Application de rétro-ingénierie pour la construction de schémas Entité/Association à partir d'un modèle relationnel

DECOURSELLE Joffrey

15 février 2013

Résumé

Dans le cadre de travaux de recherche sur la modélisation de données stockées et partagées dans des environnements dynamiques et la formation Master en informatique de l'Université Claude Bernard Lyon 1, ce document présente l'étude d'une solution concrète de rétro-conception d'une base de données. Pour ce projet, elle consiste en la recherche d'une solution d'extraction de métadonnées depuis un modèle relationnel et la traduction de ces informations dans un schéma entité/association. Pour expliquer cela, un état de l'art et une étude du contexte précéderont une proposition d'implémentation d'une solution et son évaluation.

Ce travail effectué durant un mois et demi, est encadré par M. Lumineau et M. Duchateau, maîtres de conférence au LIRIS, Laboratoire d'InfoRmatique en Image et Systèmes d'information.

Mots clés : base de données, rétro-conception, entité/association, modèle relationnel

Université Claude Bernard Lyon 1
Département Informatique
Domaine de la Doua - 23-25 Avenue Pierre de Coubertin -Villeurbanne

1 Introduction

1.1 Contexte

La rétro-ingénierie, aussi appelée rétro-conception, consiste à déterminer le mode de fonctionnement d'un objet ou d'un modèle en production. Dans le cadre des bases de données, il permet de traduire un modèle logique en un modèle conceptuel. Ce dernier permet ainsi de simplifier un raisonnement sur diverses structures de données en leur apportant un nouveau formalisme.
Ce procédé répond aux besoins liés à l'obtention d'informations lors de divers processus de conception ou de maintenance de systèmes informatiques.
Dans le cas d'une application qui nécessiterait l'intégration de plusieurs sources de données hétérogènes, la récupération d'informations sur ces sources nécessite bien souvent d'effectuer un processus de rétro-conception. Cela s'applique également pour un système d'information qui aurait été modifié et dont la documentation nécessiterait une mise à jour.

1.2 Présentation de la démarche

Dans le but de trouver une solution correspondant au sujet, la première phase du projet a consisté en l'étude des travaux connexes. Cette dernière étant limitée dans le temps, elle s'est concentrée principalement sur des propositions pragmatiques de processus de rétro-conception trouvées dans la littérature et sur des implémentations concrètes open-source trouvées sur Internet. Pour cette seconde partie, le constat principal que l'on peut mettre en avant est qu'aucune solution acceptable n'a pu être trouvée dans le temps imparti pour les recherches. C'est pourquoi la seconde phase du projet peut se définir par l'étude d'approches formelles de rétro-conception et la mise en place d'une solution répondant aux besoins. Ces phases ont été ponctuées d'étapes de rédaction d'un cahier des charges, d'un état de l'art ou de ce document et d'étapes de lecture et de développement.

1.3 Problématique

La connexion à une base de données et l'exploitation de sa structure sont des manipulations courantes effectuées aussi bien par des experts que par des étudiants. L'élaboration d'une telle base est également un processus simplifié grâce à de nombreux outils informatiques. La conception d'une base de données précède bien souvent son exploitation. L'objectif de ce travail de recherche est de trouver une méthode pour effectuer l'opération inverse i.e. obtenir la version conceptuelle d'une base de données à partir de ses identifiants de connexion.

1.4 Contributions

Les fruits du travail autour de ce projet ont permis l'élaboration des livrables suivants : un cahier des charges, un état de l'art pratique des solutions existantes, un algorithme de rétro-conception d'un modèle relationnel, une implémentation de ce dernier dans un environnement web et ce rapport. Ces contributions ont été élaborées dans un cadre précis et structuré par les enseignants en charge du projet.

2 État de l'art

2.1 Recherches bibliographiques

L'objectif de cette partie des recherches a été d'améliorer progressivement la compréhension du sujet. Ainsi, les étapes d'un processus de rétro-conception ont été affinées tout au long des recherches.
M. Hainaut a étudié le sujet et proposé un nombre significatif d'écrits sur les différentes facettes du domaine. Il propose de commencer par étudier les étapes de la conception d'une base de données i.e. de sa mise en production depuis son modèle conceptuel. En effet la bonne compréhension de ce processus est indispensable à la mise en place d'une rétro-ingénierie [1].
Dans un autre de ses écrits, il définit les trois phases clés de ce processus. La phase "conceptuelle" comprend la collecte des besoins du client et l'élaboration d'un premier schéma. La phase "logique" consiste à élaborer et optimiser un modèle relationnel sans prendre en compte le SGBD [1]. La phase "physique" consiste enfin en l'élaboration d'un modèle logique comprenant les contraintes de l'application demandée et celles du SGBD. Le modèle est alors transcrit via un DDL [2] [3].

En s'appuyant sur les écrits [2] et [3] on peut inverser ces étapes et aborder le problème en trois phases relatives à la rétro-conception : l'extraction des métadonnées [3] pour la création d'un modèle relationnel, la conceptualisation du modèle relationnel et enfin la normalisation et le raffinement du modèle conceptuel.

Même si on peut construire un plan pour établir une rétro-ingénierie, Hainaut rappelle que ce type de processus ne peut être entièrement automatisé par un simple algorithme. En effet, ce travail est un ensemble de manipulations et de décisions non déterministes. Reconstruire un schéma conceptuel ne repose pas uniquement sur la simple analyse du code d'une base de données [1].

La dimension sémantique d'un modèle relationnel que l'on souhaite conceptualiser est à prendre en compte. Les écrits [4], [3] et [1] définissent la notion de "dégradation sémantique". C'est le fait de perdre des informations au cours des différentes étapes de la mise en production d'une base de données. En accord avec la vision de M. Hainaut, ces informations ne peuvent pas toujours être retrouvées par un algorithme [1]. Ainsi, toute implémentation automatisée d'un processus de rétro-ingénierie ne peut être qu'un appui pour un expert et non l'assurance d'obtenir à chaque fois une solution acceptable.

La conceptualisation d'un modèle relationnel pour la création d'un schéma entité/association est abordée de différentes manières dans les écrits. En effet, les approches des différents auteurs cités dans ce document s'accordent sur le fait qu'elle ne consiste pas seulement à convertir les tables du modèle relationnel en entités et les clés étrangères en associations.

L'approche de [2] consiste à faire ressortir toutes les contraintes référentielles du modèle logique en analysant ses dépendances fonctionnelles. On obtient alors une liste de dépendances d'inclusion qui permet ensuite de créer un prototype de modèle relationnel en ayant découvert tous les objets nécessaires à la vue conceptuelle et leurs dépendances. Après un travail sur son modèle relationnel, M. Petit, l'auteur de [2], propose un prototype d'algorithme permettant de construire les entités et associations du schéma attendu. Cette approche pragmatique en deux temps permet de découvrir

1. Système de gestion de base de données.
2. Data Description Language.
3. Données structurelles d'un système.

tous les objets et dépendances que l'on ne verrait pas forcément à première vue sur un modèle relationnel.

