir certaines lignes dans une relation. Cet opérateur doit respecter une syntaxe particulière : attribut, opérateur et valeur. L'attribut vient de la relation R et les opérateurs sont les opérateurs de comparaisons qui sont : =, ≠, >, <, ≤ et ≥. Et les valeurs sont celles du domaine de l'attribut considéré. Si l'attribut représente des nombres, la valeur pour comparer cet attribut doit être aussi un nombre. La figure 10 ci-dessous illustre l'exemple suivant. D'abord, la fonction pour créer la sélection va vérifier que l'attribut saisi est bien présent dans la relation saisie. Et l'entête sera

[13] Nombre de t-uplets d'une relation

créé en copiant celui de la relation. Ensuite, nous devons parcourir les lignes de la relation afin de comparer l'attribut de la ligne avec la valeur saisie. Si la comparaison retourne vraie, la

```
SELECTION :
vérification que l'attribut saisi est présent dans la relation saisie

POUR (i = 0, i < nombreLigneRelation; i++)
        //condition étant la valeur à comparer avec toutes les lignes
        SI (appliqueOperation(attribut, operateur, condition))
        ALORS  récuperer la ligne i
                        et ajout de la ligne dans la nouvelle relation
        FIN SI
FIN POUR

RETOURNER la nouvelle relation


APPLIQUEOPERATION
SWITCH (operateur)
        CAS : "===" : retourne attribut === condition
        CAS : "<" : retourne attribut === condition
        CAS : ">=" : retourne attribut === condition
        //Idem pour les autres cas
        DEFAUT:
        retourne faux
```

Figure 10 : Schéma de code la fonction sélection

ligne va alors être ajoutée dans notre nouvelle relation. Sur le schéma, cette condition est illustrée dans le cadre rouge. Pour finir, elle sera retournée et affichée à l'écran.

IV. Opérateurs plus complexes

Les opérateurs simples implémentés et testés, il est venu le moment de développer les opérateurs que l'on considérait plus complexes. On parle ici de toutes les jointures et de la division. Ces opérateurs sont souvent mal compris des étudiants, avoir un outil qui permet de vérifier les calculs pour les étudiants de licence 1 informatique de Besançon serait intéressant pour eux et leur permettraient de mieux appréhender ces opérateurs.

Tout d'abord, les jointures nous semblaient les plus simples à implémenter. L'un d'entre nous a donc commencé par la jointure naturelle qui est un cas particulier de l'équijointure. Celle-ci est plutôt simple, on commence par vérifier que les deux relations possèdent bien un attribut en commun, si c'est le cas, le calcul peut avoir lieu, sinon, on précise à l'utilisateur une erreur. Pour réaliser cette jointure, on parcourt les colonnes en commun et on compare celles-ci. Si deux valeurs sont égales, on récupère alors les lignes des deux relations et on les concatène dans la nouvelle relation. Puisque deux attributs seront égaux dans la nouvelle relation à cause de l'attribut en commun, on peut se permettre une optimisation, on ne va en garder qu'un seul des deux. Cela permettra d'éviter la duplication d'informations.

Ensuite, nous allons réaliser l'équijointure et la téta-jointure qui sont en réalité très proche l'une de l'autre à une comparaison près. Pour ces deux jointures, l'utilisateur va entrer le nom des deux relations et va saisir les deux attributs avec lesquels il souhaite faire sa jointure. Pour commencer, nous vérifions que les attributs saisis appartiennent bien à leurs relations respectives. Sur le même principe de la jointure naturelle, on imbrique l'une dans l'autre deux boucles qui parcourent les valeurs des attributs choisies pour la jointure

Ensuite, nous allons réaliser l'équijointure et la téta-jointure qui sont en réalité très proche l'une de l'autre à une comparaison près. Pour ces deux jointures, l'utilisateur va entrer le nom des deux relations et va saisir les deux attributs avec lesquels il souhaite faire sa jointure. Pour commencer, nous vérifions que les attributs saisis appartiennent bien à leurs relations respectives. Sur le même principe de la jointure naturelle, on imbrique l'une dans l'autre deux boucles qui parcourent les valeurs des attributs choisies pour la jointure. Si des

```
vérification des attributs présent dans les relations respectives
création de la nouvelle entête en parcourant les deux relations

POUR (i = 0, i < tailleContenuRelation1; i++)
        valeur = Contenu[i]
        POUR (j = 0, j < tailleContenuRelation2; j++)
                SI (valeur == contenu[j])
                ALORS recuperer les deux lignes les concaténés
                        et ajout de la nouvelle ligne à la nouvelle relation
                FIN SI
        FIN POUR
FIN POUR

RETOURNER TABLE
```

