

Travail de Bachelor

Slyum – éditeur de schémas relationnels

Non confidentiel



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

Yverdon-les-Bains, le 31 juillet 2020

Département TIC

Filière Informatique

Orientation Logiciel

Étudiant Yoann Rohrbasser

Enseignant responsable Pier Donini

Travail de Bachelor 2019-2020

Slyum – éditeur de schémas relationnels

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 de classes 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 :

Aigle, le 31 juillet 2020

Signature :

Yoann Rohrbasser

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.

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 Modelling Language (UML)	8
1.2	Diagrammes de classes UML	8
1.3	Modèle relationnel	8
1.4	Schéma relationnel	8
1.5	Structured Query Language (SQL)	8
2	Cahier des Charges	9
2.1	Résumé du projet Donné par l'enseignant	9
2.2	Etapes du projet	9
2.3	Création et visualisation du schéma relationnel	9
2.3.1	Fonctionnalités	9
2.3.2	Etapes	9
2.4	Conversion du schéma UML en relationnel	9
2.4.1	Fonctionnalités	9
2.4.2	Etapes	10
2.5	Conversion du schéma relationnel en SQL	10
2.5.1	Fonctionnalités	10
2.5.2	SGBD visées	10
2.5.3	Etapes	10
2.6	Liaison entre schéma relationnel et UML	10
2.6.1	Fonctionnalités	10
3	Modélisation	11
3.1	Meta-Schéma Relationnel	11
3.1.1	Schéma	11
3.1.2	Table	11
3.1.3	Clés Primaires/Etrangères/Alternatives	11
3.1.4	Attributs/TypePrimitif	11
3.1.5	Procédures Stockées/Vues/Triggers	11
3.1.6	Procédures	11
3.2	Diagramme de classe – entité	12
3.2.1	RelationalEntity	12
3.2.2	RelationalAttribute	12
3.2.3	Key	13

3.2.4	Procédure	13
3.2.5	Trigger, TriggerType et ActivationTime	13
3.2.6	StoredProcedure.....	13
3.2.7	View	13
3.3	Diagramme de classe - relation	14
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	16
4.3	Conversion des associations multiples	16
4.4	Conversion des liens d'héritage.....	17
5	Validation Schéma	17
6	Mockup Interface.....	18
7	Structure générale de Slyum	20
7.1	Librairie graphique Swing	20
8	Création et visualisation du schéma relationnel	21
8.1	Vues graphiques	22
8.1.1	Vue du diagramme	22
8.1.2	Vue hiérarchique	23
8.1.3	Sauvegarde	23
8.1.4	Vue des propriétés.....	25
8.1.5	Propriétés clés alternatives	26
8.1.6	Procédures stockées.....	26
8.1.7	Refactoring.....	26
8.1.8	Valideur	28
8.1.9	Règles de validation.....	29
8.2	Améliorations possibles.....	29
9	Conversion du diagramme UML en relationnel.....	30
9.1	Implémentation	30
9.2	Exemple de conversion.....	30
10	Conversion du schéma relationnel en scripte SQL	32
10.1	Utilisation.....	32

10.2	Implémentation	33
11	Conclusion.....	34
11.1	Problèmes connus	34
11.2	Améliorations possibles.....	34
11.3	Slyum 2.0	34
11.4	Conclusion personnelle	34
12	Annexes.....	35
12.1	Journal de travail	35
12.1.1	Semaine 1 – 17 au 23 Février	35
12.1.2	Semaine 2 – 24 Février au 1 Mars	35
12.1.3	Semaine 3 – 2 au 8 Mars.....	35
12.1.4	Semaine 6 – 23 au 29 mars.....	35
12.1.5	Semaine 7 – 30 Mars au 5 Avril	35
12.1.6	Semaine 8 – 6 au 12 Avril	35
12.1.7	Semaine 9 – 13 Au 19 Avril	35
12.1.8	Semaine 10 – 20 au 26 Avril	35
12.1.9	Semaine 11 – 27 Avril au 3 Mai	36
12.1.10	Semaine 12 – 4 au 10 Mai.....	36
12.1.11	Semaine 13 – 11 au 17 Mai.....	36
12.1.12	Semaine 14 – 18 au 24 Mai.....	36
12.1.13	Semaine 15 – 25 au 31 Mai.....	36
12.1.14	Semaine 16 – 1 au 7 Juin	36
12.1.15	Semaine 17 – 8 au 14 Juin	36
12.1.16	Semaine 18 – 15 au 21 Juin	36
12.1.17	Semaine 19 – 22 au 28 Juin	36
12.1.18	Semaine 20 – 29 Juin au 5 Juillet	36
12.1.19	Semaine 21 – 6 au 12 Juillet	36
12.1.20	Semaine 22 – 13 au 19 Juillet	37
12.1.21	Semaine 23 – 20 au 26 Juillet	37
12.1.22	Semaine 24 – 27 au 31 Juillet	37

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 compliqués et ne sont pas plaisants 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 Modelling Language (UML)

UML est un langage de modélisation graphique à base de pictogrammes conçu pour fournir une méthode normalisée pour visualiser la conception d'un système. Il est couramment utilisé en développement logiciel et en conception orientée objet.

Abréviation de la définition de UML selon [wikipedia.org](https://en.wikipedia.org/wiki/Unified_Modeling_Language)

1.2 Diagrammes de classes UML

Les diagrammes de classes UML sont un des 13 types de diagrammes du langage de modélisation unifié (UML).

Vu que les diagrammes de classes UML ne sont pas le sujet principal de ce projet je ne vais pas élaborer d'avantage la 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

Important : Le terme “diagramme de classes UML” sera abrégé en “schéma/diagramme UML”.

1.3 Modèle relationnel

Le modèle relationnel est actuellement le modèle logique de données le plus utilisé. C'est ce modèle qu'utilise la majorité des SGBD actuels. Il se base sur la notion mathématique de relation est fait largement appel à la théorie des ensembles.

Le modèle relationnel est associé à un langage algébrique appelé algèbre relationnelle qui permet la manipulation formelle des données du modèle.

Paraphrase de la définition du modèle relationnel du cours de BDR de R. Rentsch

1.4 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.

1.5 Structured Query Language (SQL)

SQL est la représentation informatique du modèle relationnel utilisé par la majorité des systèmes de gestion de bases de données (SGBD).

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 diagramme 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 schéma UML et relationnel. (Si j'ai le temps)

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 des 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 sinon les réparer.
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 le SGBD visé

2.6 Liaison entre schéma relationnel et UML

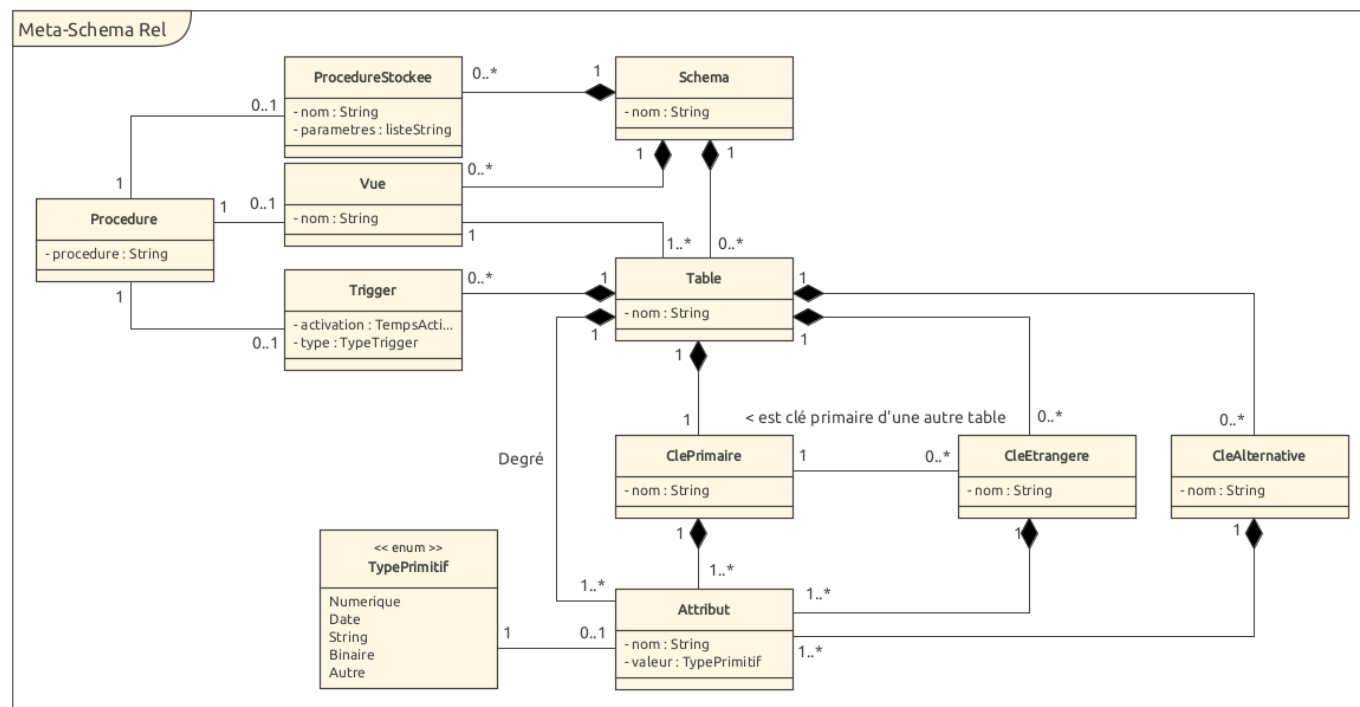
2.6.1 Fonctionnalités

- 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/TypePrimitif

Les attributs sont les différentes colonnes de la table, et TypePrimitif est le type de donnée de l'attribut en question.