Une autre approche détaillée dans les écrits [1], [5] et [6], propose une série de manipulations et de règles à appliquer pour obtenir de manière incrémentale un schéma entité/association. La première étape de ces approches est d'obtenir le schéma relationnel de la base de donnée en analysant les vues proposées par le SGBD ainsi que le DMS [4] issu des applications associées.

La pertinence du modèle conceptuel obtenu dépend ensuite des règles appliquées au modèle relationnel. M. Andersson, auteur de [5] propose trois séries de règles qui permettent respectivement de construire les entités, les associations et de raffiner le schéma. Dans [4] on soulève l'éventualité d'entités cachées derrière des attributs. L'étude des instances liées à ces attributs est nécessaire pour identifier s'il s'agit d'un objet qui ne possède pas de table dans la base de données.
Dans la méthode de Hainaut on retrouve ce type de processus. Une série de règles de traduction précède des propositions de manipulation pour la normalisation du schéma entité/association [1]. Cette approche offre une grande souplesse pour une éventuelle implémentation et permet la confrontation de plusieurs versions d'un algorithme en faisant varier les règles.

La qualité du résultat obtenu peut être très variable. M. Petit dans [2] propose, en plus de son algorithme, une méthode de normalisation du schéma entité/association afin de s'approcher encore plus de la vue conceptuelle attendue. Une telle proposition est également faite par Hainaut [1].
Toutefois la pertinence du schéma construit dépend non seulement de l'algorithme employé mais aussi du niveau de normalisation de la base de donnée [3], [4], [2].

Hainaut explique que dans un souci de performances, certains concepteurs dé-normalisent volontairement leur base de données en ajoutant de la redondance d'information. Or l'élimination de cette redondance dans un processus de conceptualisation peut difficilement être automatisée. Dans les écrits [2] et [4], on explique qu'à partir de la troisième forme normale (dont FNBC [5]), on peut espérer obtenir un modèle conceptuel exploitable en utilisant un algorithme car la redondance est impossible.

2.2 Sélection de solutions approchantes

Cet état de l'art est complété par une recherche de solutions existantes de rétro-conception sur Internet. On peut dégager trois types de solutions explorées : les applications de visualisation de graphes, les applications d'extraction de métadonnées et les client complets comprenant ces deux types de fonctionnalités.

Parmi les clients complets testés, aucun n'a retenu l'attention car bien que proposant des solutions de connexion à divers SGBD, la fonctionnalité de rétro-ingénierie n'est jamais allée plus loin que l'obtention d'un pseudo schéma relationnel.

4. Data Manipulation Statements.
5. Forme normale de Boyce-Codd.

Plusieurs solutions de visualisation de schémas de données ont été testées. Deux ont retenu l'attention : Graphviz, une API [6] Open-Source pour la visualisation de graphes, et une application web dédiée aux bases de données développée au sein de l'Université.

Pour les solutions d'extraction, une API a fortement retenu l'attention. Cette solution appelée SchemaCrawler permet d'extraire les métadonnées d'une base de données et de proposer ces informations sous la forme de POJOs [7]. De plus, elle offre plusieurs solutions de connexion à divers SGBD via JDBC [8].

Afin de faciliter le positionnement concernant les outils existant testés, une grille d'évaluation (figure 1) a été mise en place.

	Graphes	Schéma Entités/ Associations	Reverse Engineering	Principaux SGBD pris en charge	Dernière mise à jour	Facilité Intégration	Communauté « active »
Prefuse	OUI	NON	NON	NON	12/2012	NON	OUI
GraphViz Java API	OUI	NON	NON	NON	02/2011	OUI	OUI
Schema Spy	OUI	NON	OUI	Oracle PGSQL MySQL SQLServer SQLite	06/2010	OUI	NON
SQL Power Architect	OUI	NON	OUI	Oracle PGSQL MySQL SQLServer SQLite	08/2011	NON	OUI
MogWai ER-Designer	OUI	NON	OUI	Oracle PGSQL MySQL	07/2012	NON	OUI
Schema Crawler API	OUI	NON	OUI	Oracle PGSQL MySQL SQLServer	11/2012	OUI	OUI

FIGURE 1 – Grille d'évaluation des solutions retenues pour l'état de l'art.

Sur la figure 1, les cases colorées en vert indiquent que la solution (représentée en ligne) répond de manière acceptable à la contrainte (représentée en colonne). Les cases colorées en orange indiquent que la solution testées ne répond pas au critère en question. Les détails liés à la contraintes sont éventuellement indiqués dans la case. Une présentation plus détaillée de quelques solutions explorées durant l'état de l'art est disponible dans l'annexe 1.

6. Application Programming Interface.
7. Plain Old Java Object.
8. Java DataBase Connectivity.

3 Proposition d'une solution

3.1 Schéma entité/association

Ce travail propose un algorithme de rétro-conception et son implémentation afin d'obtenir un schéma entité/association.
Ce type de schéma est très largement employé quand il s'agit d'obtenir une vision conceptuelle d'une base de données. Les entités sont des objets concrets que l'on peut instancier. Ils ont une valeur sémantique et peuvent représenter des éléments du monde réel. Les associations représentent les dépendances entre ces objets. Celles-ci peuvent être plus ou moins fortes et impliquer ou non l'existence de certains objets. Il existe plusieurs formalismes pour les schémas entité/association. Un des plus utilisés est celui de CHEN [9]. Une illustration de différents formalismes est disponible en Annexe 3. La figure 2 présente un exemple de schéma entité/association.