Figure 11 : Schéma de code de l'équijointure

valeurs sont égales entre les deux, on récupère la ligne complète des deux relations et on concatène la ligne pour la nouvelle relation. Ici, on ne peut pas se permettre d'optimisation comme la jointure naturelle car les attributs choisis pour la jointure ne seront pas forcément égaux. L'illustration 11, nous montre la comparaison entre les deux cases des deux relations via l'encadre rouge. Si la condition est respectée, nous ajoutons concaténons les deux lignes pour les ajouter dans la nouvelle relation. Comme pour le produit cartésien, ici une vérification sur le nom des attributs est mise en place. Si des attributs sont présents entre les deux

```
vérification des attributs présent dans les relations respectives
création de la nouvelle entête en parcourant les deux relations

POUR (i = 0, i < tailleContenuRelation1; i++)
        valeur = Contenu[i]
        POUR (j = 0, j < tailleContenuRelation2; j++)
                SI (valeur  != contenu[j])
                ALORS recuperer les deux lignes les concaténés
                        et ajout de la nouvelle ligne à la nouvelle relation
                FIN SI
        FIN POUR
FIN POUR

RETOURNER TABLE
```

Figure 1352 : Schéma de code la téta-jointure

relations, l'un des deux sera renommé. Pour la téta-jointure, on réalise exactement le même principe sauf qu'on regarde quand les valeurs sont différentes. Si c'est le cas, on récupère la ligne des deux relations et on les concatène pour la nouvelle relation. L'illustration 12 nous montre le schéma de code de la téta-jointure et insiste sur la condition pour réaliser cette opération.

Ensuite, la division est un opérateur important. Pour effectuer les calculs, l'utilisateur saisit les deux relations avec lesquelles il veut utiliser cet opérateur. On commence par regarder que le nombre de lignes de la relation diviseur n'est pas plus élevé que celui de la

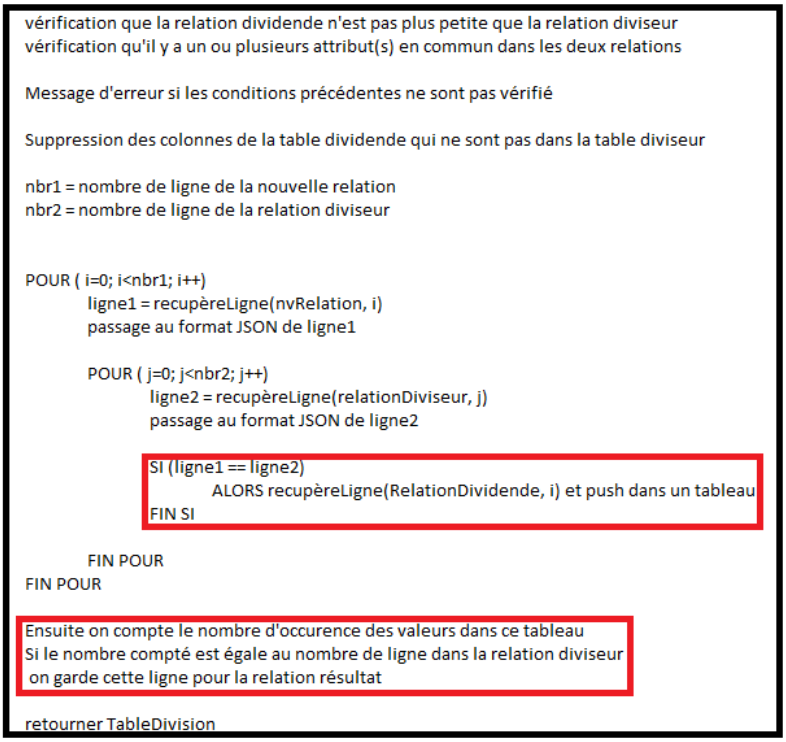

Figure 13 : Schéma de code de la division

relation dividende et on réalise la même opération pour les attributs. Ensuite, l'illustration 13 montre le fonctionnement de la fonction division. On prend les lignes de la relation diviseur, et on les fait parcourir sur les lignes des entêtes en commun entre les deux relations. Si une ligne de la relation dividende est dans la relation diviseur, on l'ajoute dans un tableau ; ce point est illustré avec le premier encadrement rouge. Ensuite, on compte le nombre de lignes de la relation diviseur, et on compte le nombre d'occurrences de chaque valeur dans les attributs de la relation dividende qui ne sont pas dans la relation diviseur. Ce mécanisme a lieu après nos deux boucles, l'illustration 12 nous montre cela avec le deuxième encadrement. Si ces nombres de lignes sont égaux, on garde cette valeur pour la relation résultat.