3.1.5 Procédures Stockées/Vues/Triggers

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

3.1.6 Procédures

Les procédures sont une représentation textuelle du code exécutable par un SGBD.

3.2.3 Key

Key correspond au 3 types de clés du meta-schéma qui seront stockées dans RelationalEntity. Elle a un nom, une liste d'attributs relationnels et une table qui correspond à sa table d'origine.

3.2.4 Procédure

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

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 chaînes de caractères mais elles pourraient, dans une future itération de Slyum, être améliorées en plusieurs objets plus complexes qui pourraient stocker la procédure indépendamment du SGBD visé.

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

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

3.2.5 Trigger, TriggerType et ActivationTime

Trigger correspond au Trigger du meta-schéma. TriggerType et ActivationTime sont respectivement le type de trigger (pour chaque ligne, pour chaque colonne) et le temps d'activation (après création, avant altération, etc.).

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

3.2.6 StoredProcedure

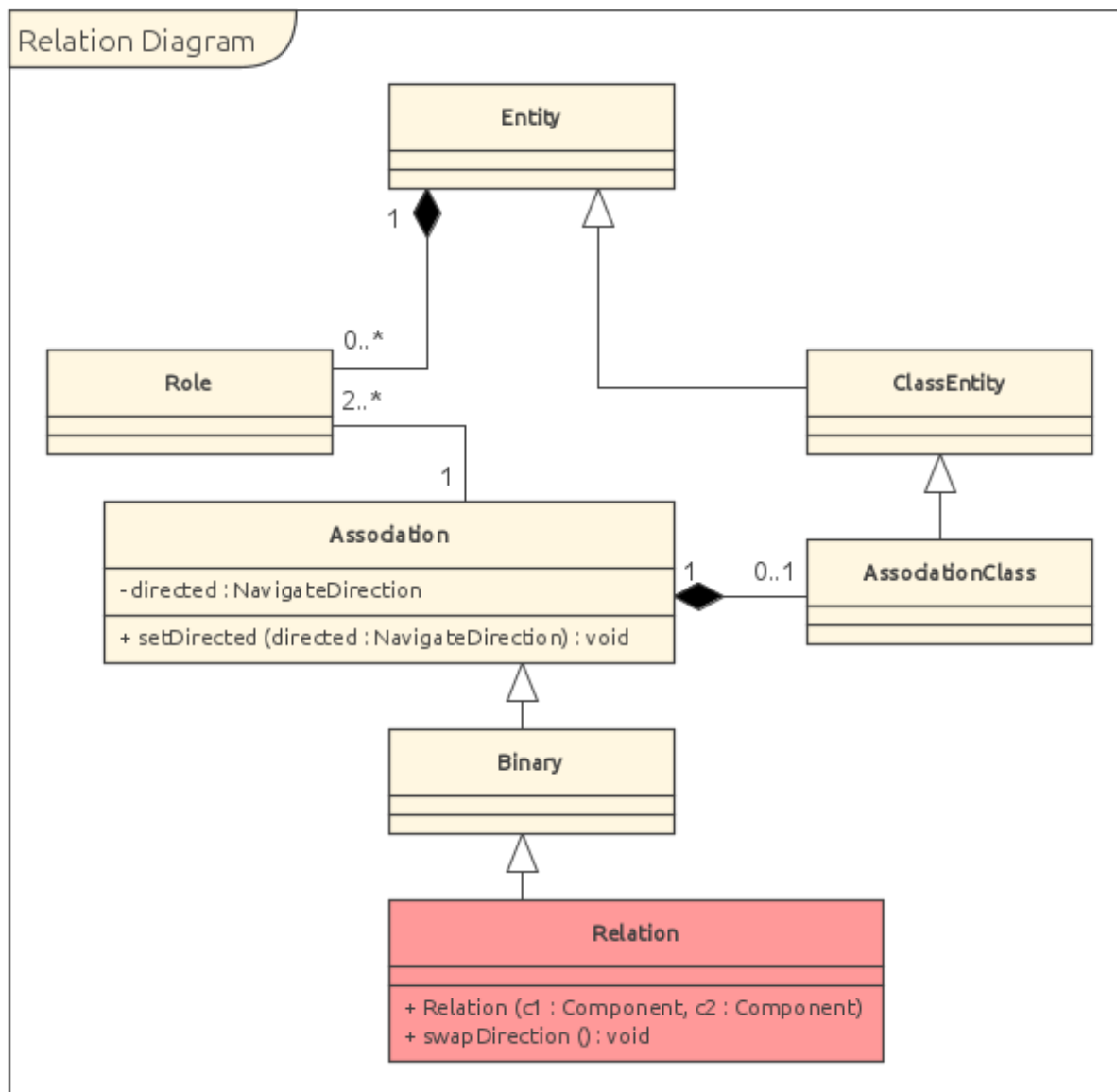
StoredProcedure correspondent à la procédure stockée du meta-schéma. Elle a un nom et une liste de paramètres.

3.2.7 View

View correspond à la vue du méta-schéma. Elle a un nom et une procédure stockée.

3.3 Diagramme de classe - relation

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



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 `swapDirection` permet de changer la "direction" du lien et ainsi changer quelle table recevra la clé étrangère.

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.

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 associations par cardinalités

4.2.1 Conversion des associations binaires 1 : 1

- On rajoute la clé étrangère dans une des 2 tables (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 0..1.

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

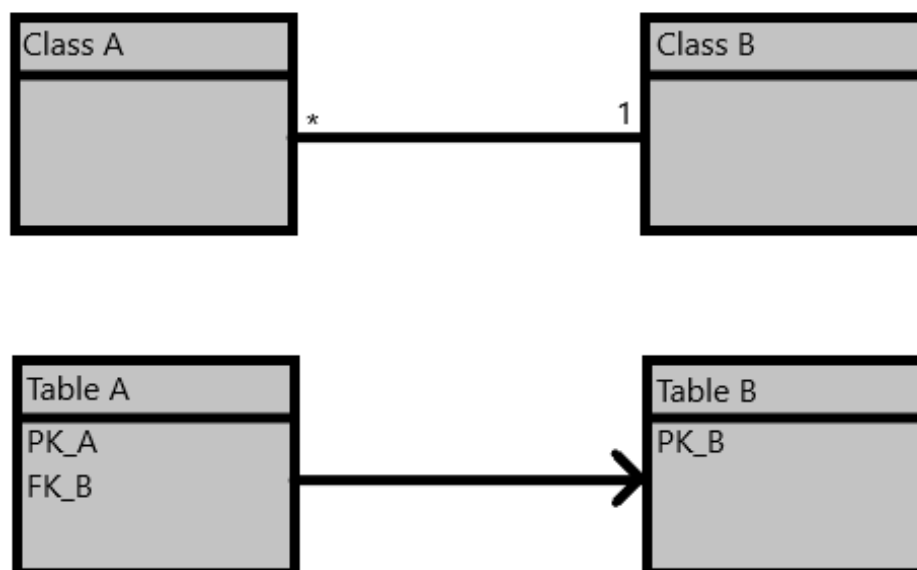
La conversion est similaire à la 1:1 à l'exception que la clé est unique et **nullable**.

4.2.4 Conversion des associations binaires 1 : N

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

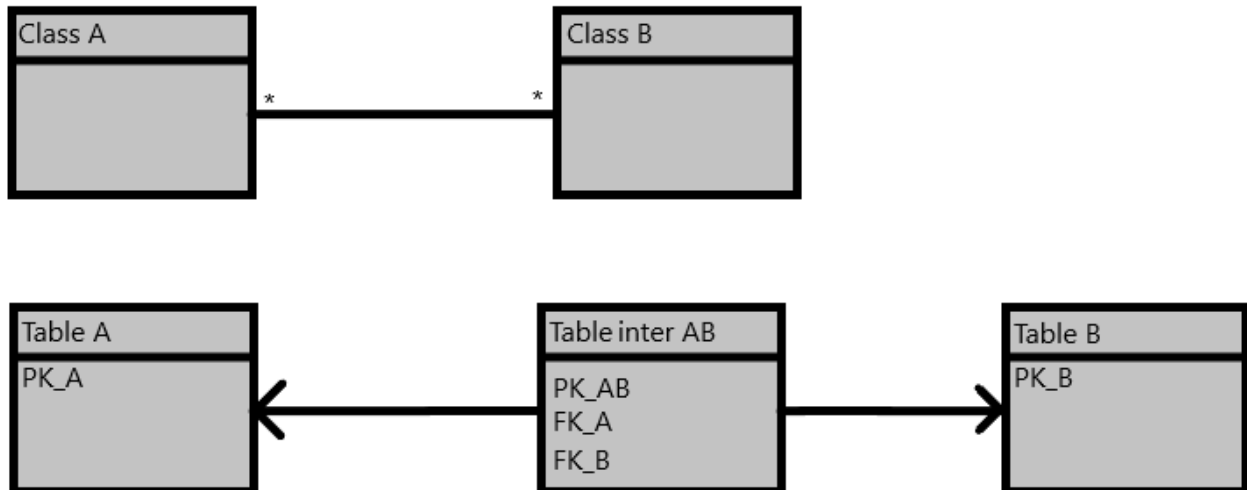
4.2.5 Conversion des associations binaires 0..1 : N

