

Travail de Bachelor

Slyum Relationnel

Non confidentiel



Étudiant :	Yoann Rohrbasser
Enseignant responsable :	Pier Donini
Année académique :	2019-2020

Yverdon-les-Bains, le 31 juillet 2018

Département TIC

Filière Informatique

Orientation Logiciel

Étudiant Yoann Rohrbasser

Enseignant responsable Pier Donini

Travail de Bachelor 2019-2020

Slyum Relationnel

Résumé publiable

Lors d'un précédent travail de bachelor, un éditeur de diagrammes de classes UML a été développé, SLYUM. Ce logiciel, écrit en Java, permet la définition graphique de diagrammes de classes UML 1.4.

Pour la première étape de ce projet j'ai adapté SLYUM pour pouvoir créer des schémas relationnels. Les schémas comportent trois composants : Des tables, des vues et les liens entre ces éléments.

Pour la deuxième étape, j'ai ajouté la possibilité de générer un schéma relationnel à partir d'un diagramme UML de la version précédente de SLYUM.

Pour la dernière étape, j'ai ajouté la possibilité de générer du code SQL pour les SGBD MySQL et PostgreSQL à partir d'un schéma relationnel de SLYUM et de le sauvegarder dans un fichier séparé.

Étudiant :

Rohrbasser Yoann

Date et lieu :

.....

Signature :

.....

Enseignant responsable :

Donini Pier

Date et lieu :

.....

Signature :

.....

Préambule

Ce travail de Bachelor (ci-après TB) est réalisé en fin de cursus d'études, en vue de l'obtention du titre de Bachelor of Science HES-SO en **Ingénierie / Economie d'entreprise**.

En tant que travail académique, son contenu, sans préjuger de sa valeur, n'engage ni la responsabilité de l'auteur, ni celles du jury du travail de Bachelor et de l'Ecole.

Toute utilisation, même partielle, de ce TB doit être faite dans le respect du droit d'auteur.

HEIG-VD

Le Chef du Département

Yverdon-les-Bains, le 31 juillet 2020

Authentication

Le soussigné, Yoann Rohrbasser, atteste par la présente avoir réalisé seul ce travail et n'avoir utilisé aucune autre source que celles expressément mentionnées.

Aigle, le 31 juillet 2020

Yoann Rohrbasser

Table des matières

Contents

1	SLYUM	8
1.1	Unified Modeling Language (UML)	8
1.2	Modèle relationnel	8
1.3	Schéma relationnel	8
2	Cahier des Charges	9
2.1	Résumé du projet Donn�� par l'enseignant	9
2.2	Etapas du projet	9
2.3	Cr��ation et visualisation du sch��ma relationnel	9
2.3.1	Fonctionnalit��s	9
2.3.2	Etapas	9
2.4	Conversion du sch��ma UML en relationnel	9
2.4.1	Fonctionnalit��s	9
2.4.2	Etapas	10
2.5	Conversion du sch��ma relationnel en SQL	10
2.5.1	Fonctionnalit��s	10
2.5.2	SGBD vis��es	10
2.5.3	Etapas	10
2.6	Liaison entre relationnel et UML	10
2.6.1	Fonctionnalit��s	10
3	Mod��lisation	11
3.1	Meta-Sch��ma Relationnel	11
3.1.1	Sch��ma	11
3.1.2	Table	11
3.1.3	Cl��s Primaires/Etrang��res/Alternatives	11
3.1.4	Attributs/PrimitiveType	11
3.1.5	Proc��dures Stock��es/Vues/Triggers	11
3.1.6	Proc��dures	11
3.2	Diagramme de classe – entit��	12
3.2.1	RelationalEntity	12
3.2.2	RelationalAttribute	12
3.2.3	Key	12
3.2.4	Trigger, TriggerTypes et ActivationTime	12
3.2.5	Proc��dure	13

3.2.6	StoredProcedure.....	13
3.2.7	View	13
3.3	Diagramme de classe - relation	13
4	Règles de conversion	15
4.1	Conversions basiques	15
4.2	Conversions des cardinalités	15
4.2.1	Conversion des associations binaires 1 : 1	15
4.2.2	Conversion des associations binaires 1 : 0..1	15
4.2.3	Conversion des associations binaires 0..1 : 0..1	15
4.2.4	Conversion des associations binaires 1 : N.....	15
4.2.5	Conversion des associations binaires 0..1 : N.....	15
4.2.6	Conversion des associations binaires N : M	15
4.3	Conversion des associations multiples	16
4.4	Conversion des liens d'héritage.....	16
5	Validation Schéma	17
6	Mockup Interface.....	17
7	Structure générale de Slyum	19
7.1	Librairie graphique Swing	20
8	Création et visualisation du schéma relationnel	20
8.1	Nouveaux éléments.....	21
8.1.1	Tables.....	21
8.1.2	Vues	21
8.1.3	Relations	21
8.2	Procédures des vues et des triggers	21
8.3	Vues graphiques	22
8.3.1	Vue du diagramme	22
8.3.2	Vue hiérarchique	23
8.3.3	Sauvegarde	23
8.3.4	Vue des propriétés.....	24
8.3.5	Propriétés clés alternatives	25
8.3.6	Procédures stockées.....	25
8.3.7	Refactoring.....	25
8.3.8	Valideur	26
8.3.9	Règles de validation.....	27
8.4	Améliorations possibles.....	27
9	Conversion du diagramme UML en relationnel.....	28
9.1	Implémentation	28

9.2	Exemple de conversion.....	28
10	Conversion du schéma relationnel en scripte SQL	30
10.1	Implémentation	30
11	Conclusion.....	31
11.1	Problèmes connus	31
11.2	Améliorations possibles.....	31
11.3	Slyum 2.0	32
11.4	Conclusion personnelle	32
12	Annexes.....	33
12.1	Journal de travail	33
12.1.1	Semaine 1 – 17 au 23 Février	33
12.1.2	Semaine 2 – 24 Février au 1 Mars	33
12.1.3	Semaine 3 – 2 au 8 Mars.....	33
12.1.4	Semaine 6 – 23 au 29 mars.....	33
12.1.5	Semaine 7 – 30 Mars au 5 Avril	33
12.1.6	Semaine 8 – 6 au 12 Avril	33
12.1.7	Semaine 9 – 13 Au 19 Avril	33
12.1.8	Semaine 10 – 20 au 26 Avril	33
12.1.9	Semaine 11 – 27 Avril au 3 Mai	34
12.1.10	Semaine 12 - 4 au 10 Mai	34
12.1.11	Semaine 13 – 11 au 17 Mai.....	34
12.1.12	Semaine 14 – 18 au 24 Mai.....	34
12.1.13	Semaine 15 – 25 au 31 Mai.....	34
12.1.14	Semaine 16 – 1 au 7 Juin	34
12.1.15	Semaine 17 – 8 au 14 Juin	34
12.1.16	Semaine 18 – 15 au 21 Juin	34
12.1.17	Semaine 19 – 22 au 28 Juin	34
12.1.18	Semaine 20 – 29 Juin au 5 Juillet	34
12.1.19	Semaine 21 – 6 au 12 Juillet	34
12.1.20	Semaine 22 – 13 au 19 Juillet	35
12.1.21	Semaine 23 – 20 au 26 Juillet	35
12.1.22	Semaine 24 – 27 au 31 Juillet	35

Bibliographie

https://fr.wikipedia.org/wiki/Mod%C3%A8le_relationnel

<https://www.javatpoint.com/java-swing>

<https://www.postgresql.org/docs/>

<https://github.com/HEIG-GAPS/slyum>

<https://www.mysql.com/products/workbench/>

<https://www.mysqltutorial.org>

<https://www.postgresqltutorial.com/>

<https://stackoverflow.com/>

<https://www.geeksforgeeks.org/detect-cycle-in-a-graph/>

[https://en.wikipedia.org/wiki/Swing_\(Java\)](https://en.wikipedia.org/wiki/Swing_(Java))

<https://github.com/yoann0000/slyum> (repo Github du projet)

1 SLYUM

“ Slyum est un logiciel de construction de diagrammes de classe UML. Slyum rend la création de ces diagrammes simple et intuitif avec une interface intuitive, propre et facile à utiliser.

Beaucoup d'autres éditeurs de diagrammes existent dans les mondes industriels et open source. Mais beaucoup d'entre eux sont compliqué et ne sont pas plaisant à regarder. De plus ajouter des nouveaux éléments est difficile pour l'utilisateur. C'est pour ces raisons que nous avons développé ce projet. Cette application sera simple d'utilisation et seulement les éléments utiles seront intégrés. Le but de ce projet est qu'il puisse être utilisé pour l'apprentissage de l'UML. ”

Traduction non-officiel du préambule de la page GitHub de Slyum

1.1 Unified Modeling Language (UML)

UML permet, en génie logiciel, de concevoir la structure d'un projet orienté objet sous la forme d'un diagramme.

Vu que UML n'est pas le sujet principal de ce projet je ne vais pas élaborer plus dessus mais une définition plus complète est disponible dans le rapport de l'ancien travail de bachelor Slyum à l'adresse suivante : https://github.com/HEIG-GAPS/slyum/blob/master/doc/Slyum_Rapport.pdf

1.2 Modèle relationnel

“ Le modèle relationnel est une manière de modéliser les relations existantes entre plusieurs informations, et de les ordonner entre elles. Cette modélisation est souvent retranscrite physiquement (« implémentée ») dans une base de données.

On appelle « relation » un ensemble d'attributs qui caractérisent une proposition ou une combinaison de propositions comme "un employé a un matricule, il a un nom, il a un employeur". Dans cet exemple, les attributs de l'employé sont : son matricule, son nom et son employeur. Chaque combinaison de propositions ainsi formée est appelée uplet ou collection ordonnée d'objets. Par exemple l'ensemble ("1245", "Jean Dupond", "Compagnie des belles lettres") constitue un uplet de relation "employé". Les relations sont d'ordinaire représentées sous la forme de tables. Dans l'exemple précédent, la table serait libellée "employé". Usuellement, les praticiens accordent la même signification aux concepts de "relation" et de "table". De même, ils assimilent d'une part la "ligne dans la table" et l'uplet, et d'autre part le "libellé de colonne dans la table" et l'attribut. Par définition, chaque uplet d'une relation est unique. Il est identifié par un attribut (un identifiant unique appelé "clef primaire") ou une combinaison de plusieurs attributs qui forme la clef. L'ordre des uplets n'est pas significatif.

Le modèle relationnel est aujourd'hui l'un des modèles les plus utilisés. Le modèle relationnel est basé sur deux instruments puissants : l'algèbre relationnelle (c'est-à-dire le concept mathématique de relation en théorie des ensembles) et la notion de produit cartésien. Ce modèle définit une façon de représenter les données, les opérations qui peuvent être effectuées ainsi que les mécanismes pour préserver la consistance des données. ”

Abréviation de la définition du modèle relationnel selon [wikipedia.org](https://fr.wikipedia.org/wiki/Mod%C3%A8le_relationnel)

1.3 Schéma relationnel

Un schéma relationnel est une représentation visuelle du modèle relationnel similaire au schéma UML qui est utilisé pour conceptualiser une base de données relationnelle.

2 Cahier des Charges

2.1 Résumé du projet Donné par l'enseignant

Lors d'un précédent travail de bachelor, un éditeur de diagrammes de classes UML a été développé, SLYUM. Ce logiciel, écrit en Java, permet la définition graphique de diagrammes de classes UML 1.4.

Objectifs

Le travail consistera à ajouter les fonctionnalités suivantes à l'application Slyum :

Permettre la traduction d'un schéma UML en schéma relationnel.

Permettre l'édition graphique de schémas relationnels.

Permettre la traduction d'un schéma relationnel en UML.

Permettre la génération de code SQL pour différents SGBD (PostgreSQL, MySQL)

Permettre l'évolution du schéma exprimé dans un langage (UML ou relationnel) tout en maintenant la cohérence de sa traduction (relationnel ou UML) et sans perte de ses modifications (p.ex. placement graphique d'une table).

2.2 Etapes du projet

1. La création et visualisation du schéma relationnel.
2. La conversion du schéma UML en relationnel.
3. La conversion du schéma relationnel en SQL.
4. L'établissement du lien entre UML et relationnel. (Si temps le permet)

2.3 Création et visualisation du schéma relationnel

2.3.1 Fonctionnalités

- Créer des tables et définir une clé primaire.
- Définir les attributs de la table.
- Création d'une clé primaire, choix d'une clé, composite ou non parmi les attributs existants.
- Création des relations entre les tables et ajout de la clé étrangère.
- Création de clés alternatives.
- Validation du schéma relationnel et indication les erreurs.
- L'éditeur de schéma relationnel gardera toutes les fonctionnalités pertinentes de l'éditeur de schéma UML.