Les opérateurs plus complexes étant maintenant implémentés, l'utilisateur a tous les outils dans sa main pour réaliser toutes les simulations d'algèbre relationnelle qu'il souhaite.

V. Convivialité & design

Un des éléments les plus importants dans la conception d'un site web est la convivialité de celui-ci. C'est souvent une des premières choses que l'utilisateur remarque avec la simplicité d'utilisation.

a. Ergonomie

Pour qu'un site web soit attractif, il est important que celui-ci soit agréable à naviguer. Nous avons donc axé une partie de notre réflexion sur cet aspect.

Dans le sujet, il était spécifié de rendre l'application simple d'utilisation. Pour cela, diverses fonctionnalités ont été mises en place pour répondre à ce besoin.

-Certains éléments de l'application devaient être déplaçables, on a donc fait le choix de pouvoir bouger chaque relation, l'utilisateur aura donc la possibilité de les déplacer où il le souhaite sur la page. Un simple clic sur la zone prévue à cet effet suivi d'un déplacement de souris réalise cette action.

-Dans la même lignée, il pourra réduire les relations grâce à un bouton réduction. Dans le cas où l'utilisateur possède de grandes relations, il pourra réduire les tables en question. Il restera alors le haut de la relation, c'est-à-dire la partie qui permet de déplacer la relation comportant les boutons de modification et la partie titre de la relation. Ces fonctionnalités donnent à l'utilisateur un gain de place et de la convivialité au site.

-Le titre des relations peut aussi être modifié en ouvrant le cadenas et en double cliquant sur l'espace réservé au nom des relations.

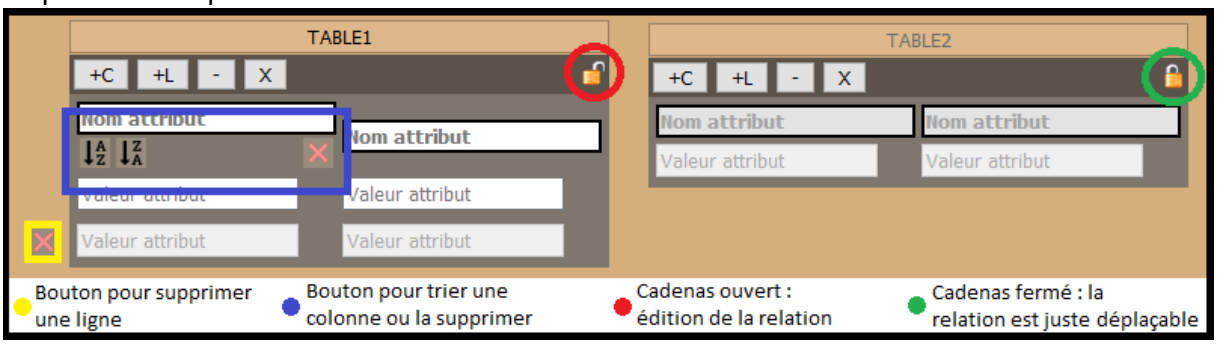

Figure 14 : Schéma pour éditer une relation

-Aussi, dans l'espace permettant de déplacer les relations, nous avons mis en place des boutons de modifications sur les tables. L'utilisateur peut ajouter des lignes et des colonnes. Il peut évidemment supprimer les lignes et les colonnes ; une fois le cadenas de la relation ouvert, au survol de l'entête d'un attribut divers boutons vont s'afficher, dont un qui permettra de supprimer la colonne choisie. Le même principe est mis en place pour les lignes, au survol de celle-ci, une croix rouge s'affichera à gauche de la ligne survolée.