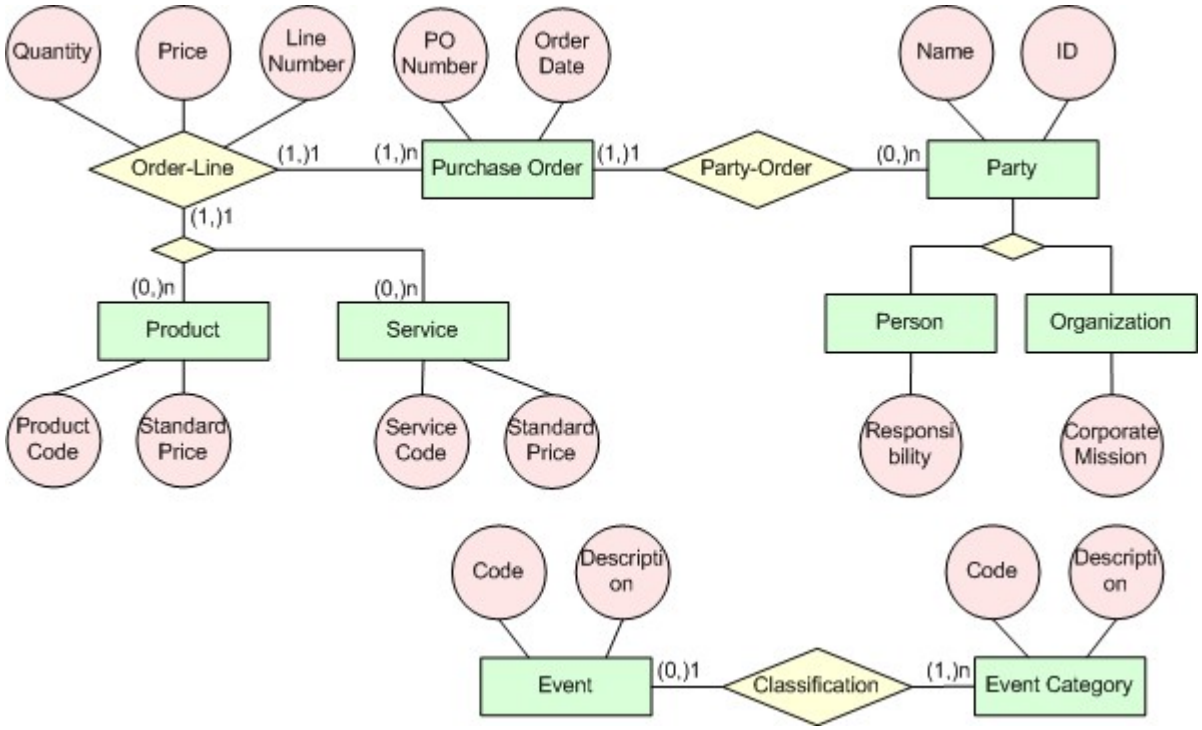

FIGURE 2 – Un schéma Entité/Association.

Les entités sont symbolisées par des rectangles (ici en vert). Les associations sont représentées par des losanges (ici en jaune). Elles sont liées à des entités et la "cardinalité" i.e. multiplicité du lien est inscrite sur le trait de liaison. Les associations et les entités peuvent posséder des attributs. Ce sont les éléments d'information qui leur sont associés. On les schématise par des ronds (ici en rose). Cette vue permet de distinguer rapidement les objets composant le système et leurs dépendances.

9. Peter Pin-Shan Chen, précurseur du modèle entité association en 1976.

3.2 Algorithme de rétro-conception

Voici l'algorithme de rétro-ingénierie, inspiré des travaux de M. Petit, de M. Hainaut, élaboré pour ce projet.

Algorithm 1 Conceptualization of a relational object model

```
Inputs : A Database object containing a relational model.
Outputs : A list of entities and associations.
Entities[] ← new Array of Entity
Associations[] ← new Array of Association
foreach t in Database.Tables do
    if t.hasIdentifier = true then
        foreach cW in t.weakColumn do
            Entity strong ← findEntity(cW.associateTable)
            Entity e ← createWeakEntity(t,strong)
            removeAttribute(cW,e)
            Boolean WeakCreation ← true
            Associations.add(CreateAssociation(SomeName,e,1-1,strong,0-N))
        end for
        if WeakCreation not true then
            e ← new Entity(t)
            Entities.add(e)
        end if
        foreach ifk in t.IncludedForeignKeys do
            Associations ← Associations + CreateAssociation(SomeName,e, 0-N ,eM,0-N)
        end for
        foreach cS in t.standardColumns do
            foreach t2 in getOtherTables(t) do
                if cS.isPartOfMissingEntity(t,t2) then
                    Entity eM ← findEntity(cS)
                    if eM = null then
                        eM ← new Entity(cS)
                    end if
                    Entities ← Entities + eM
                    Associations.add(CreateAssociation(SomeName,findEntity(t), 0-N ,eM,0-N))
                    Associations.add(CreateAssociation(SomeName,findEntity(t2), 0-N ,eM,0-N))
                end if
            end for
        end for
    else
        if t.RelatedEntites.Size >= 2 then
            foreach eR in t.RelatedEntites do
                Link tabLinks ← tabLinks + new Link(eR)
            end for
            Link tabLinks ← tabLinks + new Link(SearchEntity(t))
            Associations.add(CreateAssociation(SomeName,tabLinks))
        end if
    end if
end for
```

Une version plus détaillée de cet algorithme est proposée en Annexe 2.

3.3 Explications

Pour illustrer l'explication de cet algorithme, on utilisera un exemple de modèle relationnel (cf. figure 3) regroupant les principaux cas de figure traités. Ces cas représentent à chaque fois la découverte d'éléments essentiels d'un schéma entité/association.

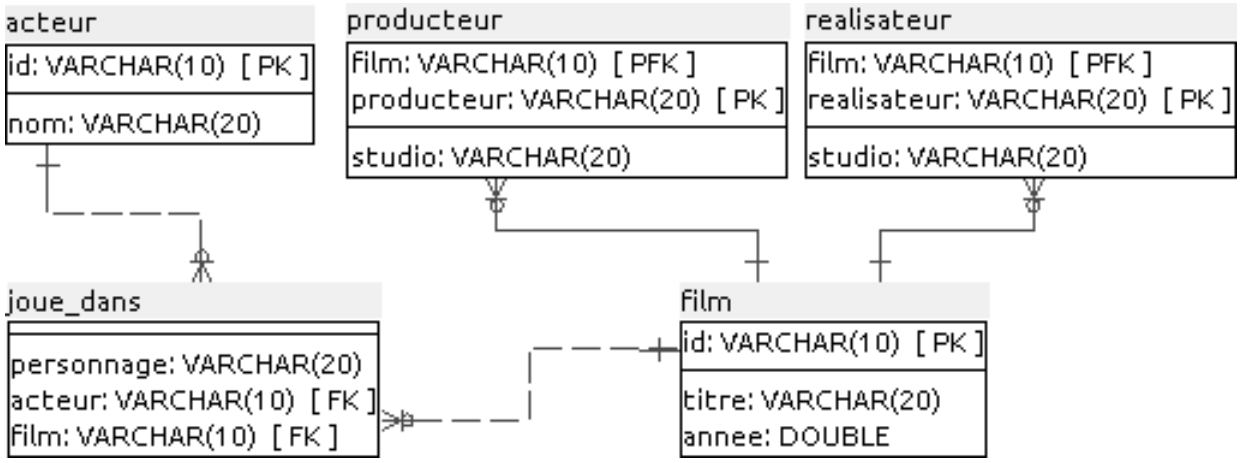

FIGURE 3 – Exemple de modèle relationnel.