2.3.2 Etapes

1. Pouvoir sélectionner si on crée un UML ou un REL quand on crée un nouveau schéma
2. Désactiver la UI UML-only et activer la REL-only (même si elle ne fait rien pour le moment)
3. Vérifier que le placement des éléments visuels fonctionne toujours et les réparer sinon
4. Implémenter les classes des éléments du REL
5. Ajouter une UI pour définir/créer la clé primaire de la classe
6. Ajouter une UI pour afficher les clés étrangères
7. Implémenter le validateur REL et les éléments de UI associés

2.4 Conversion du schéma UML en relationnel

2.4.1 Fonctionnalités

- Générer un schéma relationnel par rapport au schéma UML affiché.
- Les informations nécessaires à la complétion du schéma relationnel sont demandées à l'utilisateur via un formulaire à compléter.

2.4.2 Etapes

1. Créer l'algorithme de conversion
2. Ajouter la UI pour entrer les informations complémentaires

2.5 Conversion du schéma relationnel en SQL

2.5.1 Fonctionnalités

- Convertir le schéma relationnel en un script SQL parmi les options de SGBD implémentées.
- Si des options/informations supplémentaires sont nécessaires à la conversion elles seront demandées à la création.
- Le script SQL ne sera pas lié au schéma mais sera créé et exporté du coup tout changement du modèle demande la régénération du script.
- Avant la génération du script, vérifier que le schéma relationnel soit correct.

2.5.2 SGBD visées

- MySQL
- PostgreSQL
- SQLite

2.5.3 Etapes

1. Créer l'algorithme de conversion
2. Ajouter une UI pour choisir la SGBD visé

2.6 Liaison entre relationnel et UML

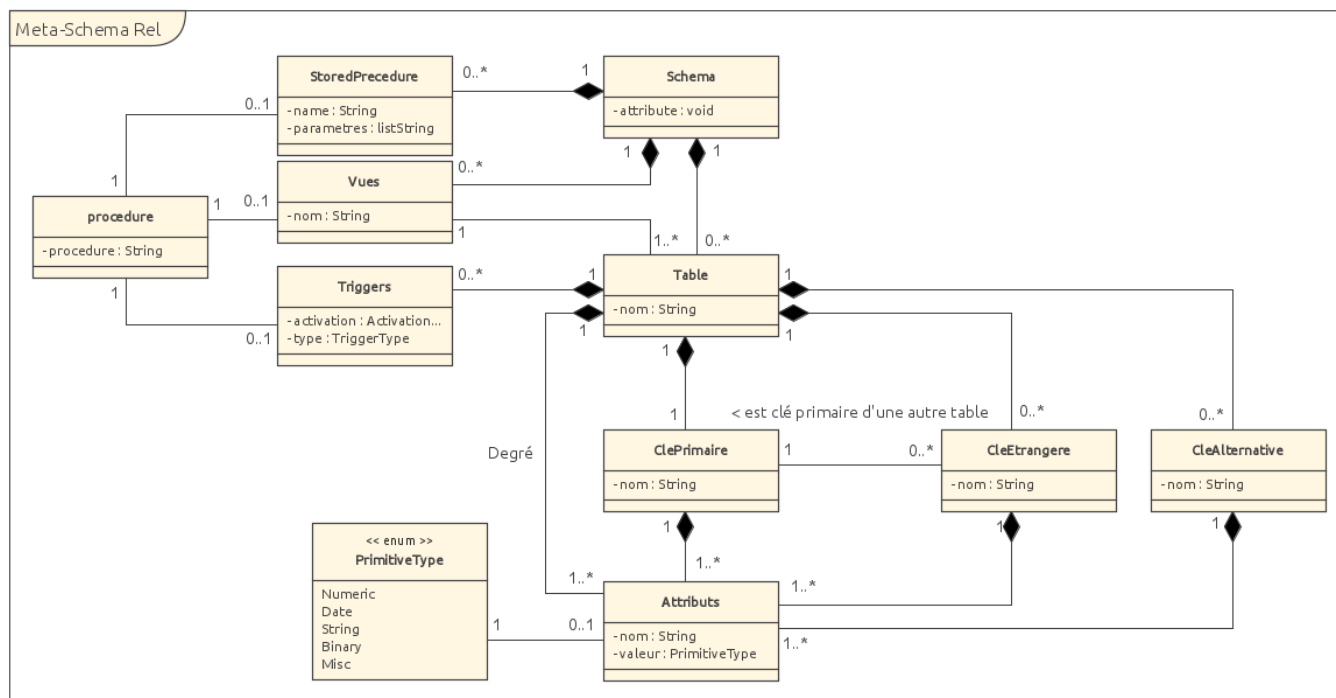
2.6.1 Fonctionnalités

- Quand un schéma UML ou relationnel est généré par rapport à un autre schéma celui-ci est rajouté dans le fichier de sauvegarde.
- Tout changement dans un des diagrammes (ajout/suppression/modification de classes/liens) devra se répercuter dans l'autre schéma.
- Si un changement dans le schéma UML casse le schéma relationnel (ex : suppression de la clé primaire) l'utilisateur est averti avant et doit confirmer d'effectuer le changement.

3 Modélisation

3.1 Meta-Schéma Relationnel

Le meta-schéma sert à représenter les éléments du schéma relationnel qui seront présent dans Slyum Relationnel.



3.1.1 Schéma

Le schéma est une classe englobante qui sert à différencier les schémas relationnels au sein du logiciel.

3.1.2 Table

La table est l'élément basique d'une base de données relationnelle.

3.1.3 Clés Primaires/Etrangères/Alternatives

Les différentes clés des tables doivent être uniques et sont composées d'un ou plusieurs attributs.

3.1.4 Attributs/PrimitiveType

Les attributs sont les différentes colonnes de la table et PrimitiveType leur type de donnée.

3.1.5 Procédures Stockées/Vues/Triggers

Ces éléments ne font pas partie du modèle relationnel strict mais sont présent dans beaucoup de SGBD et sont intéressant à inclure pour la génération du script SQL.

3.1.6 Procédures

Les procédures sont une représentation textuelle de code exécutable par une SGBD.

3.2 Diagramme de classe – entité

Ce diagramme contient tous les éléments du meta-schéma sauf la relation entre clé primaire et clé étrangère. Les classes en beige existent déjà et celles en rouge sont celles que je vais rajouter.

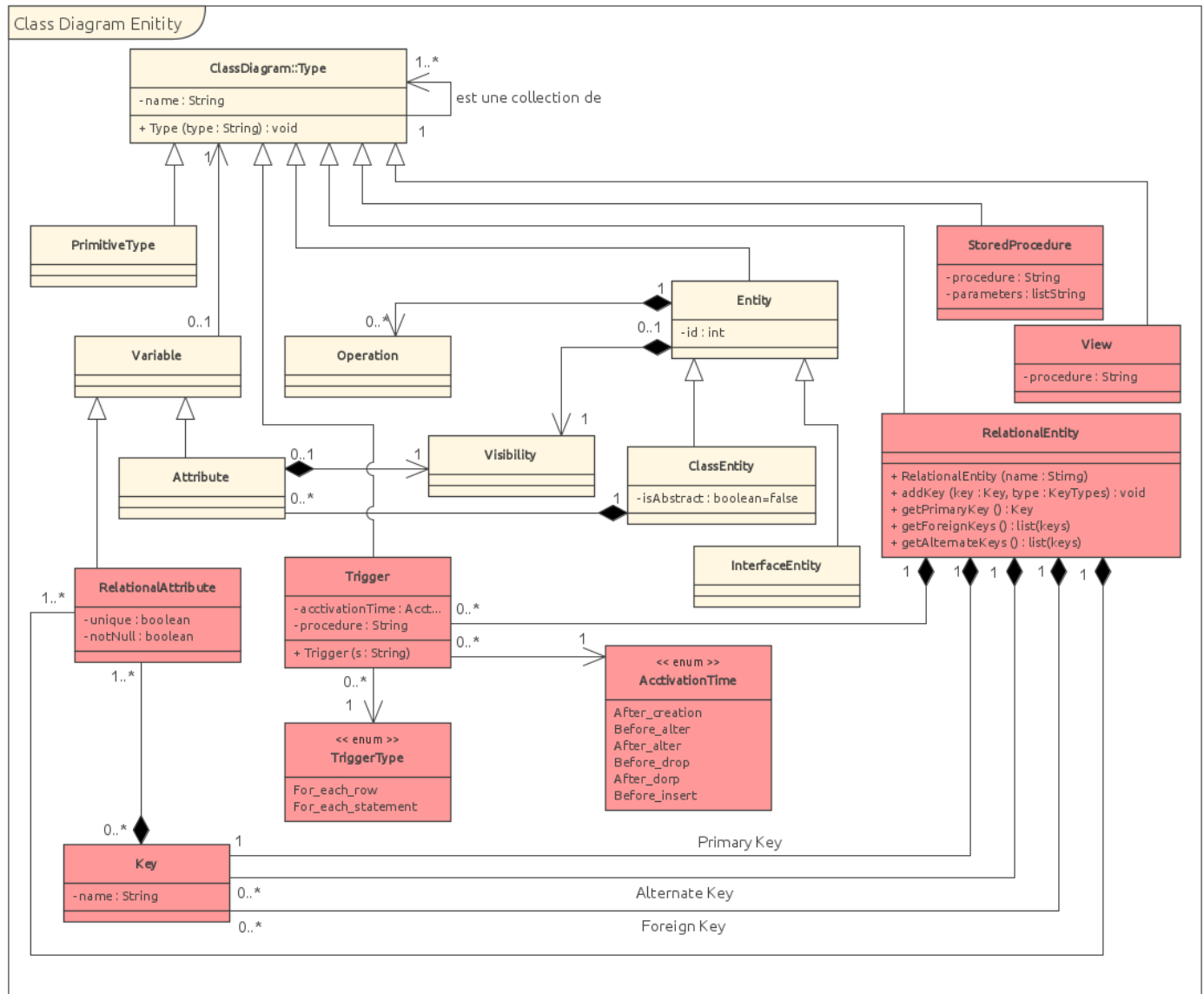


Diagramme de classe proposé au moment de la conception

3.2.1 RelationalEntity

RelationalEntity est l'objet correspondant à la table du meta-schéma.

3.2.2 RelationalAttribute

RelationalAttribute est l'objet correspondant à l'attribut du meta-schéma.

3.2.3 Key

Key correspond au 3 types de clés du meta-schéma qui seront stockées dans RelationalEntity.

3.2.4 Trigger, TriggerTypes et ActivationTime

Trigger correspond au Trigger du meta-schéma. TriggerTypes et ActivationTime sont des propriétés du trigger.

3.2.5 Procédure

L'élément procédure du meta-schéma est représenté par l'attribut "procédure" dans les classes Trigger, StoredProcedure et View.