-L'utilisateur, s'il le souhaite, peut trier les lignes des relations en fonction des valeurs de la colonne choisie. Une fois le cadenas ouvert et au survol de l'entête de la colonne souhaité pour trier, il y a deux boutons qui s'affichent : un pour trier de façon ascendante et un pour trier de façon descendante. Ces deux boutons sont présents pour toutes les colonnes, il est donc possible de trier différemment une relation en fonction du contenu d'une colonne.

-Nous avons créé un style CSS très simple, l'important du projet était de réaliser un site fonctionnel de simulation d'algèbre relationnelle. Nous avons donc fait le choix de créer des objets accessibles pour n'importe quel utilisateur. Le cadenas est quelque chose de très visuel ; s'il est activé on peut éditer les tables, sinon on ne peut que les déplacer.

-De plus, sur la droite, il y a une zone pour voir les modèles présents dans le localstorage. L'utilisateur avec un simple clic peut recharger le modèle souhaité ou le supprimer. Il y a aussi un bouton « écraser » qui permet de sauvegarder le modèle courant à la place d'un modèle déjà existant. Ce mécanisme réalise donc une mise à jour d'un modèle. A la sauvegarde d'un modèle, le site propose un nom de modèle. Et on vérifie que celui-ci n'est pas déjà enregistré et que des caractères non autorisés n'ont pas été utilisés.

-Par ailleurs, à divers endroits, nous avons mis des infobulles qui permettent d'expliquer à l'utilisateur comment fonctionnent certaines fonctionnalités. Cela évite de perdre l'utilisateur, lui évite aussi de s'agacer et rajoute de la convivialité à l'application. La figure 15 nous montre les différents exemples d'infobulles sur l'application.

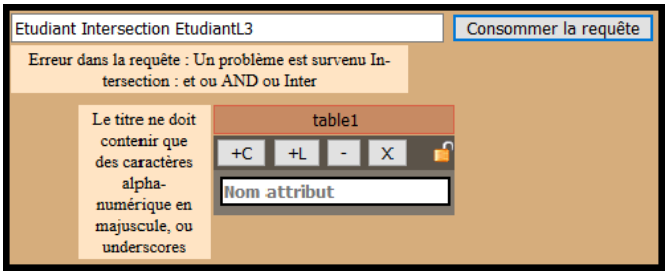

Figure 15 : Exemples d'infobulles

Ces fonctionnalités permettent de simplifier les tâches de l'utilisateur et de simplifier certaines implémentations dans le développement. Aussi, cela apporte de la convivialité et de l'ergonomie à l'application.

b. Analyseur syntaxique

Dans un esprit de convivialité et de simplicité, nous avons mis en place un analyseur syntaxique. L'utilisateur possède un espace réservé pour saisir les requêtes qu'il souhaite réaliser.

Un analyseur syntaxique est quelque chose de très connu dans le monde de l'informatique. Ce mécanisme analyse une structure pour en retenir les informations essentielles. Dans notre cas, la structure à analyser est une chaine de caractères saisie par l'utilisateur. L'analyseur syntaxique va analyser cette structure. Si l'analyse se passe bien, il va réaliser la suite des opérations ; dans notre cas, il va appeler la fonction de l'opérateur identifié pour réaliser le calcul sur deux relations saisies par l'utilisateur et afficher la nouvelle relation. Sinon, il retourne une erreur qui sera affichée via une infobulle à l'utilisateur.

Maintenant, il est temps d'expliquer le fonctionnement de cet analyseur syntaxique. L'utilisateur commence par saisir sa requête et dès qu'il la consomme avec le bouton « consommer la requête », l'analyse va se mettre en place. Pour cela, chaque opération a été définie avec des expressions régulières et on regarde si la requête saisie correspond avec l'une d'entre elles. L'utilisateur doit donc respecter une convention de nommage pour la saisie des requêtes. Chaque élément important doit être espacé d'un espace pour que l'analyseur puisse identifier les éléments importants. Par exemple, pour des opérateurs simples comme la différence ou l'intersection, l'utilisateur devra saisir une requête sous la forme : « TABLE1 et TABLE2 ». Une fois l'analyse terminée, l'application va découper la requête, une partie composée des tables, une autre de l'opérateur et une dernière sur les attributs de relations si l'opération souhaitée nécessite ces informations. Ensuite, toutes ces valeurs sont envoyées à la bonne fonction qui calcule la nouvelle relation.

Un mécanisme de ce type n'est pas toujours simple à implémenter. Mais ce principe rend le site plus propre et plus efficace. Une alternative aurait été de mettre en place un

système de menu défilant en fonction des tables et des opérations, mais ce système perd en convivialité.