Les entités faibles (Algorithme : ligne 4 à 10) sont un premier élément nécessaire dans une vue conceptuelle. Elles sont rattachées à une autre entité et ne peuvent exister sans. Ce lien a également une dimension sémantique. L'entité maître est alors appelée entité forte. Pour obtenir les entités faibles, on regarde si une partie de la clé primaire d'une table est associée par une clé étrangère à l'identifiant d'une autre table. Cet identifiant fait parti de la clé primaire de la seconde table. La découverte des entités fortes correspond au processus inverse. La mise en place d'une entité faible provoque immédiatement la création d'une association. Sur la figure 4 les tables producteur et réalisateur doivent donner lieu à des entités faibles de film.

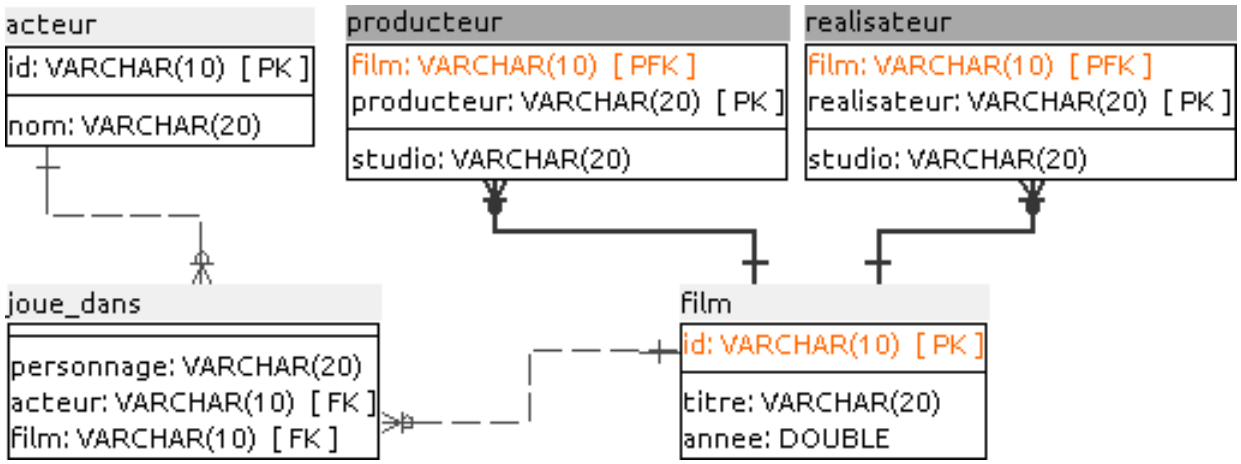

FIGURE 4 – Détection d'entités faibles dans un modèle relationnel.

Les autres entités (Algorithme : ligne 12 à 15) seront soit fortes soit normales. Leur découverte consiste simplement à traduire les tables correspondantes, qui ne sont pas associées à une entité faible, en de nouvelles entités. Sur la figure 5 la table acteur peut être traduite directement en une entité.

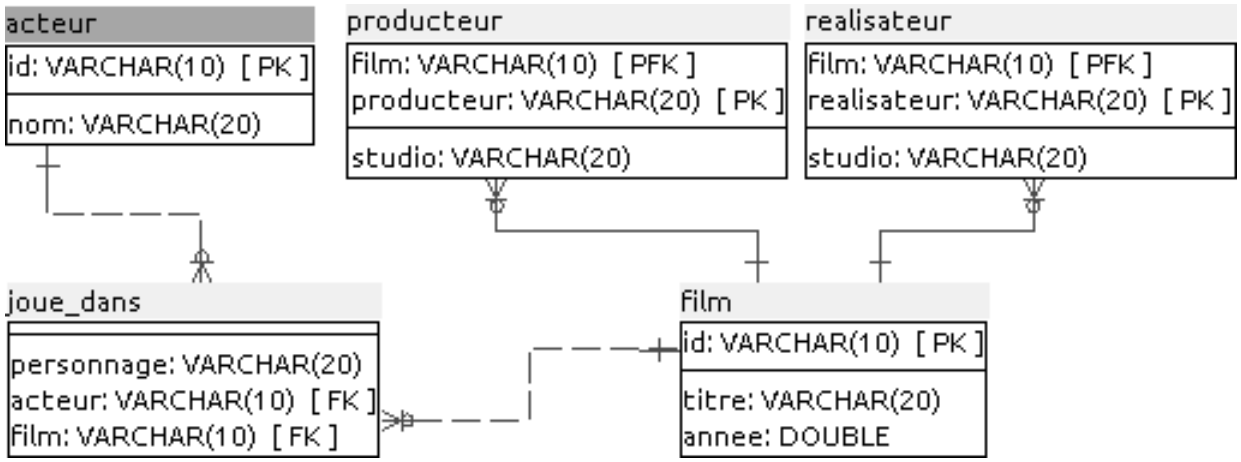

FIGURE 5 – Détection d'une entité dans un modèle relationnel.

Certaines entités cachées (Algorithme : ligne 21 à 29) doivent également être découvertes. Pour cela on détecte d'abord s'il y a une correspondance entre les noms de certaines colonnes de tables. Pour chacune de ces correspondances on va comparer les instances associées des différentes tables. Si la correspondance est supérieur à 50%, on considère que les colonnes liées représentent un objet concret. Ces colonnes sont donc agrégées pour devenir une entité au lieu de plusieurs attributs identiques. Enfin, pour chaque table "créatrice" dont la colonne en question fait partie de la clé primaire, l'entité associée à la table devient entité faible de la nouvelle entité crée. Sur la figure 6 l'attribut studio est en fait une entité studio indépendante cachée.

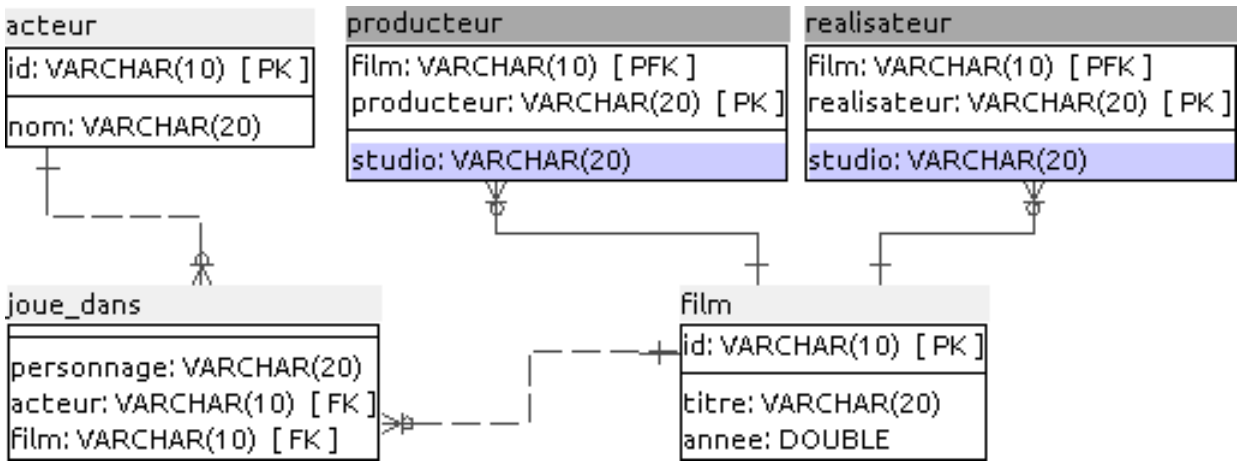

FIGURE 6 – Détection d'une entité cachée dans un modèle relationnel.