3.2.6 StoredProcedure

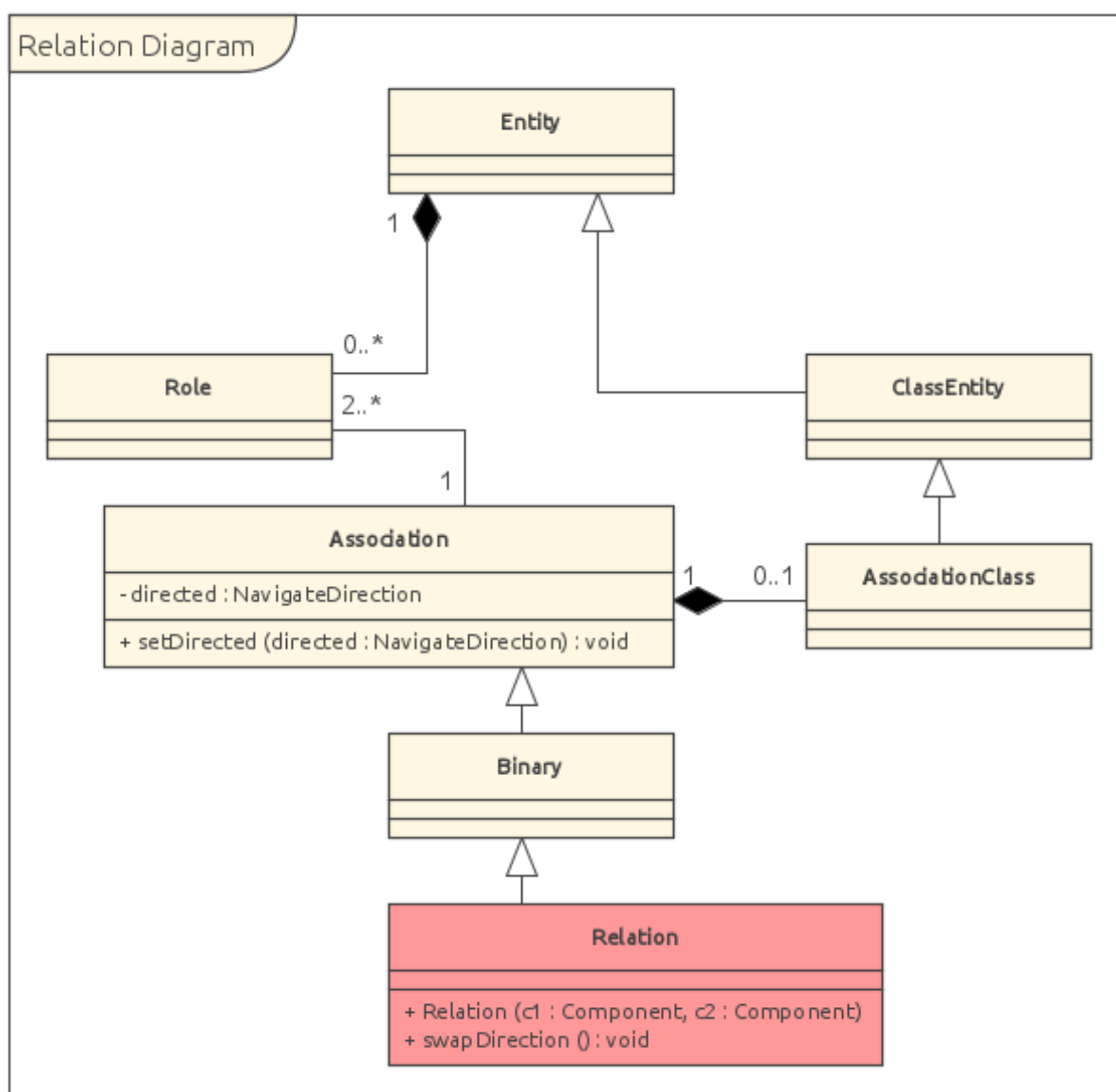
StoredProcedure correspondent à procédure stockée du meta-schéma.

3.2.7 View

View correspond aux vues du méta-schéma.

3.3 Diagramme de classe - relation

Ce diagramme montre comment la relation entre les tables est défini.



La classe relation est le seul moyen de créer des clés étrangères dans les tables. Quand une relation est créée, la table d'origine du lien est définie comme clé primaire et la table de destination comme clé étrangère. La clé primaire de la table de destination est ajoutée à la table d'origine comme clé étrangère. La méthode `swapDirrection` permet de changer la "direction" du lien et ainsi changer quelle table recevra la clé étrangère.

4 Règles de conversion

4.1 Conversions basiques

- Les classes, interfaces et classes d'association deviennent des **Tables**.
- Les attributs deviennent des **Attributs Relationnels**
- **Les opérations sont négligées à la conversion car elles n'ont pas d'équivalence directe en relationnel**
- Associations simples, agrégations et compositions deviennent des **Relations**.

Une clé par défaut appelé "ID" est ajoutée aux tables converties et doit être peuplée manuellement par l'utilisateur.

4.2 Conversions des cardinalités

4.2.1 Conversion des associations binaires 1 : 1

- On rajoute la clé étrangère dans la table du premier coté (cela n'a pas vraiment d'importance)
- La clé doit être **unique et non null**.

4.2.2 Conversion des associations binaires 1 : 0..1

La conversion est similaire à la 1:1 à l'exception que la clé étrangère doit être dans la table qui est de cardinalité 1.

4.2.3 Conversion des associations binaires 0..1 : 0..1

La conversion est similaire à la 1:1 à l'exception que la clé est unique est **nullable**.

4.2.4 Conversion des associations binaires 1 : N

- On rajoute la clé étrangère dans la table qui a **1** comme cardinalité.
- La clé doit être **non null**.

4.2.5 Conversion des associations binaires 0..1 : N

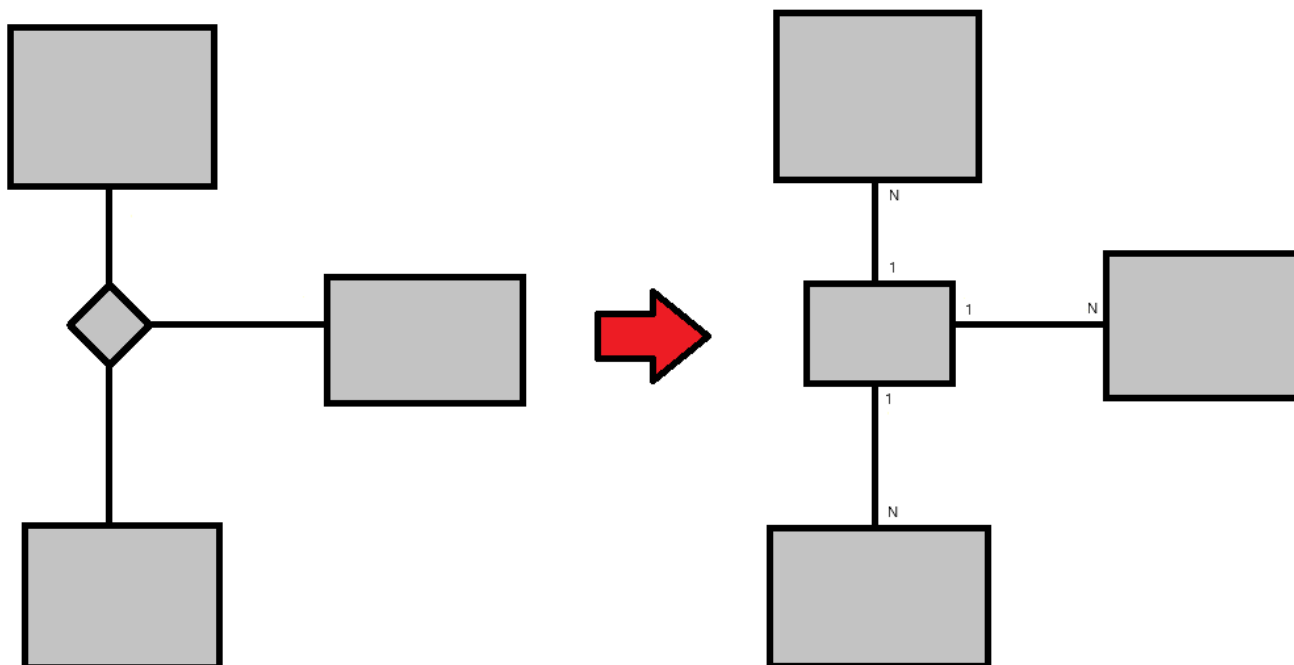
La conversion est similaire à 1:N à l'exception que la clé est **nullable**.

4.2.6 Conversion des associations binaires N : M

La conversion de d'une association N:M se fait en créant une table entre les 2 deux. Cette table prendra les clés étrangères des 2 tables de l'association et utilisera leurs composants comme clé primaire.

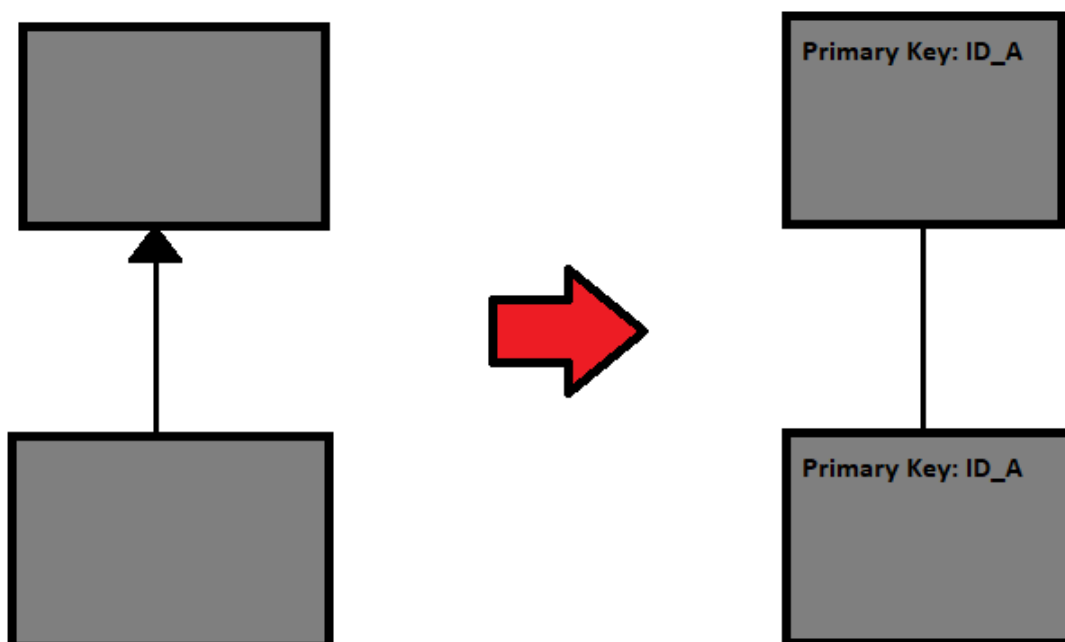
4.3 Conversion des associations multiples

Une association multiple est convertie en une table liée aux autres tables de la relation via des relations 1-N avec le N du côté des tables.



4.4 Conversion des liens d'héritage

Les liens d'**héritage** sont convertis en **relation** avec la sous-classe ayant la même clé primaire que la classe parent.



5 Validation Schéma

Un validateur du schéma relationnel pourra être lancé manuellement et sera lancé automatiquement avant la conversion en SQL. La validation peut aussi être lancée manuellement via l'interface. Les résultats de cette analyse peuvent se trouver sur l'interface de projet (voir plus bas).

6 Mockup Interface

Mockup interface tables, attributs et clés primaire

Table

Primary key

id

Attribute	Type	Visibility	NOT NULL	UNIQUE
id	int	Private	<input type="checkbox"/>	<input checked="" type="checkbox"/>
attribute_1	int	Private	<input type="checkbox"/>	<input type="checkbox"/>
attribute_2	bool	Private	<input type="checkbox"/>	<input type="checkbox"/>
attribute_3	string	Private	<input type="checkbox"/>	<input type="checkbox"/>

Mockup interface triggers

Trigger	Type	Activation	Abstract	Static
trigger	for_each_row	after_creation	<input type="checkbox"/>	<input type="checkbox"/>

Procedure

//procedure text here

Mockup interface clés alternatives

Clé Alternative

nom_clé_1

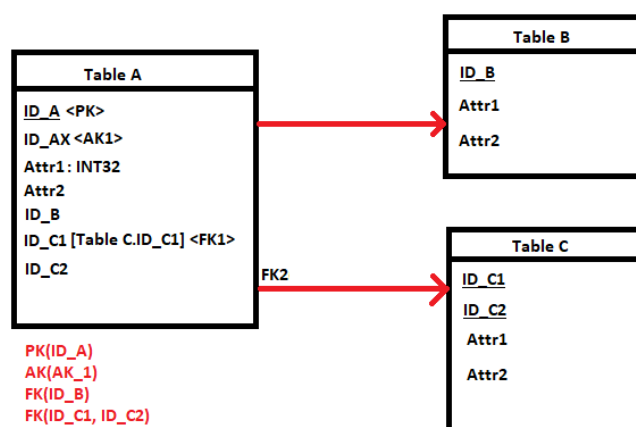
nom_clé_2

Attributs

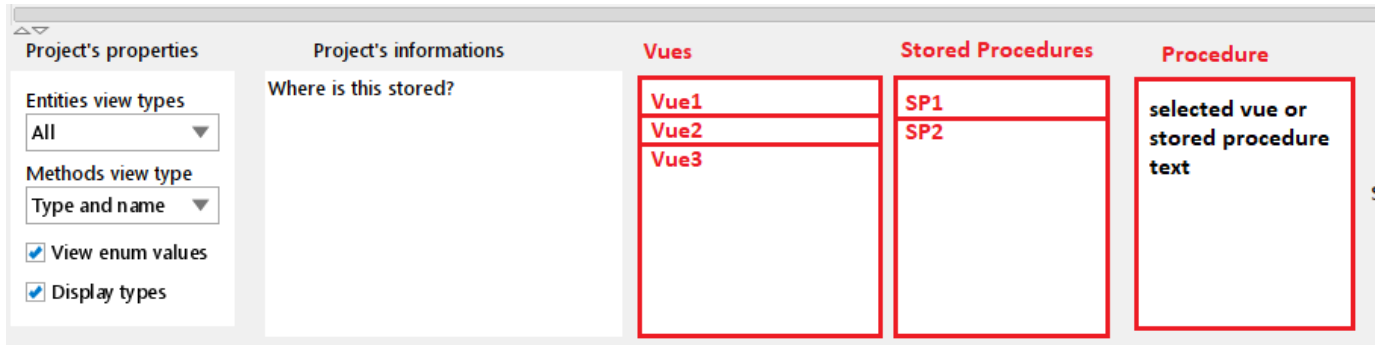
attribut_1

attribut_2

Mockup représentations des clés et attributs



Mockup interface de projet (vue et procédures stockées).