VI. Sécurité & contrôle de l'utilisateur

Dans une application, il est important de mettre en place divers mécanismes de sécurité et de vérification pour contrôler les actions des utilisateurs et éviter tout problème.

A chaque calcul, impliquant des relations, nous avons mis en place diverses vérifications pour être sûrs que les informations saisies soient justes. D'abord, pour toutes les relations reçues, on vérifie que les objets soient bien de type Table. Aussi, pour tous les attributs saisis, on regarde que ceux-ci appartiennent bien à la relation en question. Si c'est le cas, on poursuit le calcul, sinon on affiche un message d'erreur et on quitte la fonction. Egalement, nous avons réalisé un analyseur syntaxique[14], pour simplifier les saisies de requête. Aussi, pour pouvoir sauvegarder dans le localstorage, l'utilisateur doit respecter une règle de nommage, reconnue grâce à des expressions régulières.

De plus, pour vérifier et tester le résultat des calculs, nous réalisions au préalable les calculs à la main. Ensuite, nous mettions en place les relations et on faisait le calcul pour vérifier le résultat attendu. Bien évidemment, nous ne nous contentions pas d'un seul test, on essayait pour chaque calcul, des relations différentes avec des tailles différentes et du contenu différent. Par ailleurs, certains calculs peuvent retourner une relation avec juste les attributs en entête et avec un contenu vide. Dans ce cas, on affiche bien cette relation avec un message qui indique que le résultat est normal.

Pour finir, il était d'important de mettre en place quelques vérifications pour contrôler un minimum ce que l'utilisateur fait sur le site pour qu'il ne rentre pas dans des bugs et pour éviter de rentre sa session bugée.

B. Problèmes rencontrés & solutions apportées

Pour parvenir à bout d'un projet comme celui-ci, on passe par différentes étapes et parfois il faut réaliser des choix plus ou moins difficiles qui aideront à la prospérité et la réussite du résultat final.

Au début du développement, nous étions partis sur un modèle très simple, nous avions un objet composé d'un tableau à deux dimensions pour matérialiser les relations et deux variables pour déterminer les tailles de ces tableaux. Cela nous paraissait être le plus simple, nous avions toutes les informations que l'on souhaitait pour stocker les relations dans le localstorage et pour les afficher à l'écran. Mais après réflexion entre nous et M. Dadeau, nous avons convenu de changer ce modèle. A l'heure actuelle[15], une relation est représentée avec un objet pour l'entête et un tableau d'objet pour chaque attribut. Avec ce nouveau modèle, on réduit la redondance d'informations et on stocke toujours aussi bien les informations. Dès

[14] Expliqué au paragraphe 3.A.VI.b
[15] Le modèle actuel est expliqué et détaille au paragraphe 2.C

l'arrivée de ce modèle, nous avons mis en place des getters et des setters pour récupérer ou affecter les informations souhaitées. Cette nouvelle forme, nous a aidé pour les accès aux informations et pour les affecter. Egalement, grâce à ce nouveau modèle, le stockage dans le localstorage était simplifié. Il nous suffisait juste de prendre l'ensemble de toutes les relations de les passer au format JSON et de les stocker. Aussi, au fur et à mesure de l'avancée des travaux, nous rajoutions des variables et des méthodes qui apportaient des modifications aux relations, comme par exemple la réduction ou encore le tri qui sont arrivés vers la fin du projet.

4. Résultats & état du projet

Pour terminer, dans le cahier des charges, le but premier de l'application web consistait à réaliser un site simple et fonctionnel permettant de simuler des situations d'algèbre relationnelle. Le résultat de ce projet est satisfaisant, l'application[16] a été mise en ligne et est accessible par tout le monde. Toutes les fonctionnalités stipulées dans le cahier

Figure 16 : Exemple de calculs et fonctionnalités

des charges ont été réalisées. L'application permet de définir simplement des relations et de saisir leur contenu. Ces relations sont visibles de manière graphique à l'écran en directe. L'utilisateur peut saisir des opérations de l'algèbre relationnelle pour composer des requêtes et ces opérations sont calculées par l'application et créent de nouvelles relations à partir des requêtes écrites. Ces fonctionnalités sont illustrées avec la figure 16 qui nous montre un exemple d'union entre deux relations. Nous pouvons voir les relations pour le calcul, la relation résultat, la requête saisie dans l'analyseur syntaxique et une relation « ENSEIGNANT » qui est réduite. Le site web possède aussi un système de sauvegarde sur l'ensemble des relations pour les reprendre ultérieurement. Le projet a donc abouti et les spécificités attendues sont présentes. Les étudiants de Licence 1 informatique possèdent donc un outil qui leurs permet de vérifier leur calcul dans le cadre de l'algèbre relationnelle.