Un autre élément à considérer peut être appelé "table d'association" (Algorithme : ligne 33 à 39). Il s'agit de tables qui n'ont pas d'identifiant propre. C'est à dire que toutes les composantes de la clé primaire appartiennent à des clés étrangères pointant vers des identifiants d'autres tables. Ces tables doivent être transformées en associations, reliées aux entités associés par les composantes de la clé primaire de la table d'association. Sur la figure 7, la table joue_dans doit être transformée en une association entre acteur et film.

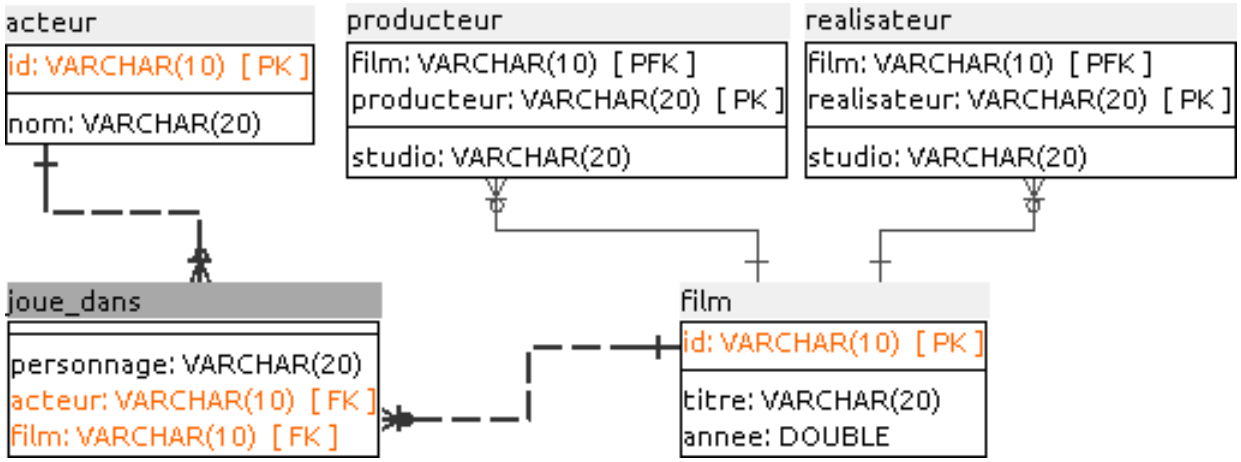

FIGURE 7 – Detection d'une association à partir d'une table d'un modèle relationnel.

Le dernier élément que l'on considère dans cette démarche sont toutes les autres associations (Algorithme : ligne 16 à 18). Il s'agit des clés étrangères dont la colonne initiale n'est pas une composante de la clé primaire de la table d'origine. Chacune de ces clés étrangères peut engendrer la création d'une association simple entre deux entités associés aux deux tables correspondantes. Sur la figure 8 il existe une association simple entre acteur et joue_dans.

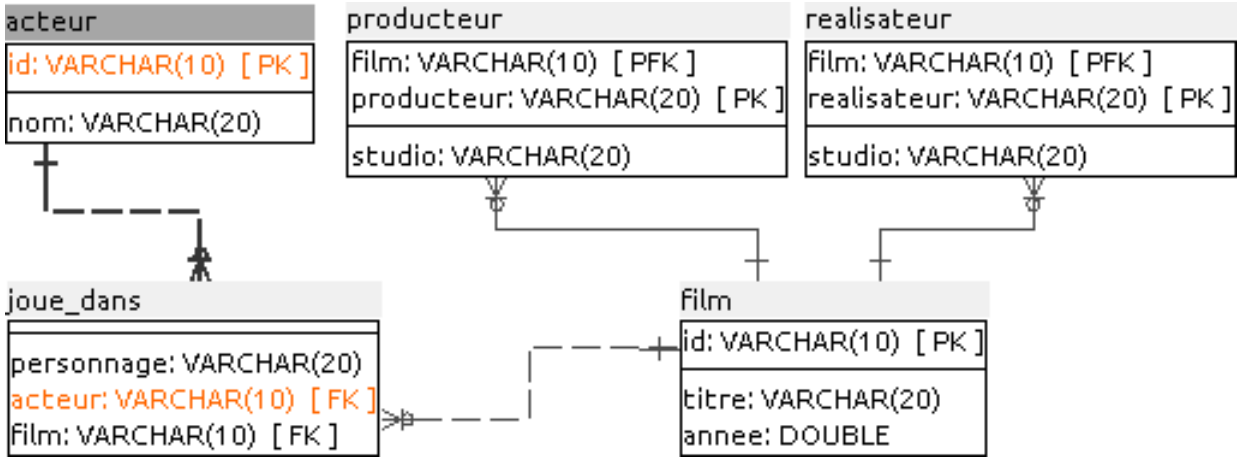

FIGURE 8 – Détection d'une association dans un modèle relationnel.

4 Expérimentation

4.1 Protocole

Une implémentation de l'algorithme proposé a été mise en place. Pour cela le langage Java a été choisi. Ce choix a été influencé par le fait que cette implémentation doit être intégrée à une application développée avec ce langage. C'est donc la simplicité d'intégration qui a été privilégiée. La structure de cette implémentation repose sur un projet Maven [10] Web. Elle est composée de trois éléments : une dépendance SchemaCrawler pour Maven, un Package Java permettant la connexion et la conceptualisation, un Package Java permettant l'exportation du schéma Entité/Association au format DOT avec GraphViz. La connexion est assurée grâce à JDBC et à l'API SchemaCrawler. Pour le SGBD Oracle, une solution alternative [11] à SchemaCrawler est utilisée.
Un processus de rétro-conception sur des bases de données MySQL, PostgreSQL et Oracle a été expérimenté avec l'algorithme implémenté sur des environnements Windows et Unix. Le résultat d'une des expérimentations est présenté en annexe 4.