The mockup interface is divided into several sections:

- Project's properties:** Includes dropdowns for 'Entities view types' (set to 'All') and 'Methods view type' (set to 'Type and name'). It also has checkboxes for 'View enum values' and 'Display types', both of which are checked.
- Project's informations:** A section titled 'Where is this stored?'.
- Vues:** A list containing 'Vue1', 'Vue2', and 'Vue3'.
- Stored Procedures:** A list containing 'SP1' and 'SP2'.
- Procedure:** A text area with the placeholder text 'selected vue or stored procedure text'.

La partie de l'interface dédiée aux éléments de la classe est adapté pour fonctionner avec les tables relationnelles.

Une table pour définir une clé primaire est rajoutée. Cette table peut contenir un ou plusieurs attributs.

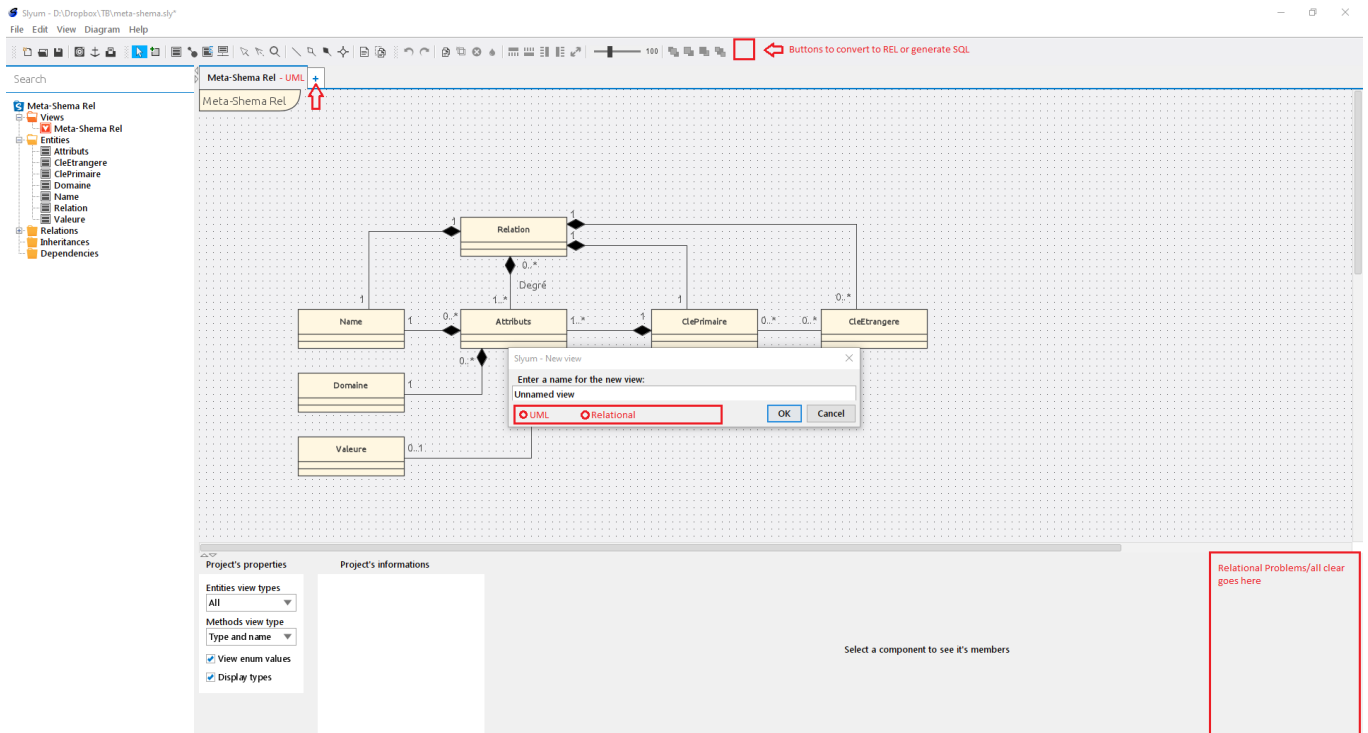
La partie contenant les méthodes de classes est changée pour pouvoir stocker des triggers. La partie montrant les paramètres de la méthode montre maintenant une zone de texte pour la procédure.

Une troisième partie est rajoutée pour définir les clés alternatives et ses attributs.

Si aucune table n'est sélectionnée, on voit l'interface du projet avec les vues et les procédures stockées à côté.

Les procédures stockées n'ayant pas été implémentée j'ai déplacé l'interface des vues dans sa propre partie.

Mockup interface création de vue "uml" ou "relationnel"



The mockup interface shows a UML diagram with the following elements:

- Relation:** A central class with a 'Degré' attribute.
- Attributs:** A class with a 'Name' attribute.
- Domaine:** A class with a 'Valeur' attribute.
- ClePrimaire:** A class with a 'ClePrimaire' attribute.
- CleEtrangere:** A class with a 'CleEtrangere' attribute.

A dialog box titled 'Slyum - New view' is open, asking 'Enter a name for the new view:'. It has two radio buttons: 'UML' (selected) and 'Relational'. The 'OK' and 'Cancel' buttons are at the bottom.

At the bottom of the interface, there is a section for 'Project's properties' and 'Project's informations'. A red box highlights a text area with the placeholder text 'Relational Problems/all clear goes here'.

Le choix de uml ou relationnel quand on crée une nouvelle vue ne change pas les classes utilisées mais détermine les fonctions utilisables sur la GUI (ex : on peut créer des liens d'héritage en uml mais pas en relationnel)

Je profite aussi de cette vue globale pour montrer où se situent les résultats de la validation du schéma relationnel.

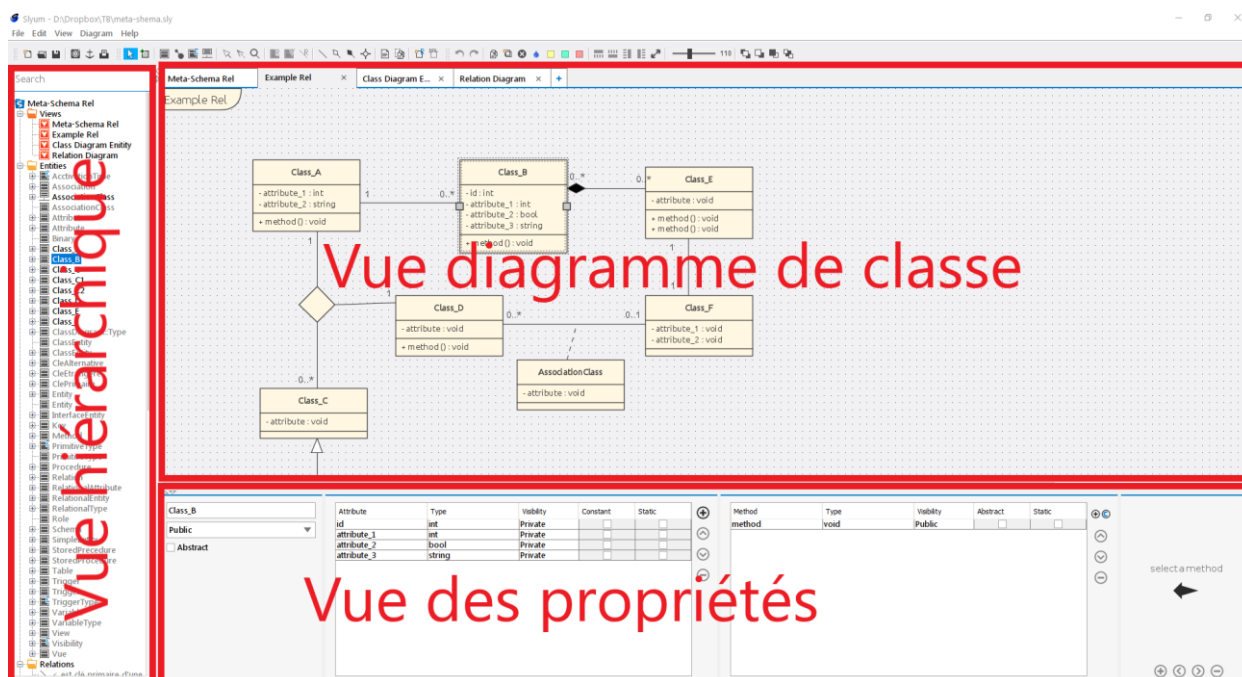
7 Structure générale de Slyum

Le projet original de Slyum est séparé en 4 parties.

La partie principale est la structure des classes qui contient les toutes les entités objet des schémas (classes, enums, attributs, associations, etc.) auquel j'ai rajouté les entités relationnelles. Cette partie permet de stocker tous les composants d'un diagramme de classe (et d'un schéma relationnel maintenant).

Les trois autres parties de Slyum sont des composants de l'interface graphique de Slyum.

- La "Vue diagramme de classe" dans laquelle est affichée la représentation graphique du schéma.
- La "Vue hiérarchique" dans laquelle est stockée une représentation en arbre de tous les éléments du schéma.
- La "Vue des propriétés" dans laquelle est affichée les propriétés de l'élément graphique sélectionné.



Ces trois parties sont reliées à la structure principale avec patron de conception "observer" qui est implémenté dans Slyum avec l'api Observer-Observable de java.