[16] Lien de l'application dans la bibliographie-netographie

5. Conclusion & perspectives

Pour conclure, l'application web SAR c'est 3 développeurs, 1 tuteur, 385 commits et beaucoup d'heures de travail. L'objectif principal de ce projet était de créer une application web à destination des étudiants de licence 1 informatique de l'UFR des Sciences et des Techniques de Besançon pour que ceux-ci appréhendent plus facilement les opérateurs de l'algèbre relationnelle. Le cahier des charges stipulait que l'on réalise une application où l'utilisateur pouvait créer des relations et exécuter tous les opérateurs de cet algèbre. Nous pouvons dire que ce projet est finalisé. Les fonctionnalités attendues dans le sujet ont été réalisées et le projet a été apprécié par le tuteur.

De plus, ce projet a été formateur pour nous trois. Mettre en place des outils de travail nous a permis de mieux comprendre les attendus dans notre future vie professionnelle. Et réaliser un projet destiné à être utilisé, nous a fait comprendre comment travailler à plusieurs et comment restituer nos tâches développées.

Par ailleurs, le projet pourrait être amélioré pour toucher un plus grand nombre de personnes et pour améliorer la convivialité de l'application. Déjà, on pourrait mettre en place une application mobile de ce site web, l'utilisateur pourrait donc utiliser le site sur tous ses périphériques. Aussi, on pourrait améliorer le style de la page pour rendre le site plus joli et plus convivial.

Bibliographie-Netographie

Application web SAR du développement réalisé.
Auteurs : C. Poncot, G. Continsouzas et N. Courvoisier
Tuteur : F. Dadeau
Mise en ligne : 13 novembre 2018
Mise à jour : 15 mars 2019
Disponible à l'adresse : Application SAR

Cours de M. Dadeau sur la théorie relationnelle.
Auteur : M. Dadeau

Cours de M. Dadeau sur le JavaScript.
Auteur : M. Dadeau

Format JSON (Web).
Auteur : Inconnu
Site : https://wikipedia.org
Mise en ligne : Inconnu
Mise à jour : 16 décembre 2018
Consulté le : 25 février 2019
Disponible à l'adresse : https://fr.wikipedia.org/wiki/JavaScript_Object_Notation

LocalStorage (Web).
Auteurs : Begmans, Bpruneau, Axnyff, EmmanuelBeziat, Nolwenning, goofy_bz et mfrederic
Site : https://developer.mozilla.org/fr
Mise en ligne : Inconnu
Mise à jour : 25 juillet 2018
Consulté le : 25 février 2019
Disponible à l'adresse : https://developer.mozilla.org/fr/docs/Web/API/Window/localStorage

Résumé

Ce rapport présente le travail réalisé dans le cadre du module de projet tutoré en troisième année de licence informatique de l'UFR-ST. Celui-ci s'est déroulé en trinôme du mois de novembre 2018 au mois de mars 2019.

L'objectif de ce projet était de mettre en place une application web qui permettra aux étudiants de Licence 1 d'appréhender plus facilement le fonctionnement des opérateurs de la théorie relationnelle. Cette application, principalement développé en JavaScript, permet donc de simuler des situations d'algèbre relationnelle. L'utilisateur peut créer des relations sur son écran et réaliser toute une série d'opération comme la différence, l'union et l'intersection ou même des opérations plus complexes comme des jointures et la division.

Mots clés : JavaScript, Algèbre relationnelle, Théorie relationnelle

Abstract

This document reports the project made during the module of supervised project as part of the bachelor degree's last year. This project was done in trinomial from November 2018 to March 2019.

Goal of this project was to set up a web application who allow first-class to understand more easily how operators of relational theory works. This application, principally developed in JavaScript, permit to simulate relational algebra situations. The user can create relations on his screen and realized some operations like difference, union or intersection, also more complicated operator like joins or division.

Keywords : JavaScript, Relational algebra, Relational theory