4.2 Évaluation

Pour ce projet, quatre versions stables de l'algorithme ont été testées sur l'exemple présenté en section 3.3. A chaque fois le nombre d'entités et d'associations a été sauvegardé. Une rétro-conception manuelle a permis de déterminer le nombre exact d'entités et d'associations attendues. Il s'agit de cinq entités et cinq associations. Les quatre versions testés sont :

- Version 1 : Traduction des tables en entités et clés étrangères en associations.
- Version 2 : Version 1 + Prise en compte des entités faibles et de leurs associations.
- Version 3 : Version 2 + Prise en compte des entités cachées.
- Version 4 : Version 3 + Prise en compte des tables d'association (version finale).

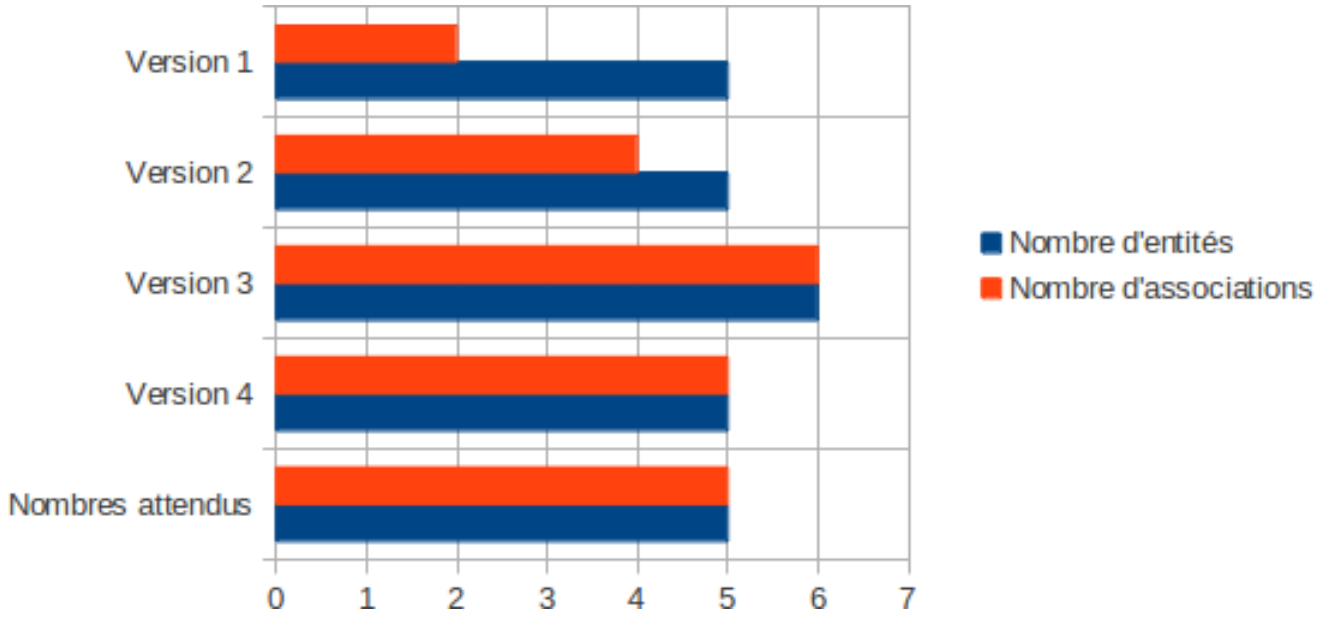

FIGURE 9 – Diagramme d'évaluation de plusieurs algorithmes de rétro-conception.

10. Outil permettant l'automatisation de productions de projets Java.
11. http://liris.cnrs.fr/ecoquery/dokuwiki/doku.php?id=enseignement:aide:oracle

Le diagramme de la figure 9 présente les nombres d'entités et d'associations obtenus après rétro-conception via diverses versions de l'algorithme.

La version 1 ne détecte pas le bon nombre d'associations car elle ne considère pas les entités faibles. Ces dernières entraînement pourtant la création d'associations.

La version 2 n'a pas détecté le bon nombre d'associations. En effet elle considère la table "joue_dans" comme une entité. Or comme présenté dans la section 3.3, cela doit donner une association.

La version 3 révèle trop d'entités et d'associations. Les deux nouvelles associations détectées en plus de la version 2 proviennent de la découverte de l'entité cachée "studio". Malheureusement, comme pour la version 2, l'interprétation de la table "joue_dans" comme une entité et non comme une association provoque cette erreur car les deux clés étrangères de cette table sont transformées en deux associations.

La version 4, qui est la version finale, détecte correctement tous les cas de figure et sort les bons nombres d'entités et d'associations préalablement obtenus par un processus manuel.

5 Conclusion

Dans une société où les systèmes d'information fleurissent dans de nombreuses entreprises, la possibilité de garder une vision conceptuelle sur ses structures de données est un atout décisif. Dans ce but, l'approche proposée dans ce document se veut pragmatique, objective et se base sur un travail de recherche sérieux. Elle permet de faciliter la compréhension du sujet et son implémentation représente une base idéale pour un outil de maintenance, d'amélioration ou même d'audit de bases de données pour la conception d'applications.

Néanmoins, si l'on souhaite s'approcher encore plus du résultat d'un expert, l'implémentation peut être améliorée sur plusieurs points.
Dans un premier temps, on pourrait imposer que chaque association ne possède au maximum que deux liens. Ainsi il faudrait traiter ce cas dans l'algorithme. Ensuite, le processus pourrait s'accompagner d'interactions avec l'utilisateur afin de proposer des choix conceptuels tout au long du déroulement de l'algorithme.
On pourrait aussi imaginer une interface graphique plus complète au niveau de la gestion des erreurs de connexion ou de la présentation des schémas.
Enfin, le tout pourrait être compilé sous la forme d'un plugin [12] pour une application ou un framework [13] de développement.

6 Remerciements

Je tiens à remercier messieurs Nicolas Lumineau et Fabien Duchateau pour m'avoir accepté dans ce projet de recherche et pour m'avoir apporté de précieux conseils et une certaine confiance.

Je remercie également l'équipe d'étudiants en Master II Technologies de l'Information pour m'avoir chaleureusement intégré à leur projet d'application.

12. Fonctionnalité additionnelle.
13. Surcouche applicative permettant de faciliter le développement.

Références

[1] Jean-Luc Hainaut, 2002. Introduction To Database Reverse Engineering , Problems, Methods and Tools, Tutorial notes. University of Namur.

[2] Jean-Marc Petit, 1996. Fondements pour un processus réaliste de rétro-conception de bases de données relationnelles, Villeurbanne.