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



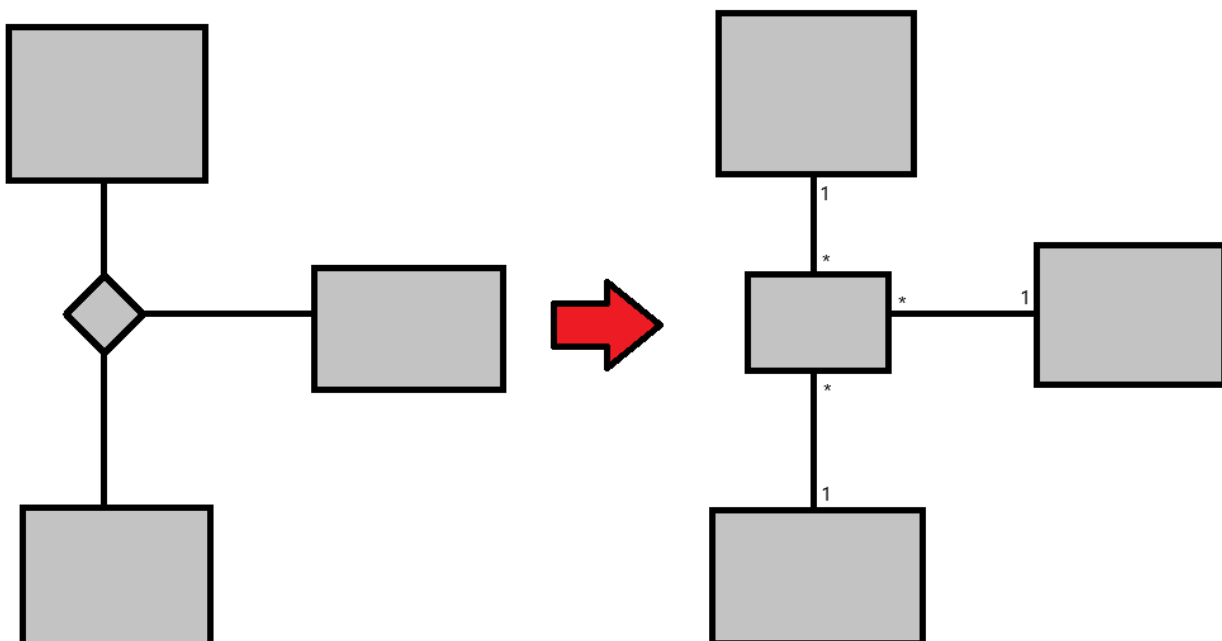
4.2.6 Conversion des associations binaires N : M

La conversion d'une association N:M se fait en créant une nouvelle table entre les 2. Cette table aura les clés des 2 primaires des tables de l'association comme clé étrangère et utilisera leurs composants comme clé primaires.



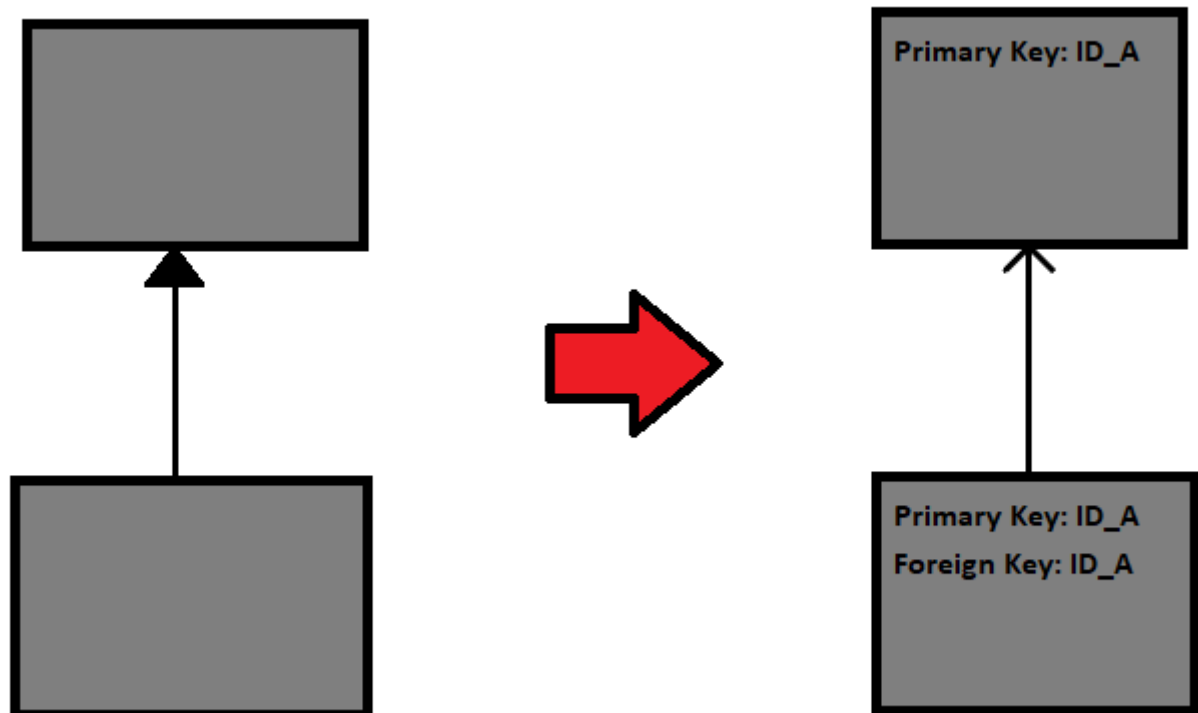
4.3 Conversion des associations multiples

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



4.4 Conversion des liens d'héritage

Les liens d'**héritage** sont convertis en **relation** avec la sous-classe ayant les mêmes composants que la clé primaire de 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 ([8.1.8](#)).

Le validateur analysera les différents éléments du schéma pour trouver les problèmes qui sont détectables par Slyum. Les règles de validation sont détaillées plus loin ([8.1.9](#)).

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	State
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

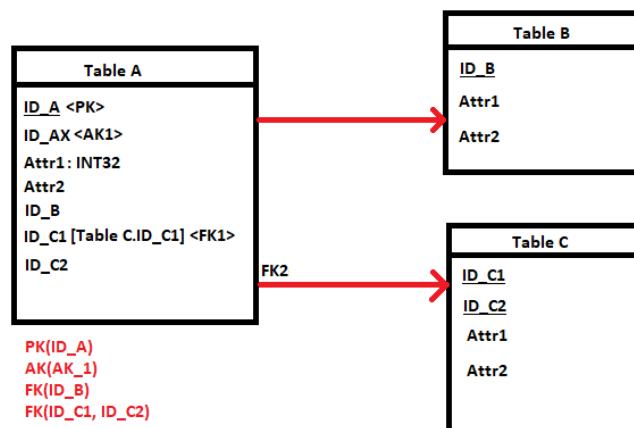
nom_clé_2

Attributs

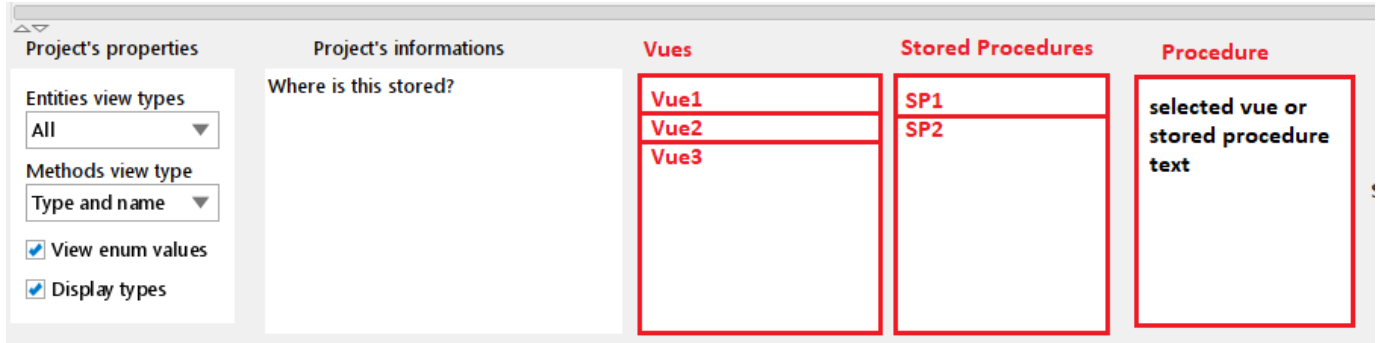
attribut_1

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 labeled '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.