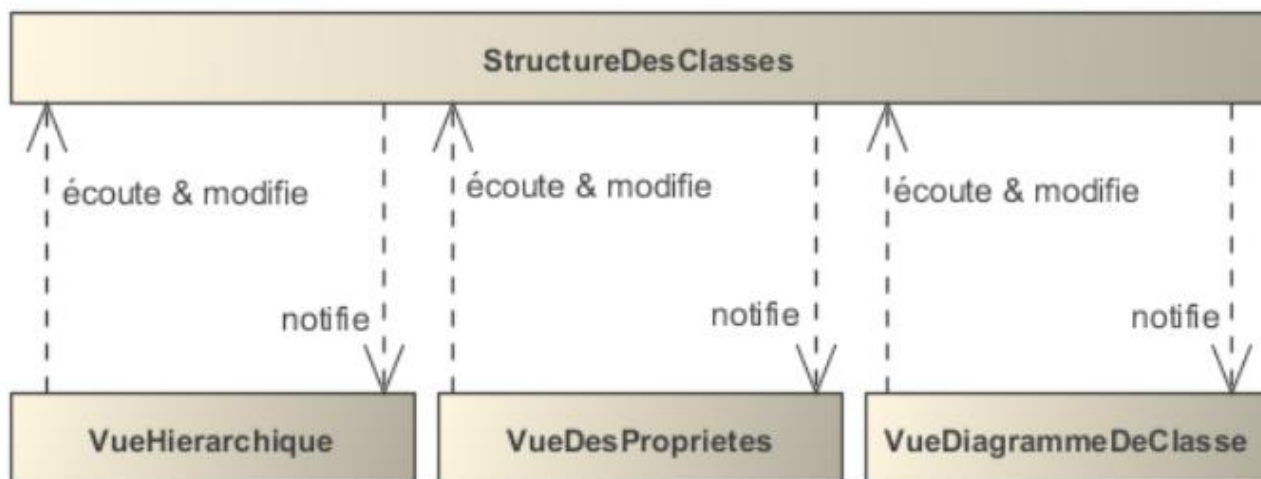


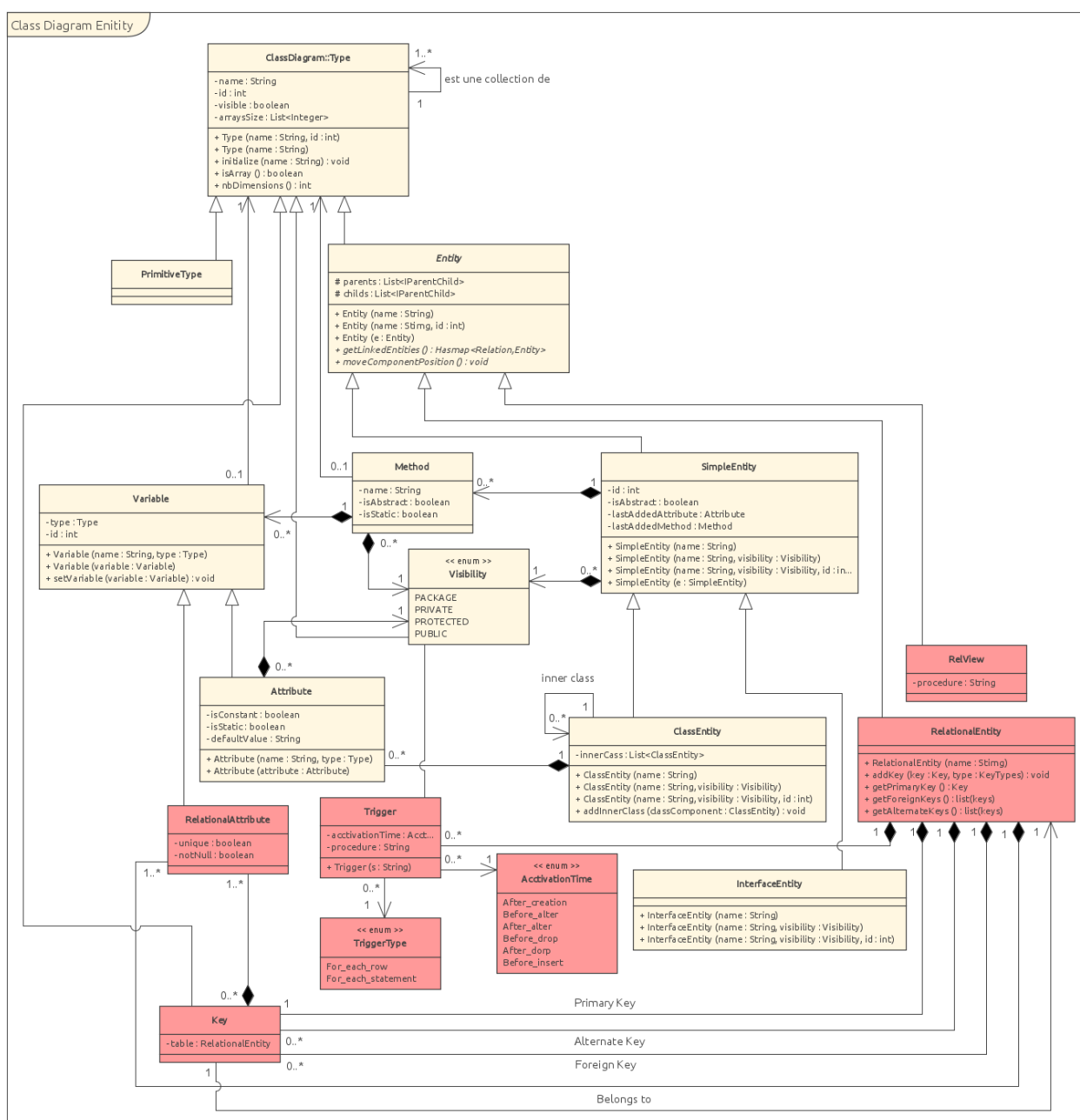
Illustration de la structure générale du projet du rapport Slyum de 2011 par David Miserez

7.1 Librairie graphique Swing

L'interface graphique de Slyum utilise le Framework graphique Swing qui fut ajouté à la JDK en 1997. C'est un Framework relativement simple et beaucoup utilisé à l'époque. En 2020 il est plutôt daté et d'autres alternatives existe mais j'en parlerai plus à la conclusion du rapport.

8 Création et visualisation du schéma relationnel

Certaines modifications ont pu être faites sur le diagramme original vu que le code de Slyum contenait une classe abstraite "simpleEntity" qui s'interpose entre la classe d'entité générique et les spécialisations UML (Classe, Enum, etc.) et qui contiens des éléments du diagramme UML non-nécessaires au schéma relationnel. J'ai pu du coup faire hériter mes composants relationnels à l'entité générique (Entity) et du coup réutiliser une plus grande partie du code.



Pour rappel tous les objets héritent de Type et peuvent donc être représenté dans un même schéma. Ça sera le rôle de l'interface de faire en sorte que l'utilisateur ne mette pas des éléments uml et rel dans le même diagramme.

8.1 Nouveaux éléments

Pour chacun de ces éléments il faut créer une classe de structure et trois autres classes pour chacune des trois vues graphiques de Slyum : GraphicView, Property et Node soit une entité visuelle, un panneau de contrôle et un nœud ajoutable dans l'arbre des entités. Certains éléments n'ont pas de Classe GraphicView ou Property à leur nom car elles font partie de celle d'une sur-entité (ex : les propriétés des attributs sont dans celle des tables

8.1.1 Tables

Une table a un nom, une clé primaire, une liste d'attributs, une liste de clé alternatives, une liste de clés étrangères et une liste de triggers.

8.1.1.1 Attributs relationnels

Un attribut relationnel est implémenté de façon similaire aux attributs UML. Elles ont un nom et un type mais aussi deux valeurs booléennes pour définir si elles sont qualifiées d'unique et/ou de non nulle.

8.1.1.2 Clés

Une clé peut correspondre à un des 3 types de clés (primaire, alternative, étrangère). Elle a un nom, une liste d'attributs relationnels et une table qui correspond à sa table d'origine.

8.1.1.3 Triggers

Un trigger est composé d'un nom et d'une procédure.

8.1.2 Vues

Une vue a un nom et une procédure stockée sous la forme d'un string.

8.1.3 Relations

Une relation a les mêmes attributs qu'une association binaire, c'est à dire un nom, une entité source, une entité cible et une direction.

8.2 Procédures des vues et des triggers

Dans cette version de Slyum, les vues relationnelles et les triggers stockent leur procédure respective sous la forme d'une chaîne de caractères. Cela veut dire que si l'utilisateur veut utiliser la fonction de génération de scripte SQL il doit connaître à l'avance quel SGBD il vise et écrire la procédure manuellement.

Pour le moment elles sont de simples strings mais elles pourraient, dans une future itération de Slyum, être améliorés en plusieurs objets plus complexes qui pourraient stocker la procédure indépendamment de la SGBD visé.

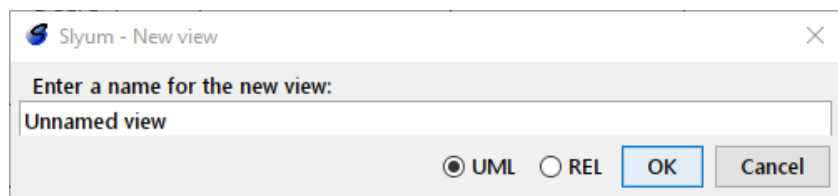
```
SELECT *  
FROM films  
WHERE kind = 'Comedy'
```

Exemple de procédure de <https://www.postgresql.org/docs/9.2/sql-createview.html>

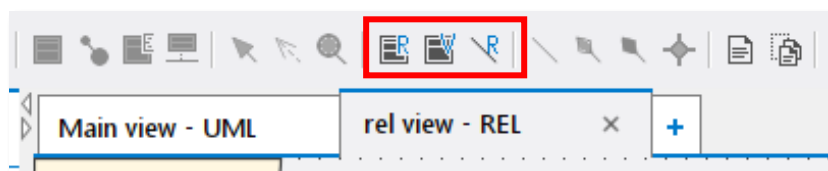
8.3 Vues graphiques

8.3.1 Vue du diagramme

A la création d'un nouveau diagramme, le dialogue de création propose de choisir entre créer une vue relationnelle et une vue UML.



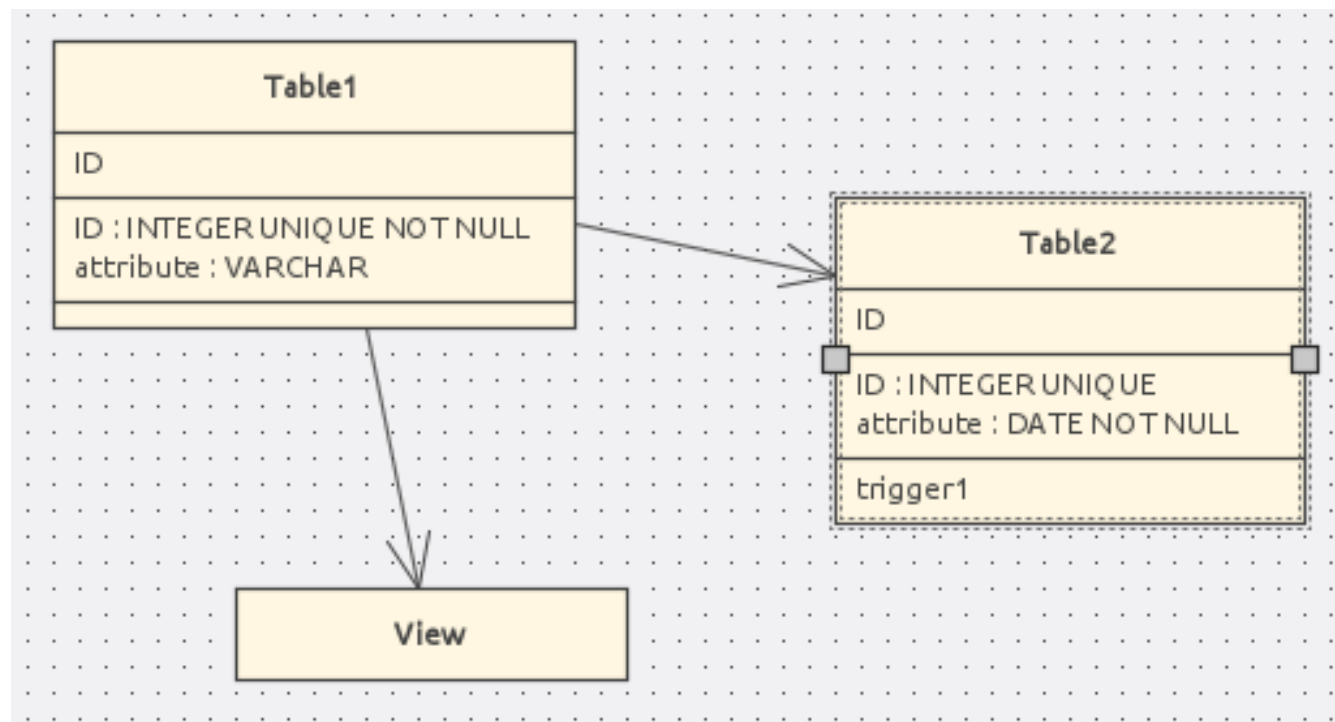
Quand une vue relationnelle est créée, les éléments d'interface pour ajouter des entités de diagramme UML sont désactivés et ceux des entités relationnelles sont activés.



De plus, quand une nouvelle vue graphique (un onglet) est créée, un identifiant (-UML ou -REL) est rajouté à l'onglet pour identifier son type. Celui-ci peut faire partie du nom de la vue et peut du coup être enlevé si l'utilisateur désire.

Les entités instanciables sont dans l'ordre : les tables, les vues et les liens entre ces entités.

Les tables montrent dans l'ordre : les clés, les attributs et les triggers. Vu que le seul composant de la vue relationnelle est ça, seule son nom est affiché dans la vue du diagramme. Les flèches indiquent le sens de la relation. La clé étrangère est placée du côté de la source. La liaison entre table et vue est purement visuelle.



8.3.2 Vue hiérarchique

La vue hiérarchique est une représentation graphique des éléments des diagrammes sous la forme d'un arbre. Elle utilise le composant Jtree de la librairie Swing.

Vu que tous les éléments de toutes les représentations graphiques sont affichés, les entités UML sont mélangées aux relationnelles. Les nouveaux éléments ont des icônes différentes qui permettent de les différencier. De plus, les éléments qui ne sont pas de la vue graphique ouverte sont grisés donc normalement si une vue graphique relationnelle est ouverte les éléments UML sont grisés.

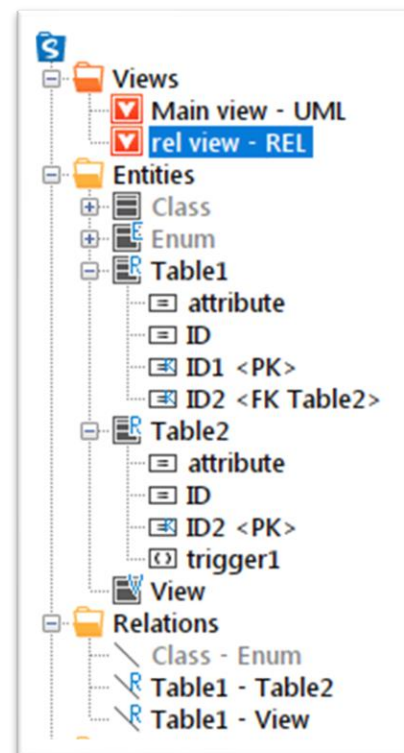
Les tables ont 3 sous éléments possibles : les attributs, les triggers et les clés.