[3] Jean-Luc Hainaut, 1991. Database reverse engineering : Models, techniques and strategies, 10th Entity-Relationship Conference, San Mateo.

[4] Mfourga, 1996. Database reverse engineering : a form-driven approach. Colloque CARI '96 : Actes du 3e Colloque africain sur la recherche en informatique, ORSTOM, Paris.

[5] Martin Andersson, 1994. Extracting an entité relationship schema from a relationnal database through reverse engineering. 13th International Conference on the Entity-Relationship Approach, Manchester, United Kingdom.

[6] Jean-Marc Petit, Jacques Kouloumdjian, 1994. Using Queries to Improve Database Reverse Engineering. 13th International Conference on the Entity-Relationship Approach, Manchester, United Kingdom.

7 Annexes

7.1 Annexe 1 : Les solutions explorées pour l'état de l'art ayant retenu l'attention.

7.1.1 Graphviz Java API

Graphviz [14] est une API Open Source dédiée à la visualisation de graphiques et développée par les laboratoires de recherche d'AT&T.
Elle est composée d'un ensemble d'outils dédiés à la représentation de graphes. Ces derniers sont décrits grâce au langage DOT [15].
Les fichiers DOT peuvent également être lus par les librairies Canviz [16] ou Grappa [17]. Une API Java existe pour simplifier l'utilisation de Graphviz avec des POJOs. Néanmoins et bien que très simple à utiliser, elle ne permet de générer que des diagrammes fixes au format GIF, PNG, PDF ou encore SVG.

7.1.2 Prefuse

Prefuse est un ensemble d'outils Open Source développés sous la forme d'un framework de visualisation de schémas de données. Il repose sur JAVA (1.4) et sur la librairie graphique Java2D. Il est proposé sous la licence BSD (Berkeley software distribution license), licence libre. Ces outils permettent de créer toute sortes de graphes et de gérer des interactions avec l'utilisation au travers de nombreux effets.

7.1.3 Eclipse Entity Relationship Diagram Editor

Ce plugin pour l'IDE Eclipse permet de créer "à la main" un schéma entité/association et de l'exporter en SQL ou en POJOs. Bien qu'il respecte la structure attendue des schémas il est difficilement ré-utilisable et n'est pas Open-Source.

7.1.4 SchemaCrawler

SchemaCrawler est une API Open-Source qui permet de récupérer les méta-données d'une base de données sous la forme de POJOs. Concrètement il s'agit d'une couche supplémentaire de l'API JDBC [18] permettant d'utiliser les données récupérées comme des objets Java. Cette API possède aussi une version "utilisateur" permettant via des commandes de manipuler les informations de bases de données. On peut par exemple filtrer les résultats ou même générer des graphes grâce à une intégration de GraphViz.
SchemaCrawler, distribuée sous la licence LGPL [19], est très simple à implémenter et possède même une version pour les projets Maven.

7.1.5 SchemaSpy

SchemaSpy est un projet Open Source développé en JAVA et permettant de générer des rapports sur la structure d'une base de données. Les résultats sont proposés au format html. Un rapport n'est pas le résultat d'un processus de rétro-conception mais simplement d'une analyse structurelle permettant un audit de la base de données concernée. L'application est sous licence LGPL. Elle utilise le langage DOT de GraphViz pour la représentation des schémas et l'API JDBC.

14. Graph Visualization Software.
15. Le langage DOT est un langage de description de graphe dans un format texte.
16. Canviz est une librairie Javascript pour la représentation de graphes GraphViz sur le Web.
17. Grappa est un package Java permettant de simplifier la représentation de graphes GraphViz.
18. Java Database Connectivity.
19. Gnu Lesser General Public Licence.

7.1.6 SQL Power Architect

Ce projet propose une application Java Open-Source sous licence GNU GPL version 3 de modélisation de schémas de base de données. Il est également basé sur l'API JDBC. Il s'agit ici d'une application complète permettant de se connecter à des bases de données pour générer un modèle relationnel. Ce dernier peut être modifié via une interface utilisateur et enfin être propagé de nouveau sur la base de données. Power Architect ne propose malheureusement que des schémas relationnels et ne peut donc être considéré comme une solution acceptable pour le sujet.

7.1.7 MogWai ER-Designer

Cette application Open-Source sous licence GNU GPL se situe dans la même lignée que SQL Power Architect. Elle permet une forme d'extraction de base de données recréant un modèle relationnel et propose une interface graphique permettant d'agir sur les diagrammes. Elle offre aussi de nombreuses options comme par exemple la comparaison de schémas, un diagnostique d'optimisation ou la génération de code DDL [20]. Comme pour Power Architect, l'absence de schémas entités/associations fait que ce projet ne répond pas à exactement aux besoins du sujet.

7.2 Annexe 2 : Version plus détaillée de l'algorithme.

Algorithm 2 Conceptualization of a relationnal object model

```
    Inputs : A Database object containing a relational model.
    Outputs : A list of entities and associations.
 1: List of Entite listEntities ← new List of Entite
 2: List of Association listAssociations ← new List of Association
 3: // TransitionTable : An object to contain a table of the relational
 4: // model and several useful features.
 5: TransitionTable table ← null
 6: foreach t in ex.Database.Tables do
 7:     table ← new TransitionTable
 8:     // Creation of Entities
 9:     if t.hasIdentifier = true then
10:         // Weak Entity Case
11:         foreach Column c in table.WeakAssociationColumns do
12:             Entite strong ← new SearchEntity(c.ReferencedColumn)
13:             if strong = null then
14:                 String refName ← c.TableReference
15:                 strong = table.EntityGeneration(c.TableReference)
16:                 listEntities.add(strong) ;
17:             end if
```

20. Langage de Définition de Données.

Algorithm 3 Suite de l'algorithme

```
            Entite we ← SearchEntity(t)
            if we != null then
                we.setForte(strong) ;
            else
                we ← WeakEntityGeneration(t) ;
            end if
            if we != null then
                listEntities.add(we) ;
                listAssociations.add(new WeakAssociation(we,strong)
            end if
        end for
        // Normal Entity Case
        if !table.isWeakEntity then
            Entite e ← EntityGeneration(t)
            listEntities.add(e) ;
        end if
        // Association Case
        ForeignKey fks ← null
        foreach fk in t.ImportedForeignKeys do
            foreach ColumnReference cr in fks.ColumnReferences do
                Entite initEnt ← SearchEntity(cr.ForeignKeyColumn.Parent)
                Entite destEnt ← SearchEntity(cr.PrimaryKeyColumn.Parent)
                if destEnt = null then
                    destEnt ← EntityGeneration(cr.PrimaryKeyColumn.Parent)
                    listEntities.add(destEnt)
                end if
                if !ExistAssociation(initEnt, destEnt) then
                    Association a ← CreateAssociation(initEnt, destEnt) ;
                    listAssociations.add(a) ;
                end if
            end for
        end for
        // Missing Entity Case
        foreach Column c in t.Columns do
            foreach Table oT in allTables do
                if isMissingEntite(c, t, oT then
                    Entite MissingE ← .SearchEntity(c)
                    listEntities.add(MissingE)
                    Entite e1 ← SearchEntity(t)
                    Entite e2 ← SearchEntity(oT)
                    if t.getColumn(c).isPartOfPrimaryKey then
                        e1.setForte(MissingE) ;
                    end if
                    if oT.getColumn(c).isPartOfPrimaryKey then
                        e2.setForte(MissingE) ;
                    end if
                    listAssociations.add(CreateAssociation(e1,MissingE))
                    listAssociations.add(CreateAssociation(e2,MissingE))
                end if
            end for
        end for
```