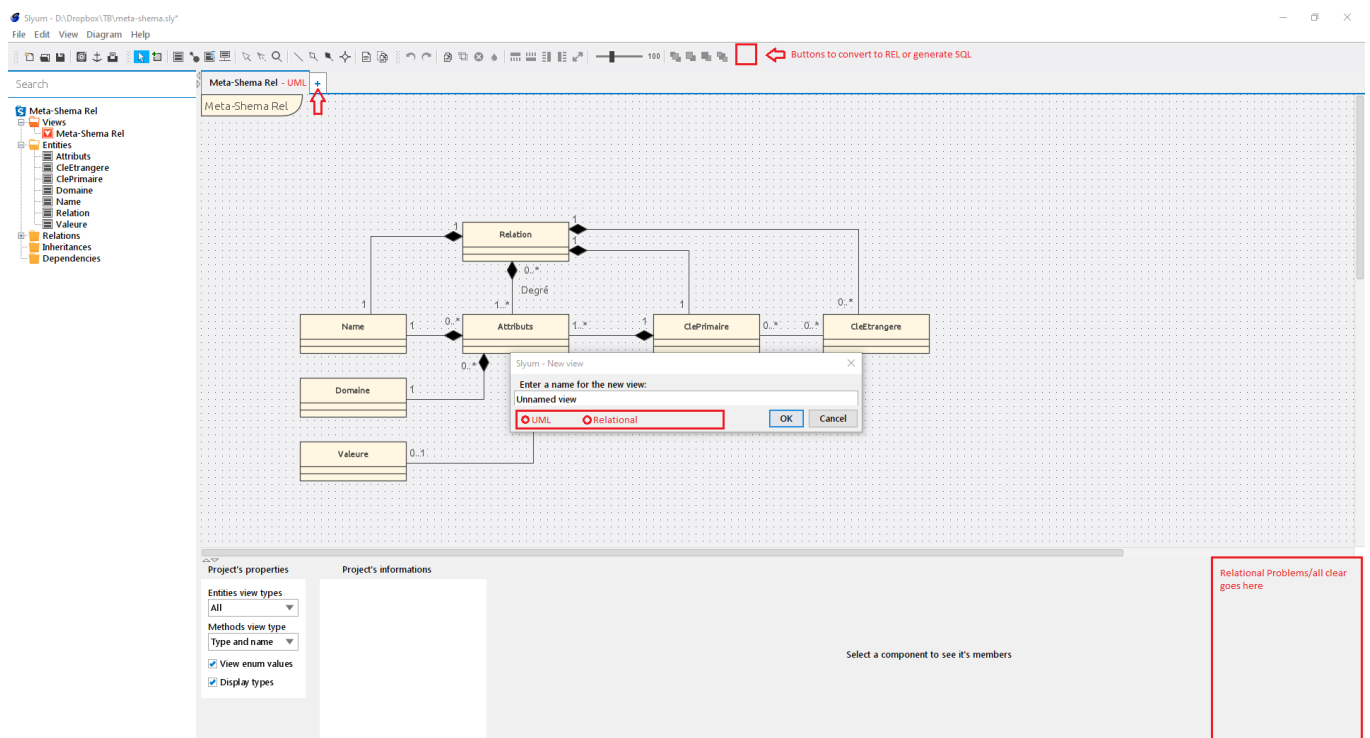
La partie contenant les méthodes de classes est changée pour pouvoir stocker des triggers. La partie montrant les paramètres de la méthode montre maintenant une zone de texte pour la procédure.

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 implémentées j'ai déplacé l'interface des vues dans sa propre partie.

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



Quand on crée une nouvelle vue, le choix d'un schéma uml ou relationnel ne change pas les classes utilisées mais détermine les fonctions utilisables sur la GUI (par exemple, 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.

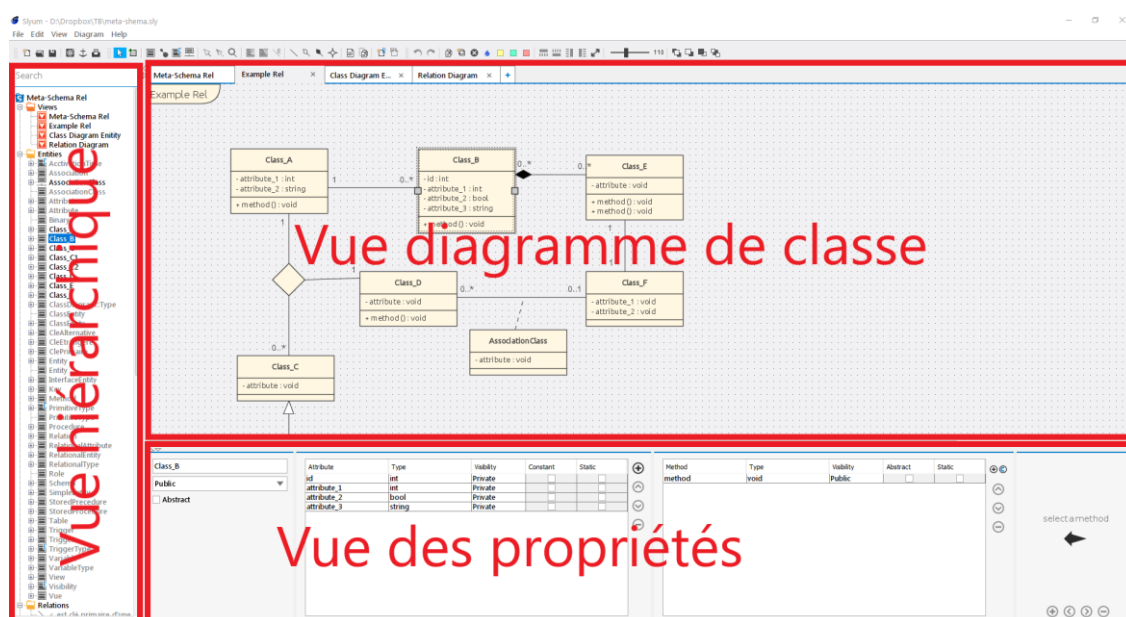
7 Structure générale de Slyum

Le projet original de Slyum est séparé en 4 parties.

La partie principale est la structure des classes. Elle contient toutes les entités des schémas (classes, enums, attributs, associations, etc.) ainsi que les entités relationnelles qui ont été rajoutées. Cette partie permet de stocker tous les composants des diagrammes de classes et des schémas relationnels.

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.

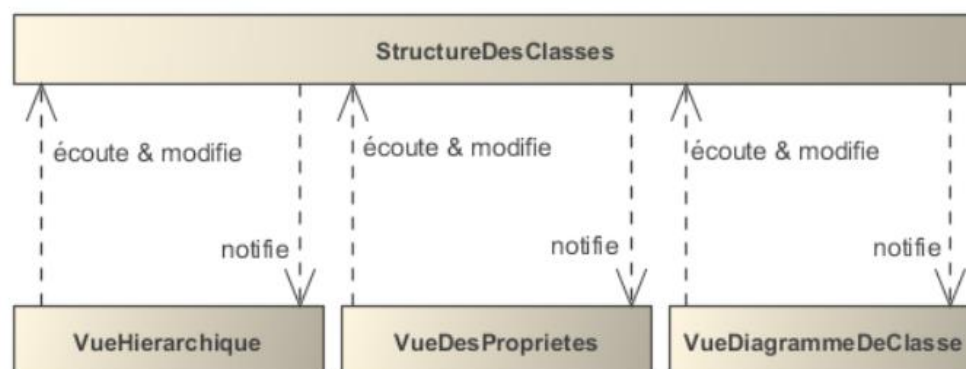
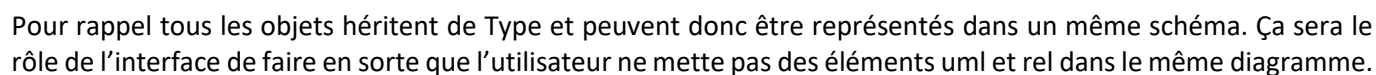


Illustration de la structure générale du projet du rapport Slyum de 2011 par David Miserez

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 existent mais j'en parlerai plus dans ma conclusion.

En explorant le code de Slyum, j’ai découvert l’existence d’une classe abstraite “simpleEntity” qui s’interpose entre la classe d’entité générique et les spécialisations UML (Classe, Enum, etc.) qui n’était pas présente dans le rapport original. J’ai dû, en conséquence, faire des modifications au diagramme de classe que j’avais fait pendant la modélisation. J’ai fait hériter mes composants relationnels à l’entité générique (Entity) et, ainsi, j’ai pu réutiliser une plus grande partie du code des entités de Slyum.

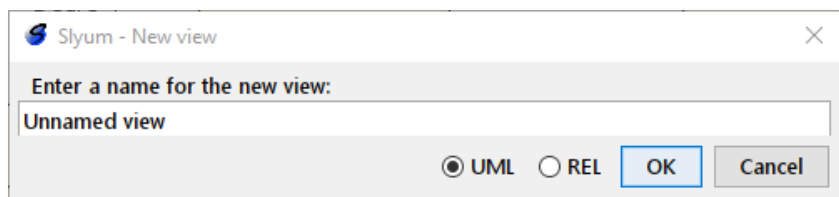


8.1 Vues graphiques

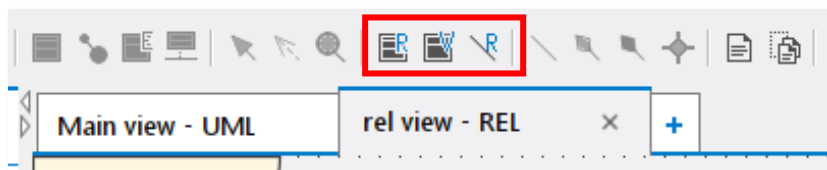
Pour chaque élément du diagramme relationnel il faut créer une classe de structure et trois autres classes pour chacune des trois vues graphiques de Slyum : GraphicView, Property et Node soit une entité visuelle, un panneau de contrôle et un nœud ajoutable dans l'arbre des entités. Certains éléments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

8.1.1 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.