Le type de la clé est indiqué dans les crochets. PK pour clé primaire, AK pour clé alternative et FK pour clé étrangère. Si la clé est étrangère, ça table d'origine est aussi noté.

8.3.3 Sauvegarde

Le format XML du fichier de sauvegarde de Slyum est maintenu mais de nouveaux éléments sont rajoutés pour sauvegarder les nouvelles entités. Le seul ancien élément qui est changé est la vue graphique dans laquelle j'ai rajouté un attribut pour designer s'il s'agit d'une vue UML ou relationnelle. Cette propriété est mise à UML si l'attribut n'est pas présent donc il est entièrement possible de charger des fichiers de sauvegarde de la version précédente de Slyum.

Ceci veut dire qu'il est possible de lire les fichiers de sauvegarde de la version précédente avec la nouvelle version et vice-versa mais les nouveaux éléments ne seront pas parsés par le parseur XML de la version précédente.



```
<entity entityType="ENUM" id="49194" name="Enum">
  <EnumValue>VALUE</EnumValue>
</entity>
<entity entityType="CLASS" id="49193" isAbstract="false" name="Class" visibility="PUBLIC">
  <attribute const="false" defaultValue="" isStatic="false" name="attribute" type="void" visibility="PRIVATE"/>
  <attribute const="false" defaultValue="" isStatic="false" name="attribute" type="void" visibility="PRIVATE"/>
  <method is-constructor="true" isAbstract="false" isStatic="false" name="Class" returnType="" view="DEFAULT" visibility="PUBLIC"/>
</entity>
<entity entityType="TABLE" id="48254" name="Table1">
  <relationalAttribute const="false" defaultValue="" id="48262" name="ID" notNull="true" type="INTEGER" unique="true"/>
  <relationalAttribute const="false" defaultValue="" id="48435" name="attribute" notNull="false" type="VARCHAR" unique="false"/>
  <key id="48254" name="ID1" primary="true">
    <raID id="48262"/>
  </key>
</entity>
<entity entityType="TABLE" id="48256" name="Table2">
  <relationalAttribute const="false" defaultValue="" id="48500" name="ID" notNull="true" type="INTEGER" unique="true"/>
  <relationalAttribute const="false" defaultValue="" id="48666" name="attribute" notNull="true" type="DATE" unique="false"/>
  <key id="48256" name="ID2" primary="true">
    <raID id="48500"/>
  </key>
  <trigger activationTime="After Creation" name="trigger1" procedure="Hello, world!" triggerType="For Each Row"/>
</entity>
<entity entityType="VIEW" id="48721" name="View" procedure="What a nice view"/>
```

Extrait de fichier de sauvegarde avec des entités UML et relationnelles

8.3.4 Vue des propriétés

Quand un élément de la vue du diagramme est sélectionné la vue des propriétés affiche ces propriétés.

8.3.4.1 Propriétés des tables

The screenshot shows the 'Table Properties' dialog box. It is divided into several sections:

- 1**: 'Table Name' field containing 'Table1'.
- 2**: 'Primary Key Name' field containing 'ID1'.
- 3**: 'Pk Attributes' section containing a list of attributes for the primary key, currently showing 'ID'.
- 4**: 'Attributes' table with columns: Attribute, Type, Unique, and Not Null. It lists 'ID' as an INTEGER and 'attribute' as a VARCHAR.
- 5**: 'Triggers' section showing 'No trigger'.
- 6**: 'Trigger Procedure' section with the text 'Select a Trigger to Edit'.

1. Le nom de l'entité
2. Le nom de la clé primaire
3. Les attributs de la clé primaire
 - Les flèches peuvent réordonner les attributs.
 - Le moins supprime l'attribut sélectionné de la clé primaire.
4. Tous les attributs de la table
 - On peut rajouter un nouvel attribut avec le plus.
 - Les flèches peuvent réordonner les attributs.
 - Le moins supprime l'attribut sélectionné de la table.
 - Le plus du bas rajoute l'attribut sélectionné à la clé primaire
5. Les triggers de la table
 - On peut rajouter un nouveau trigger avec le plus.
 - Les flèches peuvent réordonner les triggers.
 - Le moins supprime le trigger sélectionné de la table.
6. La procédure du trigger sélectionné

8.3.4.2 Propriétés des vues

The screenshot shows the 'View Properties' dialog box. It is divided into two main sections:

- Relational View Name**: A text field containing the word 'View'.
- View Procedure**: A large, empty text area for defining the view's procedure.

Les propriétés des vues sont relativement simples. Le petit champ sur la gauche est le nom de la vue et le gros champ sur la droite est la procédure de cette vue.

8.3.4.3 Propriétés des relations


Les propriétés des relations sont simples aussi. Il y a le nom de la relation qui est optionnel et deux boutons radiaux qui indiquent le sens de la relation.

Quand le sens de la relation est changé la clé étrangère est supprimée puis elle est rajoutée du côté où elle doit être.

8.3.5 Propriétés clés alternatives

Les propriétés des tables étaient sensées contenir deux tableaux pour gérer les clés alternatives. Malheureusement, j'ai mal calculé la place que chaque élément prendra et je n'avais plus la place de les rajouter à l'interface et, au lieu de perdre du temps à refaire ma modélisation et trouver une façon ergonomique de rajouter cet élément d'interface, j'ai préféré mettre ça de côté et me concentrer sur les éléments du projet que j'ai jugé plus important.

Cela veut dire qu'un utilisateur ne peut pas rajouter des clés alternatives via l'interface. La seule façon de les rajouter est de les rajouter manuellement dans un fichier de sauvegarde et de charger ce fichier.



8.3.6 Procédures stockées

Les procédures stockées ne font pas partie du schéma relationnel au sens propre mais elles sont souvent ajoutées dans SGBD pour la validation, des mécanismes de contrôle d'accès, etc. A la modélisation j'avais proposé de les rajouter mais à cause de soucis de temps j'ai pris la décision de ne pas les rajouter.

Vu que leur implémentation aurait été relativement similaire à celle des vues mis appart quelques soucis d'interface et de conversion SQL, je ne pense pas que leur absence soit une grande perte pour ce projet.

8.3.7 Refactoring

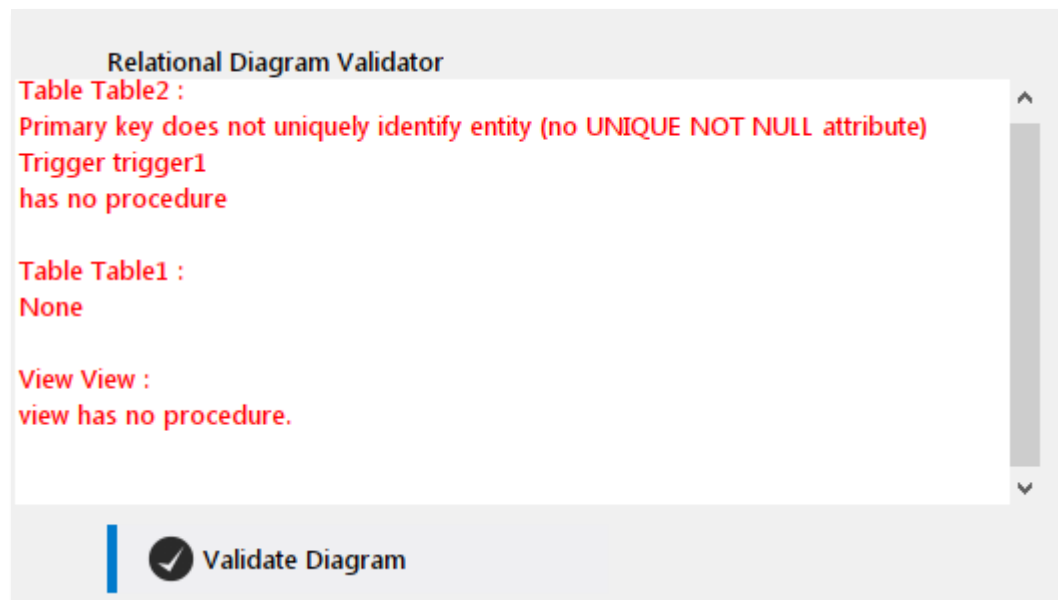
Pour pouvoir implémenter les fonctionnalités du créateur de schéma relationnel, j'ai dû beaucoup analyser et parfois changer le code déjà présent. J'ai du coup pu remarquer, en grande partie grâce à mon IDE, qu'il y a beaucoup de code qui pouvait être refactorisé.

- Des appels méthodes qui peuvent être remplacé par des méthodes lambda.
- Des attributs de classes qui peuvent devenir finaux ou être rendu local à une méthode.
- Du code dupliqué.
- Etc.

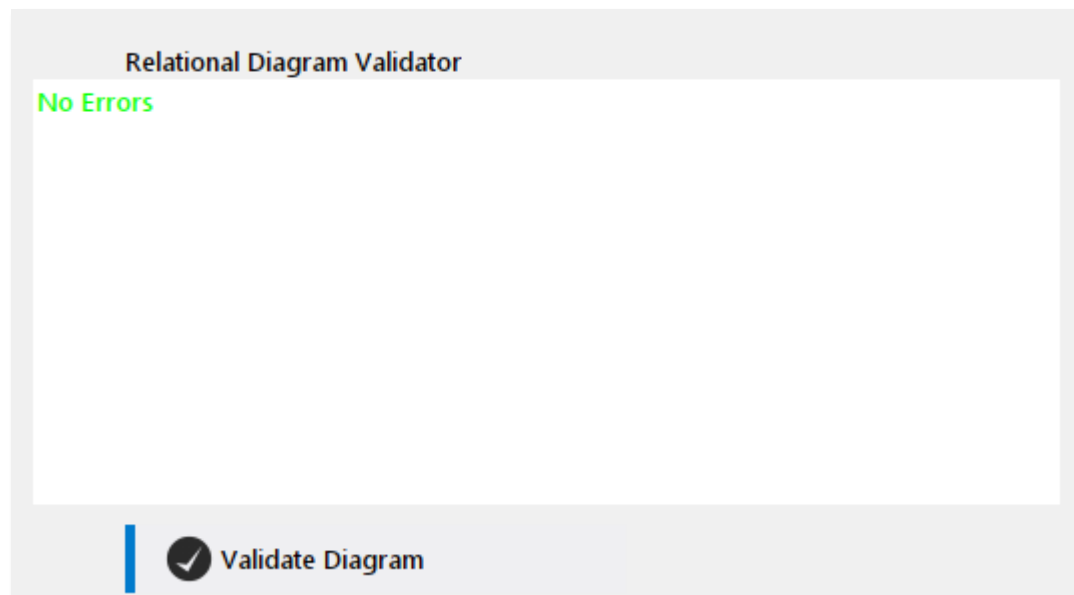
J'ai corrigé ces erreurs quand je pouvais les corriger sans trop de problèmes mais il en reste encore beaucoup du coup un passe de correction et de refactoring du code pourrait s'avérer nécessaire si on voulait rajouter de nouvelles fonctionnalités ou faire des changements sur celles déjà présente.

8.3.8 Valideur

Quand aucun élément n'est sélectionné l'utilisateur peut voir les options globales. J'y ai rajouté une section qui permet de trouver les erreurs dans un schéma relationnel. Quand l'utilisateur appuie sur le bouton un rapport des erreurs est généré et est affiché.



Exemple de rapport d'erreurs du valideur relationnel



Exemple de rapport d'erreur vide

Le valideur est aussi lancé quand l'utilisateur essaye de générer un script SQL pour ne pas avoir des erreurs vérifiables dans le scripte.

8.3.9 Règles de validation

Tables :

- Une table doit avoir un nom unique (inclut les vues).
- Une table doit avoir une clé primaire et cette clé doit être valide.
- Une table doit avoir au moins 1 attribut.
- Chaque attribut doit avoir un nom unique.
- Chaque trigger doit être valide
- Chaque clé alternative doit être valide

Clé :

- Une clé doit avoir un nom.
- Une clé doit avoir au moins 1 attribut.
- Une clé doit pouvoir identifier sa table (elle doit contenir au moins un attribut unique)

Triggers :

- Un trigger doit avoir un nom unique.
- Un trigger doit avoir une procédure.

Vues :

- Une vue doit avoir un nom unique (inclut les tables).
- Une vue doit avoir une procédure.

Cycles :

Un schéma relationnel ne doit pas contenir de cycles. Je détecte les cycles en transformant le schéma en un graphe orienté puis j'utilise un algorithme de recherche de cycles en DFS repris du site www.geeksforgeeks.org pour détecter un cycle.

Ces règles permettent de garantir une justesse relative du schéma et du script SQL généré. En effet, quelques éléments ne sont pas vérifiés strictement et peuvent du coup encore contenir des erreurs (ex : le contenu de la procédure n'est pas vérifié ou les types des attributs ne sont pas garanti d'exister dans SQL).