Algorithm 4 Suite de l'algorithme

```
    else
        List of Lien links ← new List of Lien
        foreach Table rt in t.RelatedTables do
            Entite e ← SearchEntity(rt)
            if e != null then
                links.add(new Lien(e))
            end if
        end for
        if LinksNumber > 1 then
            Association a ← new Association(links)
            listAssociations.add(a) ;
        end if
    end if
end for
```

7.3 Annexe 3 : Différents formalismes de schémas conceptuels.

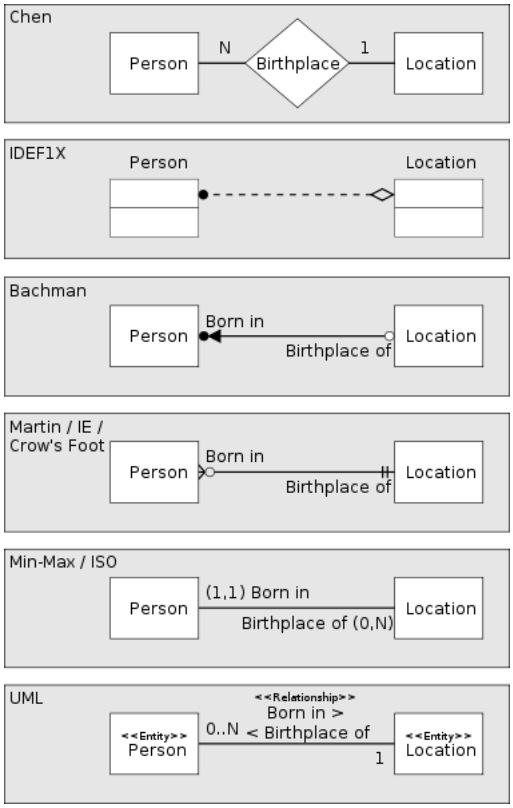

FIGURE 10 – Différents formalismes de schémas conceptuels.

7.4 Annexe 4 : Exemple de la section 3.3 testé avec l'implémentation réalisée durant le projet.

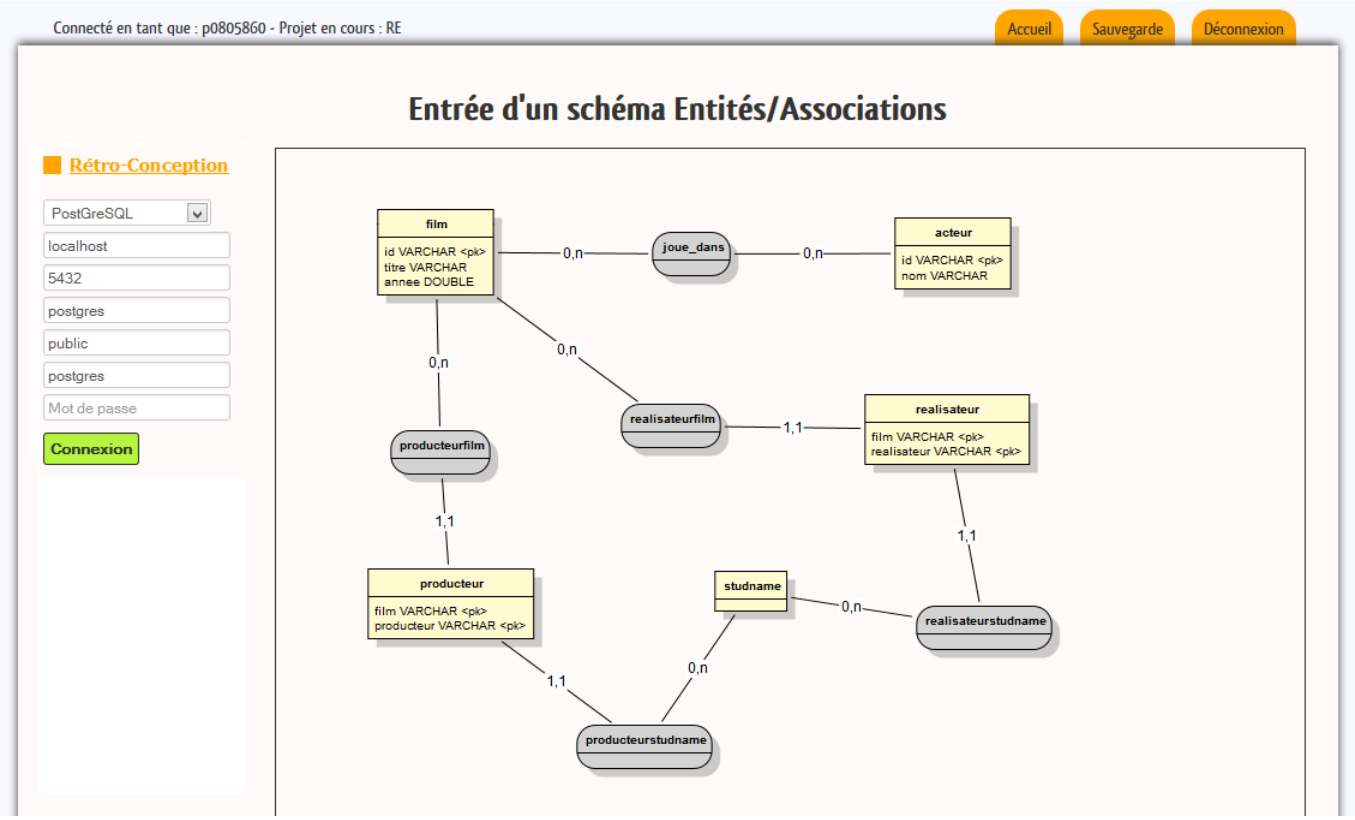

FIGURE 11 – Résultat brut de rétro-conception obtenu dans un environnement web à partir de l'algorithme proposé.