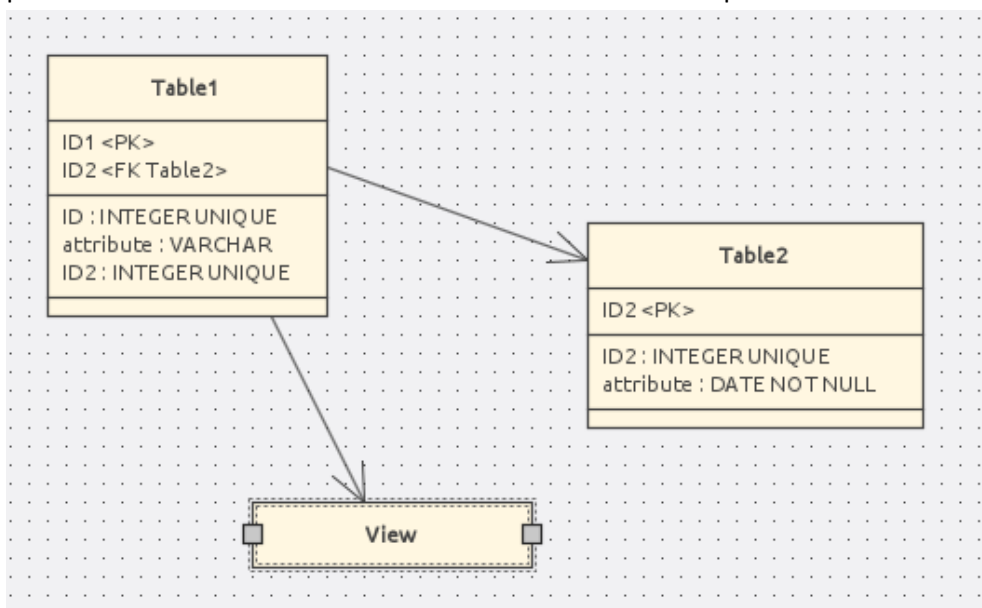
De plus, quand une nouvelle vue graphique (un onglet) est créée, un identifiant (-UML ou -REL) est rajouté à l'onglet pour identifier son type. Celui-ci peut faire partie du nom de la vue et peut du coup être enlevé si l'utilisateur le désire.

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.

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, sa table d'origine est aussi notée. La liaison entre table et vue est purement visuelle.

Pour savoir quels attributs font partie de ces clés, il faut regarder dans la vue des propriétés.



8.1.2 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 des diagrammes UML sont mélangées à ceux des schémas relationnels. 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 de diagrammes UML sont grisés.

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

La nomenclature des clés est la même que dans la vue du diagramme.

8.1.3 Sauvegarde

Slyum sauve ses projets sous format XML. La racine du document est la balise "classDiagram". Cette balise contient d'abord la balise "diagramElements" qui contient la structure des diagrammes (classes, tables, associations, etc) ensuite une balise "umlView" par vue graphique (onglet) qui contient les informations nécessaires au positionnement et la taille de chaque élément. Plus d'informations sur le modèle de sauvegarde de Slyum UML se trouvent dans le chapitre 8 du rapport de [l'ancien travail](#).

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.

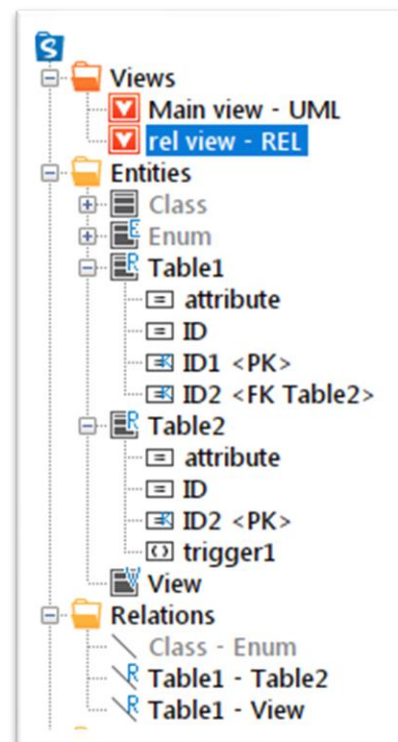
Ceci veut dire qu'il est possible de lire les fichiers de sauvegarde de la version précédente avec la nouvelle version et vice-versa mais les nouveaux éléments ne seront pas parsés par le parseur XML de la version précédente.

```

<entity entityType="TABLE" id="54443" name="Plane">
  <relationalAttribute const="false" defaultValue="" id="54449" name="model_number" notNull="false" type="INT" unique="true"/>
  <relationalAttribute const="false" defaultValue="" id="54476" name="brand" notNull="false" type="VARCHAR" unique="false"/>
  <key id="54443" keyType="primary" name="ID">
    <raID id="54449"/>
  </key>
</entity>
<entity entityType="TABLE" id="54503" name="Pilot">
  <relationalAttribute const="false" defaultValue="" id="54513" name="employee_id" notNull="false" type="INT" unique="true"/>
  <relationalAttribute const="false" defaultValue="" id="54548" name="name" notNull="false" type="VARCHAR" unique="false"/>
  <key id="54503" keyType="primary" name="ID">
    <raID id="54513"/>
  </key>
</entity>
<entity entityType="TABLE" id="54575" name="Flight path">
  <relationalAttribute const="false" defaultValue="" id="54585" name="flight_id" notNull="false" type="INT" unique="true"/>
  <relationalAttribute const="false" defaultValue="" id="54620" name="departure" notNull="false" type="VARCHAR" unique="false"/>
  <relationalAttribute const="false" defaultValue="" id="54655" name="arrival" notNull="false" type="VARCHAR" unique="false"/>
  <relationalAttribute const="false" defaultValue="" id="54690" name="duration" notNull="false" type="INT" unique="false"/>
  <key id="54575" keyType="primary" name="ID">
    <raID id="54585"/>
  </key>
</entity>

```

Extrait de fichier de sauvegarde avec des entités relationnelles



Les nouvelles balises sont les suivantes :

8.1.3.1 Tables et vues

Les tables et les vues n'ont pas de balises propres à elles. Comme les classes et enums elles partagent la balise "entity". Cette balise a un attribut "entityType" qui permet de différencier les différentes entités et j'ai rajouté le type "TABLE" et "VIEW". Le seul attribut rajouté à la balise "entity" est "procédure" si cette entité est une vue.

8.1.3.2 Attribut relationnel

La balises "relationalAttribute" contiennent les attributs suivants :

- Name : le nom de l'attribut
- Id : l'id de l'attribut
- Type : le type de l'attribut
- notNull : si l'attribut est nullable
- unique : si l'attribut est unique
- const et defaultValue : attributs de la classe parent non-utilisé par les attributs relationnels

8.1.3.3 Trigger

La balise "trigger" contient les attributs suivants :

- Name : le nom du trigger
- Procedure : la procédure du trigger
- activationTime : le temps d'activation du trigger
- triggerType : le type du trigger

8.1.3.4 Clés

La balise "Key" contient les attributs suivants :

- Id : l'id de la clé
- Name : le nom de la clé
- keyType : détermine si une clé est primaire ou alternative. (Les clés étrangères sont gérées par les liens relationnels)

Dans les balises "Key" on trouve aussi les balises "raID" qui ont comme attribut les ID des composants "relationalAttribute" de la clé.

8.1.3.5 Liens relationnels

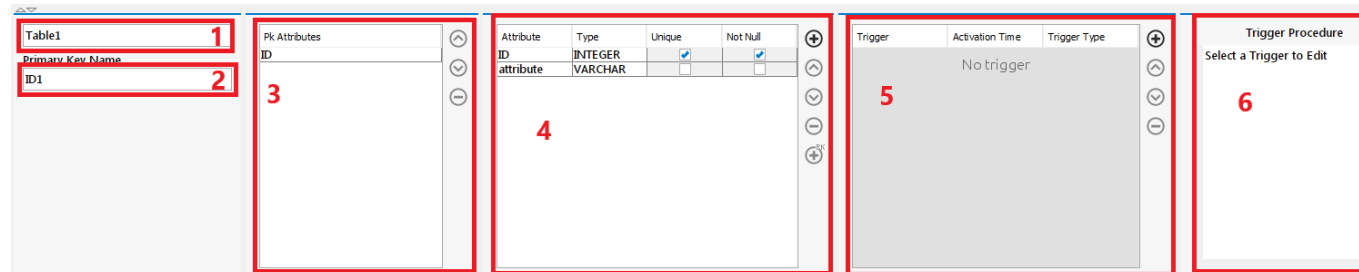
Les liens relationnels utilisent les mêmes balises que toutes les associations avec le tag "REL" pour les différencier. Ces liens sont utilisés pour rajouter les clés étrangères aux tables.

```
<association aggregation="REL" direction="FIRST_TO_SECOND" id="54866" name="">
  <role componentId="54789" name="" visibility="PRIVATE">
    <multiplicity>
      <min>1</min>
      <max>1</max>
    </multiplicity>
  </role>
  <role componentId="54717" name="" visibility="PRIVATE">
    <multiplicity>
      <min>1</min>
      <max>1</max>
    </multiplicity>
  </role>
</association>
```