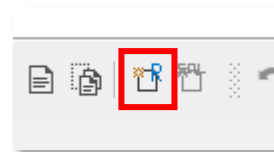
8.4 Améliorations possibles

Voici une liste non-exhaustive de potentielles améliorations qui pourraient être apporté au créateur de schéma relationnels avec plus de temps de développement.

- Rendre les procédures des vues et les triggers plus génériques de façon à ce qu'elles puissent être transformé en un scripte SQL adapté à la SGBD visé.
- Inclure les procédures stockées qui ont été mise de côté pour le développement de cette version de Slyum.
- Rajouter des raccourcis claviers pour les éléments graphiques relationnels.
- Trouver une façon ergonomique d'inclure l'interface de gestion des clés alternatives.
- Changer les types des attributs de façon à ne que pouvoir choisir entre des types d'attributs défini ou alors changer la validation pour détecter si le type est valide.
- Faire une passe de refactoring sur le code de Slyum UML.
- Améliorer le validateur relationnel de façon à pouvoir valider les modifications proposées ci-dessus mais aussi rajouter des éléments de validation liés à des SGBD spécifiques qui seraient lancés au moment de la génération du script SQL.

9 Conversion du diagramme UML en relationnel

Quand une vue(onglet) UML est sélectionné, l'utilisateur peut convertir le diagramme UML en schéma relationnel en utilisant les règles définies plus haut dans ce rapport (section 4).



9.1 Implémentation

Le convertisseur est une classe singleton nommé RelConverter qui se situe dans le package utility.relConverter. Elle prend une vue graphique UML comme attribut, convertit un à un les éléments du diagramme, les rajoute dans la liste de toutes les entités du projet et crée de nouvelles entités graphiques dans une nouvelle vue graphique aux mêmes coordonnées que l'entité dans la vue graphique UML. Cette nouvelle vue graphique est rajoutée dans la liste des vues dans la classe qui gère les vues du projet (MultiViewManager).

9.2 Exemple de conversion

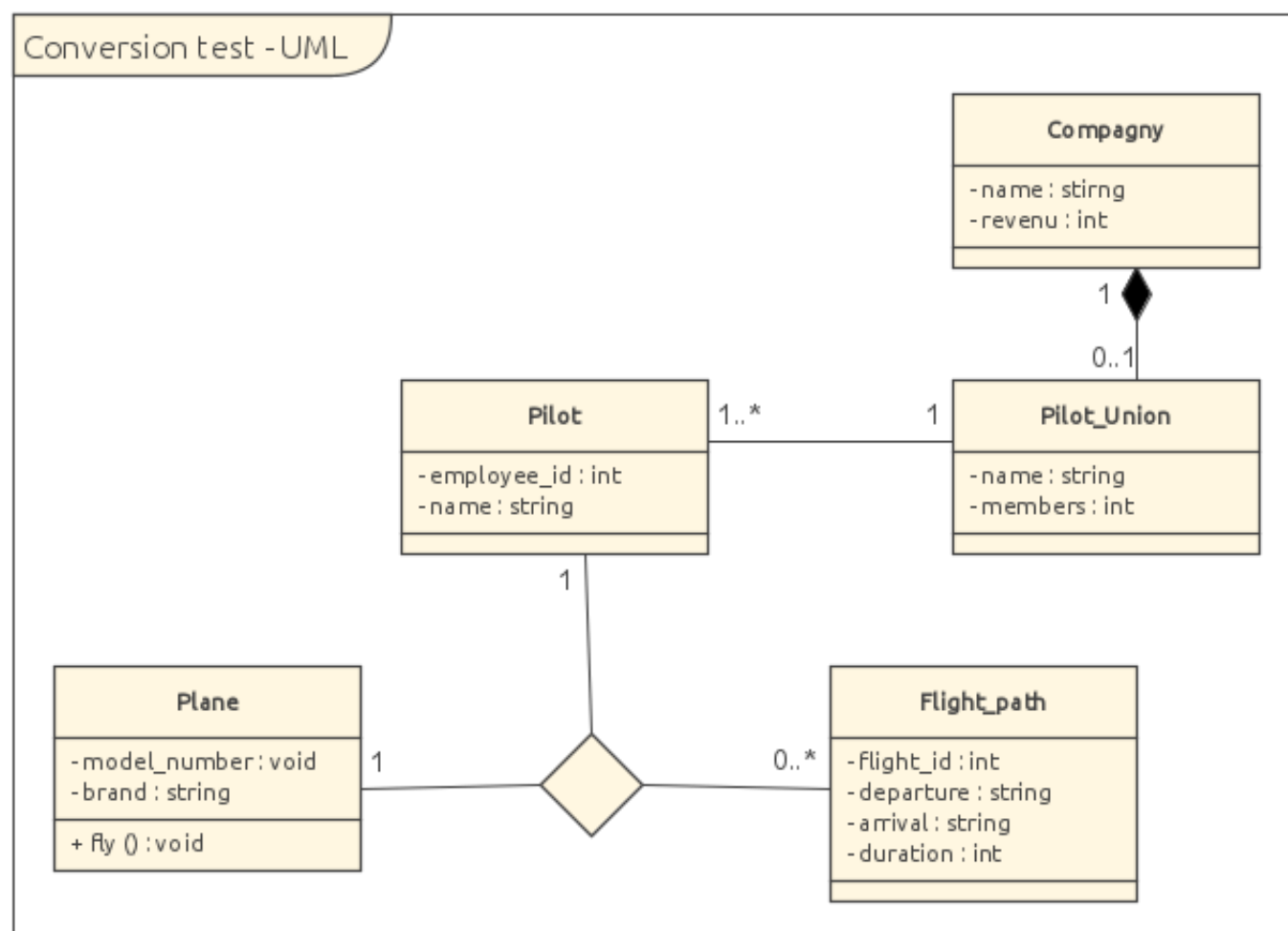


Diagramme UML à être converti

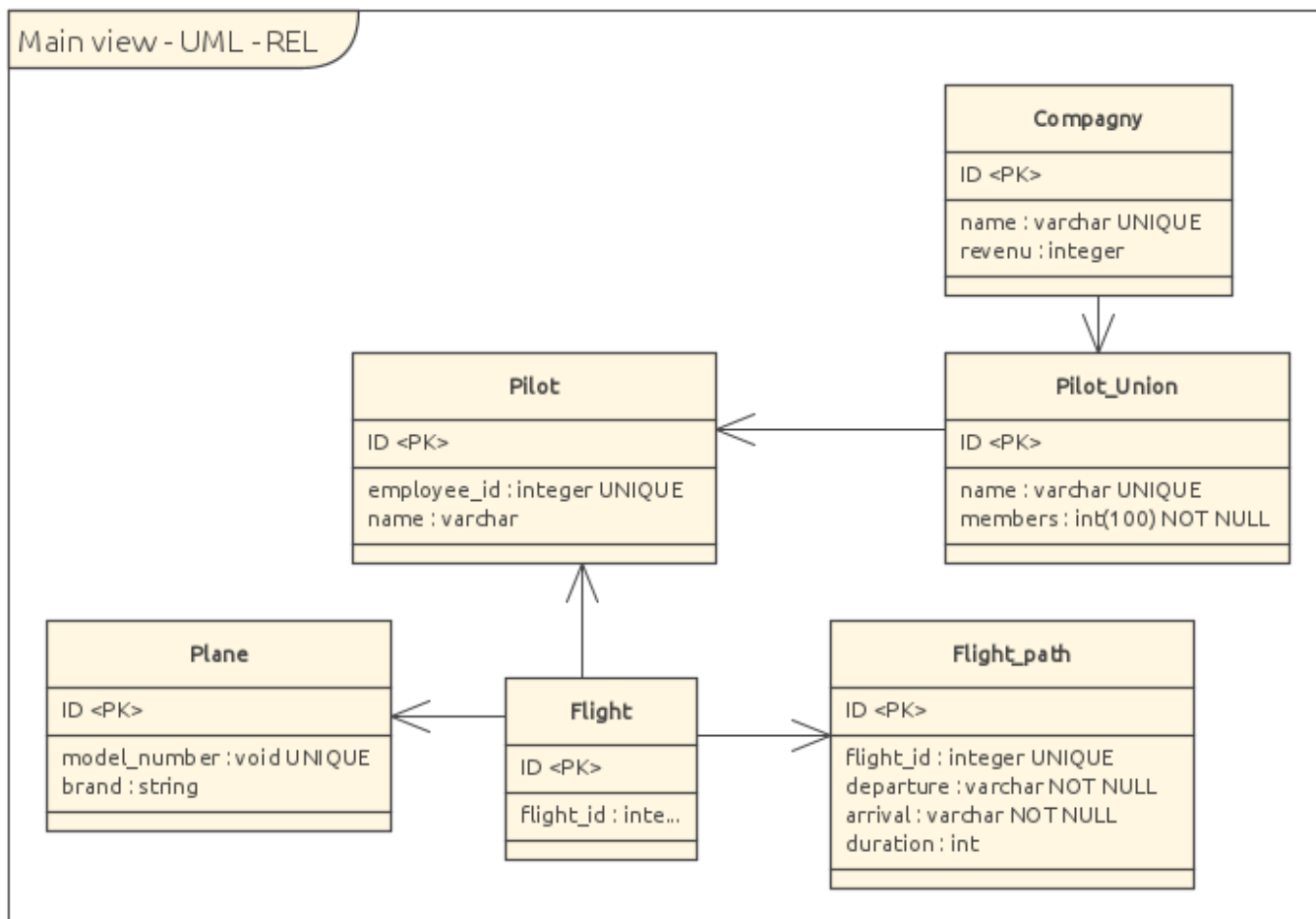
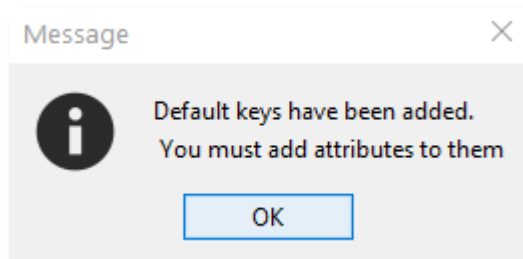


Diagramme UML converti en Schéma relationnel

Une fois le diagramme converti, l'onglet du diagramme converti est ouvert et un message s'affiche pour vous indiquer que des clés primaires par défaut ont été créées et que c'est à l'utilisateur de changer leur nom s'il veut et de les peupler avec des attributs.

Il faut aussi changer les types des attributs pour refléter leur type SQL ainsi que mettre les attributs en unique et/ou non-null.

Dans le schéma ci-dessus j'ai déjà effectué ces changements et j'ai renommé la table créée pendant la conversion de l'association multiple (qui s'appelle "multi" par défaut).



10 Conversion du schéma relationnel en scripte SQL

Quand une vue(onglet) REL est sélectionnée, l'utilisateur peut convertir le schéma relationnel en un scripte SQL via un algorithme de "génération procédurale".



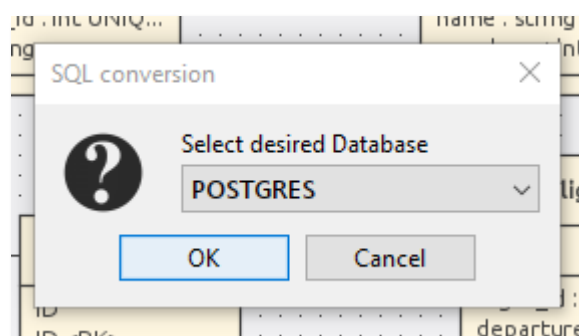
L'algorithme convertit les tables et vues une à une et les rajoute dans une chaîne de caractères qui sera ensuite écrite dans un fichier ".sql".

A la remise de ce rapport, il est possible de générer un scripte pour 2 des 3 SGBD. Vu que cette fonctionnalité fut la dernière que je développai, j'ai décidé de ne pas perdre trop de temps et de me concentrer sur les plus importantes : MySQL car c'est la SGBD open-source la plus utilisée et PostgreSQL car c'est la SGBD de prédilection du professeur qui a proposé ce projet.

10.1 Implémentation

Le générateur SQL est une classe singleton nommé SQLConverter qui se situe dans le package "utility.SQLConverter". Elle prend une vue graphique REL comme attribut, extrait les classes et les vues de celle-ci et les convertit en un scripte SQL qui peut être utilisé pour créer une base de données.

Quand la génération est lancée, le validateur relationnel est lancé. S'il y a toujours des erreurs un popup apparaît pour les indiquer à l'utilisateur. S'il n'y a plus d'erreurs l'utilisateur est demandé de choisir la SGBD pour laquelle il veut créer le scripte.