8.1.4 Vue des propriétés

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

8.1.4.1 Propriétés des tables

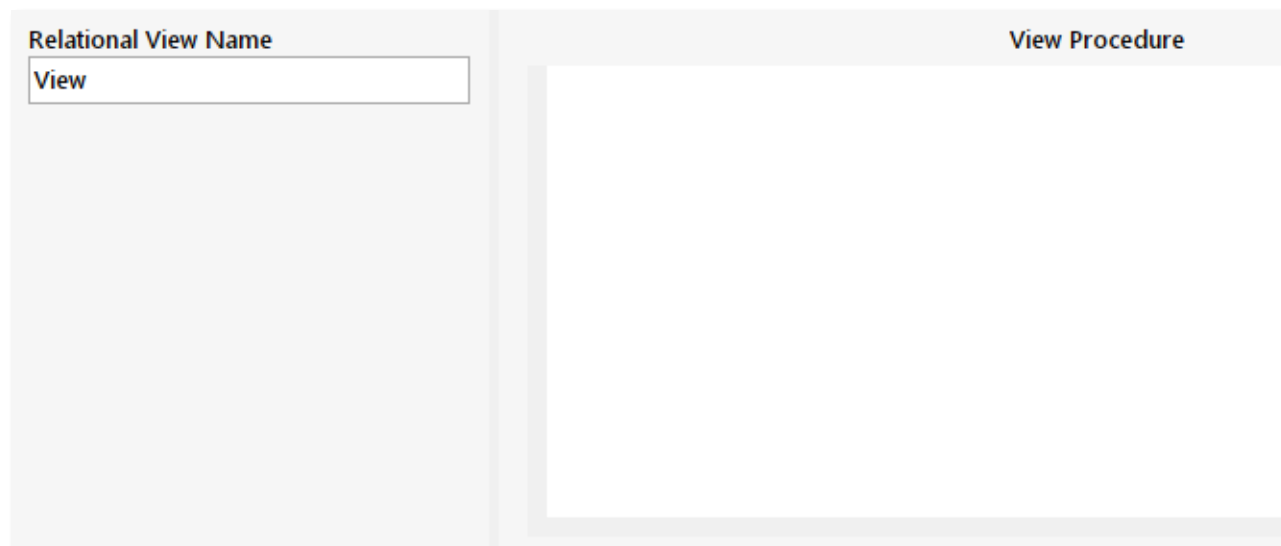


The screenshot shows the 'Tablet' properties window. It is divided into several sections:

- 1**: The 'Tablet' name field at the top left.
- 2**: The 'Primary Key Name' field below the table name.
- 3**: The 'Pk Attributes' list on the left side.
- 4**: The main table of attributes with columns: Attribute, Type, Unique, and Not Null.
- 5**: The 'Triggers' section on the right, showing 'No trigger'.
- 6**: The 'Trigger Procedure' section on the far right, 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

8.1.4.2 Propriétés des vues



The screenshot shows the 'View' properties window. It is divided into two main sections:

- Relational View Name**: A section on the left containing a text field with the value 'View'.
- View Procedure**: A large, empty text area on the right 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.

8.1.4.3 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.

8.1.5 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 pour 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.

Enter the relation's name

☒ Table1 -> Table2

☐ Table2 -> Table1

8.1.6 Procédures stockées

Les procédures stockées ne font pas partie du schéma relationnel au sens propre mais elles sont souvent ajoutées dans un 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 à part quelques soucis d'interface et de conversion SQL, je ne pense pas que leur absence soit une grande perte pour ce projet.

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

Vu que je n'avais pas originalement considéré le refactoring comme quelque chose qui devrait être documenté dans le rapport, je n'ai pas pris de notes concrètes sur ce que j'ai modifié. J'ai pu retrouver des traces de mon refactoring en regardant les commits de mon git. Voici quelques exemples de refactoring que j'ai fait :

Classe "Slyum"

- Changer des appels de méthode "valueOf()" en "parse__()".
- Changer des fonctions lambdas en références méthodes.
- Remplacer la création d'une liste dans un appel méthode par la liste de ces éléments.

```
PanelClassDiagram.getInstance().exportAsVectoriel("pdf", new String[] {"pdf", "svg", "eps"});  
PanelClassDiagram.getInstance().exportAsVectoriel("pdf", "pdf", "svg", "eps");
```

Classe "GraphicView"

- Enlever des imports non-utilisés.
- Enlever des appels à "stream()" pour parcourir des listes.
- Rendre des variables finales quand elles peuvent l'être.
- Changer des fonctions lambdas en références méthodes.

```
- components.stream().forEach((c) -> { c.userDelete();});  
+ components.forEach(GraphicComponent::userDelete);
```

- Changer des appels de méthode "valueOf()" en parse__().
- Remplacer des "new runnable()" par des méthodes lambda.
- Enlever des variables temporaires/intermédiaires inutiles.
- Enlever la répétition des types dans les créations des Listes.

```
- final LinkedList<EntityView> selectedEntities = new LinkedList<EntityView>();  
+ final LinkedList<EntityView> selectedEntities = new LinkedList<>();
```

- Remplacer des classes anonymes par des expressions lambda quand c'est possible.
- Enlever des clauses dans les "if" qui sont toujours vraies/fausses.

Classe "SimpleEntityView"

- Remplacer des imports individuels par des imports de package regroupé (ex : java.awt.*)
- Enlever des castes non nécessaires

Interface "IComponentObserver"

- Enlever le "public" des méthodes de l'interface

Les classes suivantes ont eu des changements similaires aux exemples précédents :

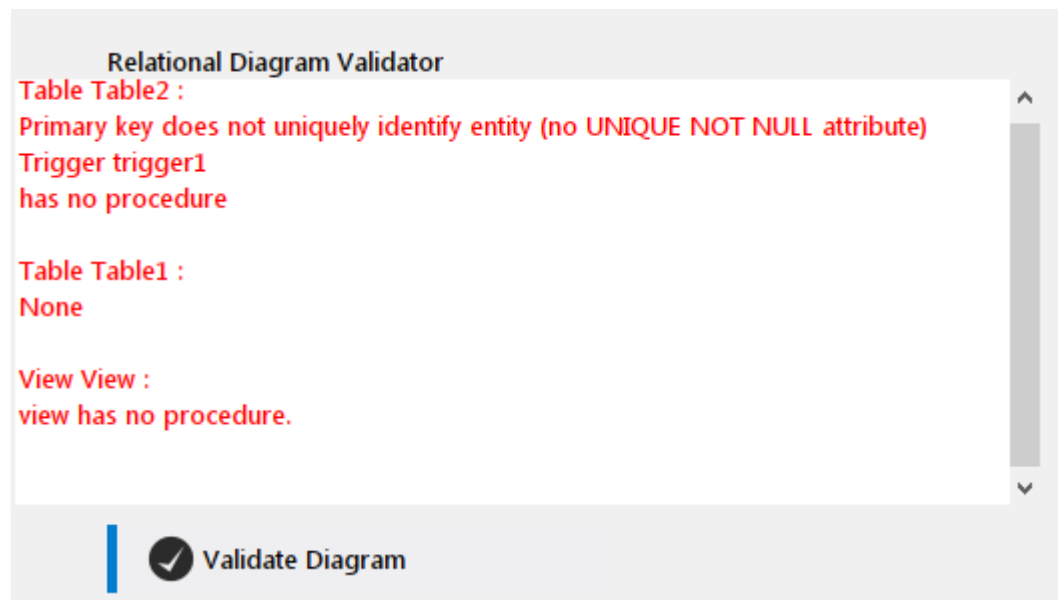
- STab
- EntityView
- EnumView
- GraphicView
- EnumEntityProperties
- GlobalProperties
- NoteProperties
- RelationProperties
- SimpleEntityProperties
- SPanelDiagramComponent
- PanelClassDiagram
- XMLParser
- Utility

Cette liste contient la majorité des classes qui ont été impactées par mon refactoring. Il y en a encore quelques-unes qui ont soit peu voir un seul changement ou des changements non conséquents (ré-indentation, renommage, fautes d'anglais dans le code et dans les commentaires, etc.).

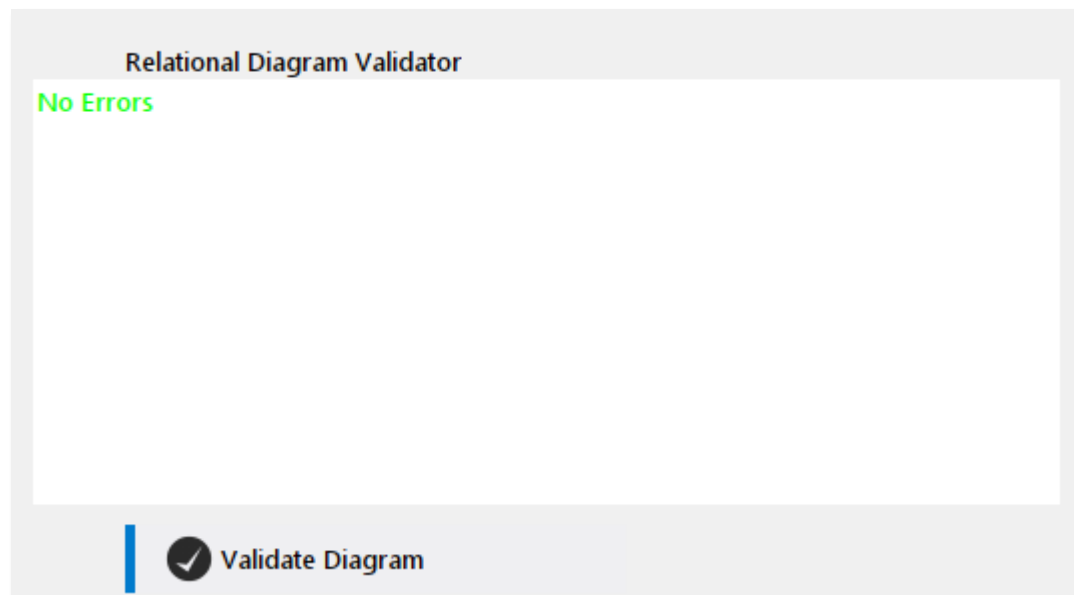
J'ai corrigé ces erreurs quand je pouvais les corriger sans trop de problèmes mais il en reste encore beaucoup. Du coup, une passe de correction et de refactoring du code pourrait s'avérer nécessaire si on voulait rajouter de nouvelles fonctionnalités ou faire des changements sur celles déjà présentes.

8.1.8 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.

8.1.9 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 qui contient des cycles n'est pas nécessairement faux mais peut l'être si la sémantique est incorrecte. Le validateur peut détecter si un est présent mais pas si celui-ci représente une erreur de sémantique. Du coup, quand un cycle est détecté, l'utilisateur est averti de la présence d'un cycle mais celui-ci ne compte pas comme une erreur. C'est à l'utilisateur de vérifier et déterminer si le cycle est voulu ou non. 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 les détecter.

Les règles ci-dessus 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 encore, les types des attributs ne sont pas garantis d'exister dans SQL).

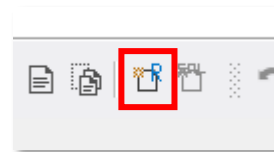
8.2 Améliorations possibles

Voici une liste non-exhaustive de potentielles améliorations qui pourraient être apporté au créateur de schémas 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 transformées en un scripte SQL adapté au 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 pouvoir choisir qu'entre des types d'attributs définis, 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

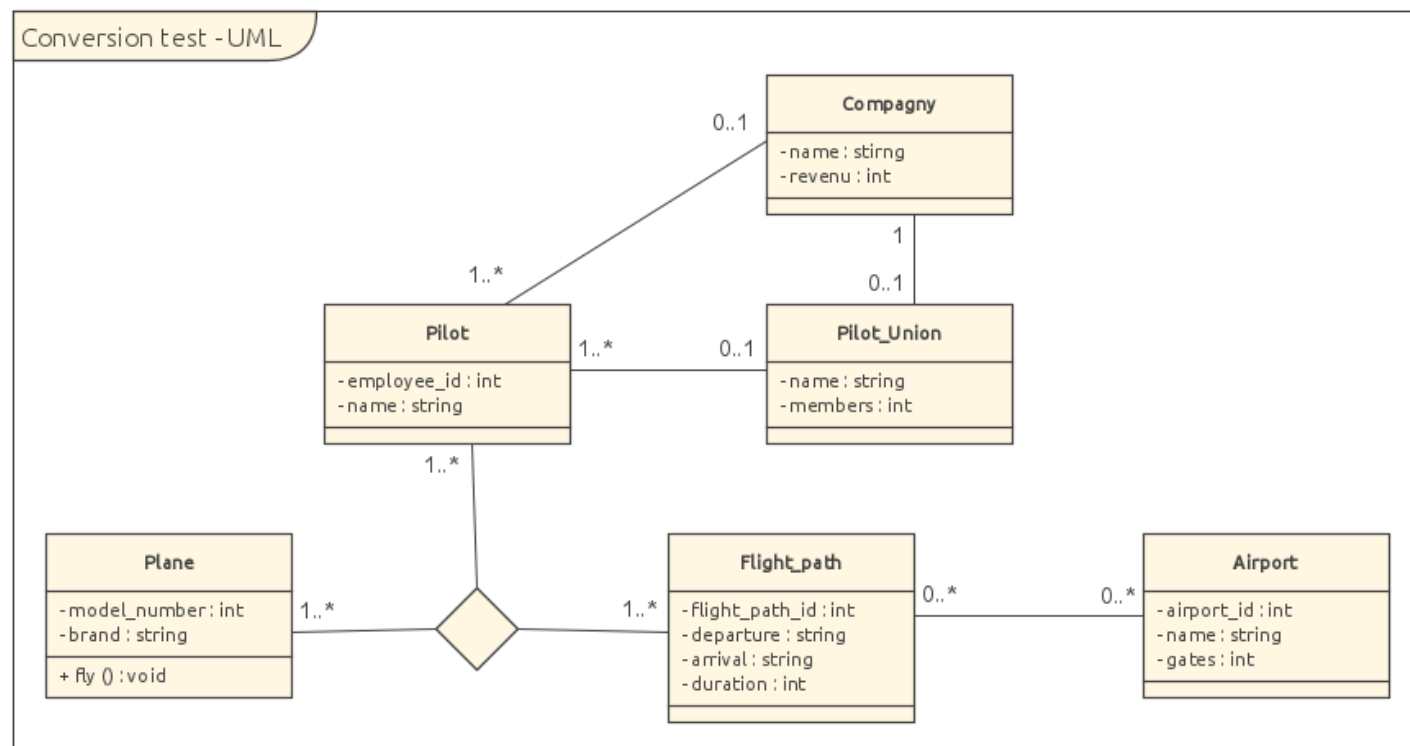


Diagramme UML à être converti

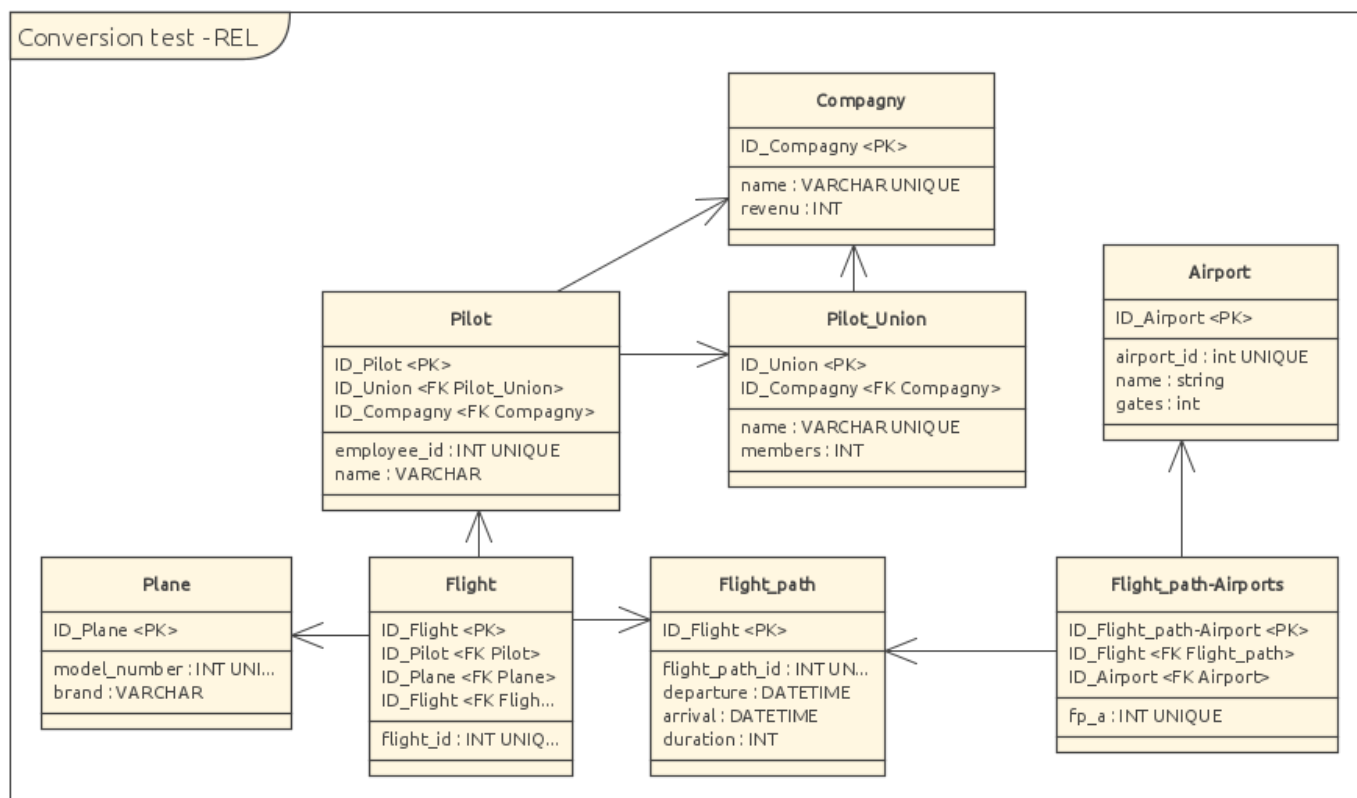


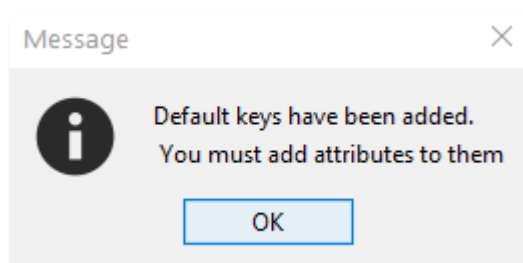
Diagramme UML converti en Schéma relationnel

On peut voir ici comment le convertisseur rajoute des tables pour l'associations multiple et l'association "0..*-0..*" entre "Flight_path" et "Airport". Ces tables ont été redispesées pour être plus lisibles.