Le convertisseur va ensuite prendre la vue de digramme ouverte et extraire chaque entité table et vue et les convertir en un string qui sera écrit dans le fichier ".sql"

```
private String convertTableMySQL(RelationalEntity entity) {
    StringBuilder sb = new StringBuilder("DROP TABLE IF EXISTS ");
    sb.append(entity.getName()).append("\n\n");
    sb.append("CREATE TABLE ").append(entity.getName()).append(" (");

    //attributes
    for (RelationalAttribute ra : entity.getAttributes()) {
        sb.append(" ").append(convertAttribute(ra, fkAttribute: false)).append(",\n");
    }

    //add fk attributes
    entity.getForeignKeys().forEach(fk -> fk.getKeyComponents().forEach(
        ra -> sb.append(" ").append(fk.getTable().getName()).append("_").append(
    ));

    //primary key
    sb.append(" PRIMARY KEY(");
    for (int i = 0; i < entity.getPrimaryKey().getKeyComponents().size(); i++) {
        RelationalAttribute ra = entity.getPrimaryKey().getKeyComponents().get(i);
        sb.append(ra.getName());
        if (i+1 < entity.getPrimaryKey().getKeyComponents().size()) {
            sb.append(",");
        }
    }
    sb.append(")");

    //alternate keys
    for (Key ak : entity.getAlternateKeys()) {
```

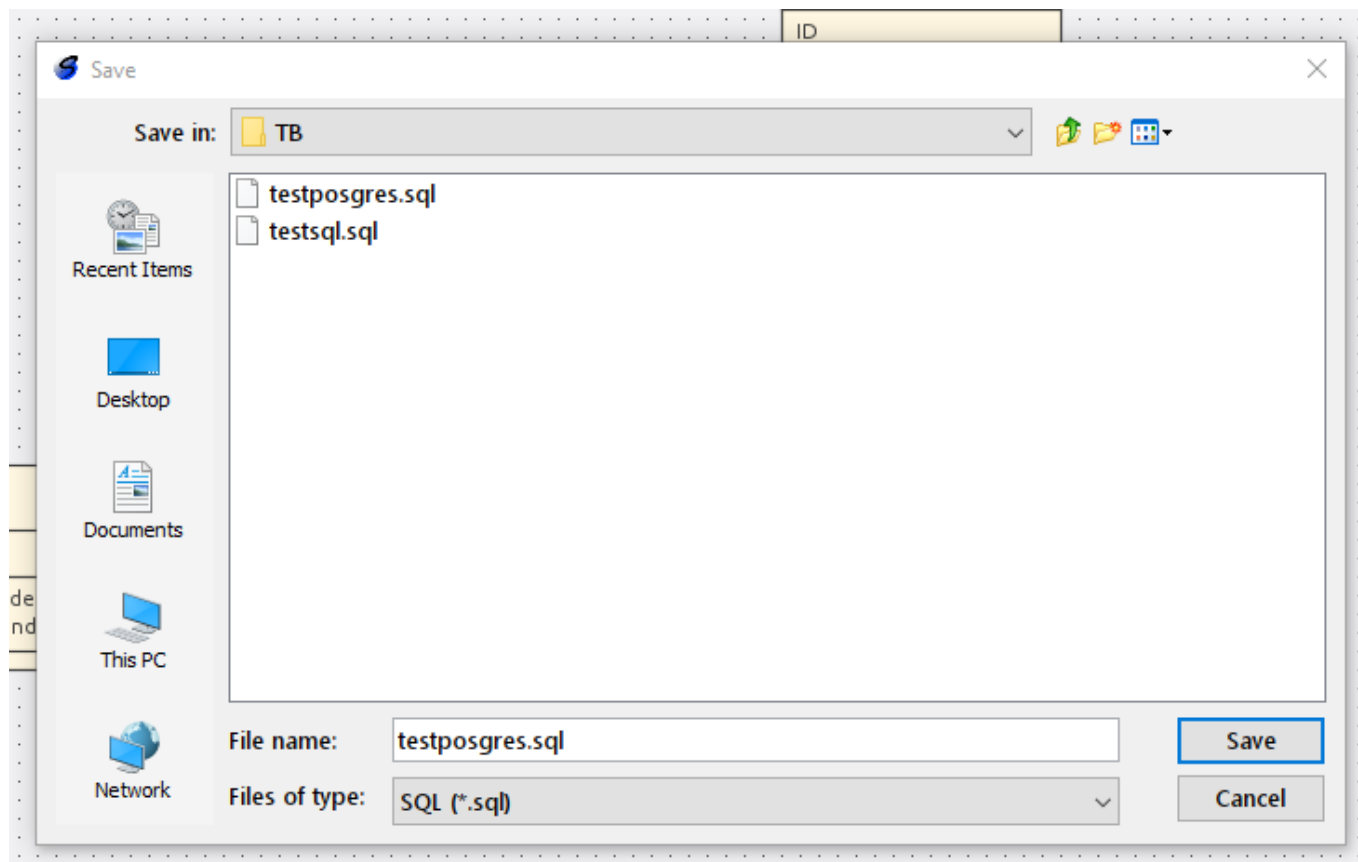
```
DROP TABLE IF EXISTS Compagny;

CREATE TABLE Compagny (
    name varchar UNIQUE,
    revenu integer,
    ID_name varchar,
    CONSTRAINT ID PRIMARY KEY (name),
    CONSTRAINT ID
    FOREIGN KEY (Pilot_Union_name)
    REFERENCES Pilot_Union (name)
);
```

Extrait de code de génération de classe

Exemple de script de la table Compagny

Une fois le scripté généré une fenêtre s'ouvre pour proposer à l'utilisateur ou le sauvegarder.



11 Conclusion

11.1 Problèmes connus

- Le changement de direction des associations relationnelles ne change pas la disposition des clés étrangères.
- L'affichage des clés dans la vue du diagramme ne s'actualise pas correctement.
- Quand un projet est chargé, les clés dans la vue du diagramme n'affichent pas le type de la clé.
- Le bouton de l'interface de propriété des tables qui permet de rajouter des attributs à la clé primaire n'est pas désactivé quand aucun attribut de la table n'est sélectionné.
- Lorsqu'une vue graphique(onglet) est supprimée les boutons de créations d'entités et relations ne sont pas activés/désactivés pour correspondre au type de la vue (UML/REL)
- Il est possible de placer des éléments UML dans un schéma relationnel et vis-versa. Des mesures de précaution ont été mise en place dans le validateur de schéma mais des éléments d'interface, particulièrement la vue hiérarchique et le menu contextuel du clic droit sur une entité.

11.2 Améliorations possibles

Il y a une pléthore d'améliorations possibles comme celles que j'ai proposé à la fin du chapitre 8 sur la représentation graphique. Mais une des améliorations principales que j'apporterai à ce logiciel ne serait même pas une simple amélioration, mais un recodage complet de l'interface du logiciel. En effet quand la première version de Slyum fut créée en 2010 la librairie graphique Swing était une des plus utilisés en java mais en 2012 Swing fut remplacé par javaFX comme le Framework graphique java par défaut. De ce fait des informations pour créer des interfaces deviennent de plus en plus rare et les créateurs d'interface des IDE modernes passent au javaFX.

11.3 Slyum 2.0

Le logiciel Slyum a été itéré dessus à plusieurs reprises au point où le code commence à devenir confus. Les changements que j'y ai apporté n'ont fait qu'exacerber ce problème car celui-ci n'avait pas été prévu pour supporter des schémas relationnels à sa conception. De plus, Slyum est un logiciel qui a été développé il y a maintenant 10 ans et le style de l'interface commence à montrer son âge.

Si j'avais beaucoup plus de temps j'aurais proposé de garder le concept de Slyum mais de le remodeliser pour prendre en compte les schémas relationnels de tel sorte à avoir une base plus propre et plus solide si de nouvelles fonctionnalités devaient être rajoutées. De plus je pense que malgré le fait que la Java Virtual Machine a le bénéfice de fonctionner nativement sur toutes les plateformes elle est un peu lente d'exécution. Si je pouvais, je recoderais le logiciel en C++ en utilisant un Framework graphique plus performant comme Qt ou GTK.

11.4 Conclusion personnelle

Ce projet a été difficile pour moi. Non seulement parce que la situation covid-19 et le confinement a perturbé tout le monde mais j'ai dû faire face des problèmes médicaux en même temps. Malgré cela j'ai persévéré et pour finir j'ai réussi à implémenter les 3 fonctionnalités majeurs que j'avais prévu.

Il est possible de créer un schéma relationnel qui contient des tables et des vues, sur ces tables on peut rajouter des attributs et des triggers on peut aussi rajouter ces attributs à la clé primaire de la table. Sur les vues on peut écrire une procédure. On également créer des liens entre les tables qui vont rajouter les clés étrangères dans ces tables.

On peut aussi convertir un diagramme UML en un schéma relationnel avec des clés primaires par défaut.

Finalement on peut convertir ce schéma relationnel en un scripte SQL pour les SGDB MySQL et PostgreSQL.

Certaines fonctionnalités mineures ont été coupée comme les procédures stockées ou la conversion pour SQLite et un élément d'interface n'a pas été rajouté (les clés alternatives) mais dans l'ensemble je suis satisfait de l'utilisabilité du logiciel même si j'aurai voulu avoir le temps de le peaufiner un peu plus.

12 Annexes

12.1 Journal de travail

12.1.1 Semaine 1 – 17 au 23 Février

- Lecture des directives de TB de l'HEIG
- Lecture du rapport Slyum
- Rapport Slyum

12.1.2 Semaine 2 – 24 Février au 1 Mars

- Rédaction du cahier des charges

12.1.3 Semaine 3 – 2 au 8 Mars

- Meta-schéma relationnel
- Meta-schéma rel. et modification du diagramme de classe UML pour implémenter REL

12.1.4 Semaine 6 – 23 au 29 mars

- Continuation modification du diagramme de classe UML pour implémenter REL

12.1.5 Semaine 7 – 30 Mars au 5 Avril

- Modélisation rapport
- Séparation des classes UML et REL

12.1.6 Semaine 8 – 6 au 12 Avril

- RDV assistants
- Reformulation de la modélisation

12.1.7 Semaine 9 – 13 Au 19 Avril

- Introduction des triggers dans le schéma
- Introduction des vues et des procédures stockées
- Changement de l'héritage de RelationalEntity et RelationalAttribute
- Mise à jour du meta-schéma

12.1.8 Semaine 10 – 20 au 26 Avril

- Finalisation de la modélisation

12.1.9 Semaine 11 – 27 Avril au 3 Mai

- Implémentation de classes
 - RelationalAttribute
 - BufferRelationalAttribute
 - Key
 - RelationalEntity
- Changement de l'héritage de RelationalEntity grâce à la découverte de SimpleEntity

12.1.10 Semaine 12 - 4 au 10 Mai

- Lecture et compréhension du code graphique Slyum

12.1.11 Semaine 13 – 11 au 17 Mai

- Révision du Framework swing
- Suite révision swing
- Essai et expérimentation avec le code graphique de Slyum

12.1.12 Semaine 14 – 18 au 24 Mai

- Implémentation de la sélection de schéma “uml ou rel”
- Début de UI “rel-only” et implémentation du swap de UI quand on change de schéma

12.1.13 Semaine 15 – 25 au 31 Mai

- Début de l'implémentation des éléments graphiques du schéma relationnel
- Investigation de comment créer la UI pour les “vues” et “procédures stockées”

12.1.14 Semaine 16 – 1 au 7 Juin

- Remodélisation de l'affichage des clés
- Remodélisation des Views et StoredProcedures

12.1.15 Semaine 17 – 8 au 14 Juin

- Début reimplémentation de la UI des vues
- Début rédaction rapport intermédiaire

12.1.16 Semaine 18 – 15 au 21 Juin

- Rédaction rapport intermédiaire
- Préparation présentation intermédiaire

12.1.17 Semaine 19 – 22 au 28 Juin

- Présentation intermédiaire

12.1.18 Semaine 20 – 29 Juin au 5 Juillet

- Investigation profonde du code Slyum des propriétés et des vues graphiques

12.1.19 Semaine 21 – 6 au 12 Juillet

- Premier draft des tables
- Propriété des attributs des tables

12.1.20 Semaine 22 – 13 au 19 Juillet

- Interface des triggers
- Interface des clés
- Relations REL
- Validateur REL
- Debugging et refactoring

12.1.21 Semaine 23 – 20 au 26 Juillet

- Convertisseur UML-Relationnel
- Clés alternatives
- Génération MySQL
- Génération PostgreSQL
- Debugging général
- Documentation code
- Début rapport final

12.1.22 Semaine 24 – 27 au 31 Juillet

- Rapport final
- Affiche
- Debugging final