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".



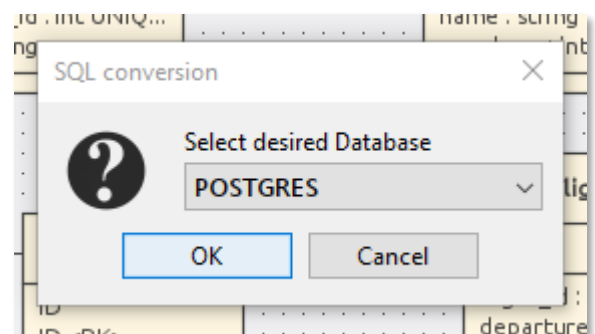
L'algorithme convertit les tables et vues une à une et les rajoute dans une chaîne de caractères qui sera ensuite écrite dans un fichier ".sql".

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 le SGBD open-source la plus utilisée et PostgreSQL car c'est le SGBD de prédilection du professeur qui a proposé ce projet.

10.1 Utilisation

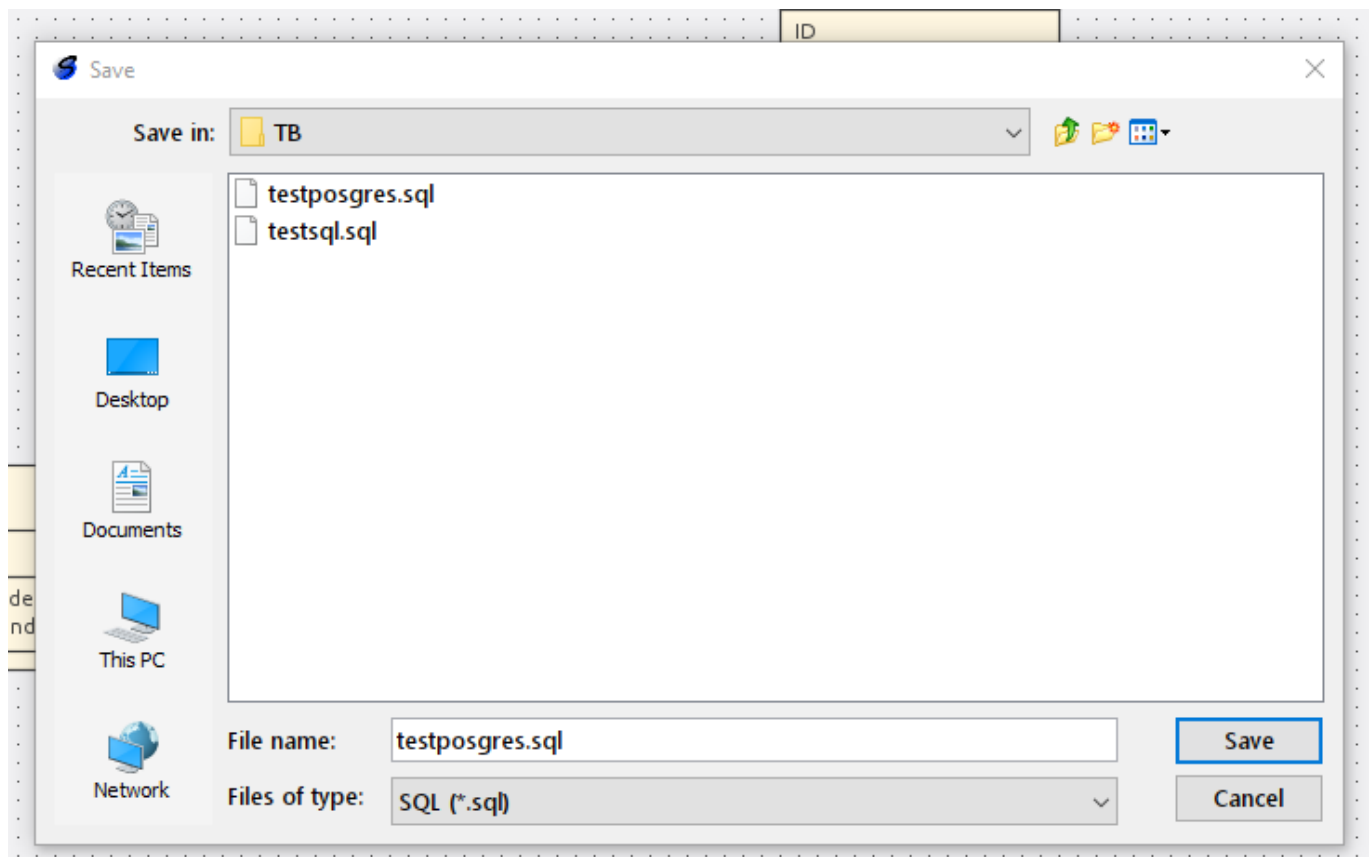
Le générateur SQL est une classe singleton nommé SQLConverter qui se situe dans le package "utility.SQLConverter". Elle prend une vue graphique REL comme attribut, extrait les classes et les vues de celle-ci et les convertit en un scripte SQL qui peut être utilisé pour créer une base de données.

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 le SGBD pour laquelle il veut créer le scripte.



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

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



10.2 Implémentation

La conversion en scripte SQL utilise un constructeur de String (StringBuilder). Cette classe de la librairie Java me permet de construire une chaîne de caractères au fil de la conversion sans avoir besoin de constamment concaténer des chaînes de caractères.

La conversion se fait par étape. La première étape est de choisir le SGBD visée. Si le SGBD n'est pas reconnu, soit parce qu'elle n'a pas été implémentée ou qu'un faux paramètre a été passé à la méthode (ce qui ne devrait pas être possible s'il a été choisi par l'interface), la conversion s'arrête et l'utilisateur est averti que le SGBD n'est pas implémenté. Sinon la méthode de conversion du SGBD est lancée.

Chacune de ces méthodes s'occupe de générer les éléments du langage pour créer la base de données et va ensuite analyser les tables et les vues du schéma relationnel et lancer les méthodes de génération pour chaque méthode.

Vu que les vues devraient déjà contenir une procédure correctement formatée, la méthode de la conversion a besoin uniquement de la rajouter dans le squelette du code de la vue.

Les tables par contre ont plusieurs éléments qui sont rajoutés séquentiellement via leurs propres méthodes de conversion. Les éléments sont rajoutés dans cet ordre :

- Attributs de la table
- Attributs des clés étrangères
- Clé primaire
- Clés alternatives
- Clés étrangères

Le code des triggers se situant à l'extérieur des tables en SQL, sa méthode de génération est appelée depuis la méthode de conversion du SGBD et non depuis celle de la table.

Les attributs de la table et ceux des clés étrangères sont traités séparément car le nom de ces attributs est modifié pour éviter d'avoir deux attributs qui ont le même nom dans une table. Par exemple, si deux tables ont une clé primaire défini par un attribut "ID" et qu'une relation est formée entre les deux on se retrouve avec une table qui a deux attributs "ID" du coup, pour éviter cela, le nom de la table d'origine est ajouté au début de l'attribut. Malgré cela, si deux attributs ont le même nom dans une table le validateur relationnel informe l'utilisateur pour lui permettre de changer cela.

Malgré le fait que l'implémentation de SQL dans les différents SGBD diffère suffisamment qu'il n'est pas possible de créer un set de fonction qui puisse générer du code pour tous les SGBD, certains éléments restent suffisamment similaires que j'ai pu créer une fonction qui marche pour tous les SGBD implémentés. En effet, les attributs des tables et les composants des clés sont suffisamment génériques que j'ai pu créer une fonction pour chacun de ces éléments qui fonctionne pour les SGBD implémentés.

```
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(" (\n");

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

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

    //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()) {
        sb.append(" ");
    }
}
```

Extrait de code de génération de table pour mySQL

```
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)
);
```

Exemple de script de la table Compagny

11 Conclusion

11.1 Problèmes connus

Les clés étrangères ne montrent pas si elles sont nullable ou non. Un oubli à la modélisation m'a fait passer par-dessus cette propriété et d'ici à ce que je réalise son absence, il était trop tard pour la rajouter proprement.

Les attributs des clés étrangères ne sont pas rajoutés à la table.

Il est possible de placer des éléments de diagramme 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, permettent toujours d'en placer à des endroits non-désirés.

11.2 Améliorations possibles

Il y a une pléthore d'améliorations possibles comme celles que j'ai proposées 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 apportés 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. De plus, Slyum est construit en swing, un Framework graphique que je n'avais pas utilisé depuis un moment et qui m'a pris du temps à me réhabituer. J'ai aussi pris beaucoup plus de temps que prévu à me réhabituer à l'implémentation du patron de conception Observer de Java utilisé ici. Malgré cela j'ai persévéré et pour finir j'ai réussi à implémenter les 3 fonctionnalités majeures que j'avais prévues.

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 peut 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 SGBD MySQL et PostgreSQL.

Certaines fonctionnalités mineures ont été coupées 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'aurais préféré avoir plus de temps pour le peaufiner un peu plus. J'étais parti dans

ce projet avec l'intention d'intégrer un éditeur de schéma relationnel dans Slyum en gardant l'apparence des ajouts aussi similaire que possible à ce qui est déjà présent et, même si cela a été, pour moi, une des grandes difficultés de ce projet, Je suis arrivé à un résultat avec lequel je suis heureux.

12 Annexes

Exécutable Jar : https://github.com/yoann0000/slyum/tree/relational_model/bin/relational%20release

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

