Projet
Gestion des dépendances fonctionnelles et normalisation
Cours de bases de données 1

UMONS
Faculté des Sciences
BAC 2 Science-Informatique.

Charlier Maximilien
Ducruet Corentin

Année académique
2013-2014

Table des matières

0.1 Présentation

Il nous a été demandé de concevoir une application graphique permettant la manipulation de dépendances fonctionnelles au sein d'une base de données.
Pour se faire nous avons implémenté l'application en Python 2.7 et utilisé Sqlite3 pour gérer la base de données (comme demandé).
L'application permet de jongler avec les dépendances fonctionnelles pour chaque table (relation) de la base de données fournie. On entend par là le fait de pouvoir en ajouter/supprimer et de voir leurs impacts sur les relations qu'elles affectent. Pour chaque relation, il est possible de voir si elle respecte les normes `3NF` et `BCNF`.
Les tables pouvant contenir des tuples de données, on vérifiera que ses tuples respectent ou non les DF relatives à leur table. Si ce n'est pas le cas, l'utilisateur pourra visualiser les tuples posant problème pour chaque DF.
Il est demandé que l'application offre une exportation `3NF` de toutes relations présentent dans la base.

0.2 Point fort

Application supportant un grand nombre de relations (affichage prévu pour).

Affichage des DF non satisfaites et possibilité de lister les tuples qui la rendent fausse, plus de leur suppression ciblée.

Ajout de DF simplifiées via des cases à cocher.

Exportation 3NF disponible, avec possibilité de personnaliser le nom des relations exportées.

Affichage des DF qui sont des conséquences logiques d'autre DF.

Compatible Windows, Linux, Mac (utilisation de librairie disponible sur les 3 OS.)

Image libre de droit.

Exportation 3NF simplifiée en 2 étapes : choix des relations à exporter, renommage des relations générées issues des relations à décomposer.

0.3 Interface graphique

L'application utilise *Tkinter* pour l'interface graphique. Les images ont été réalisé par Charlier Maximilien (hors mis le logo de la FS et de l'UMONS.) via `Adobe©Photoshop©CC` et sont donc libre de droit. Voici quelques copies d'écran de l'interface graphique.

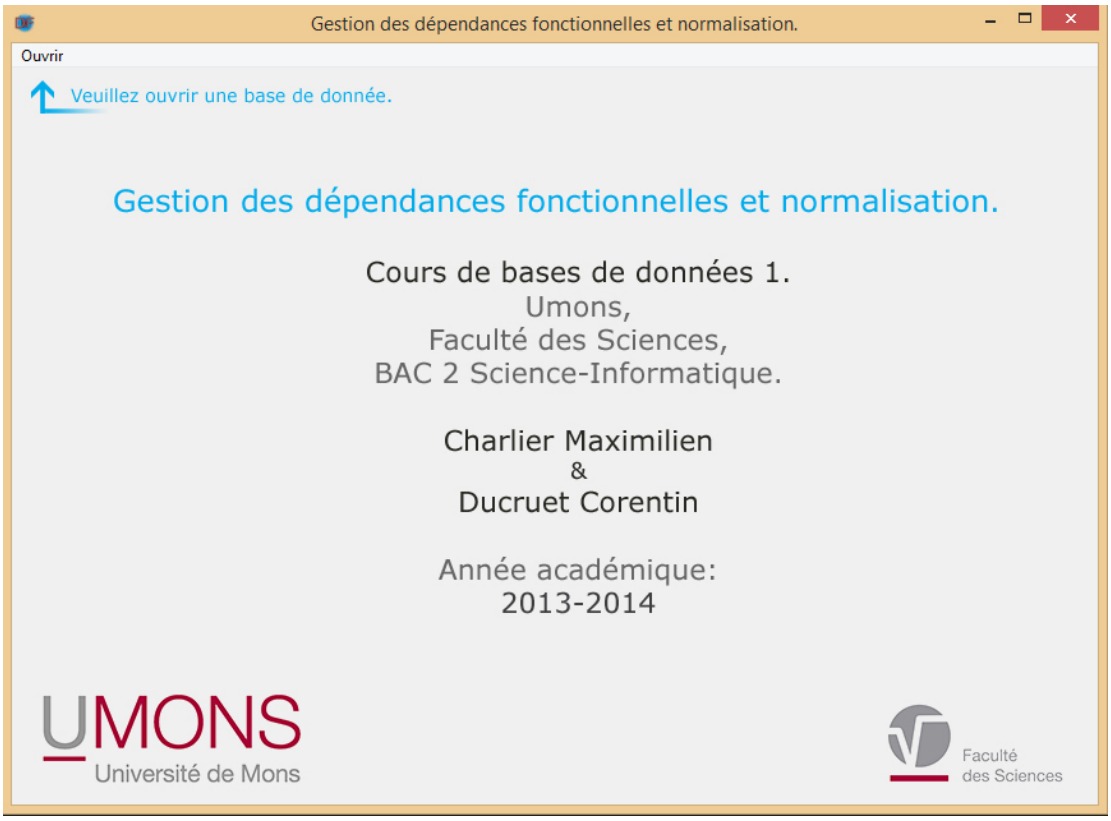

FIGURE 1 – Page d'accueil de l'application.
Possibilité d'ouvrir une base de données.

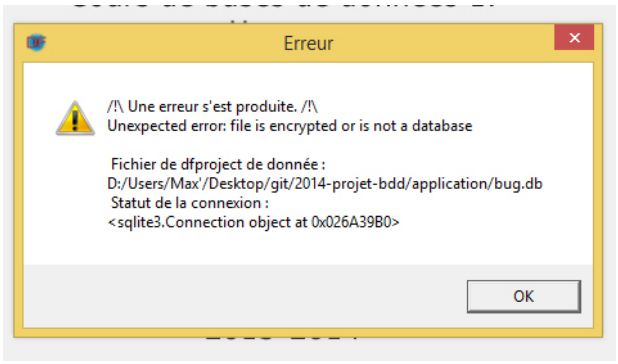

FIGURE 2 – Pop-up en cas de base de données incompatible.

FIGURE 3 – Affichage de la liste des relations avec des informations pour chacune de celles-ci :
Suis-je en 3NF/BCNF ?, nombre de clés, super-clés, tuples, attributs, DF.
Accès à diverses options : "Table" (Affiche les tuples que la relation compte), "DF" permet de voir les DF (en ajouter, supprimer), Faire l'exportation en 3NF.

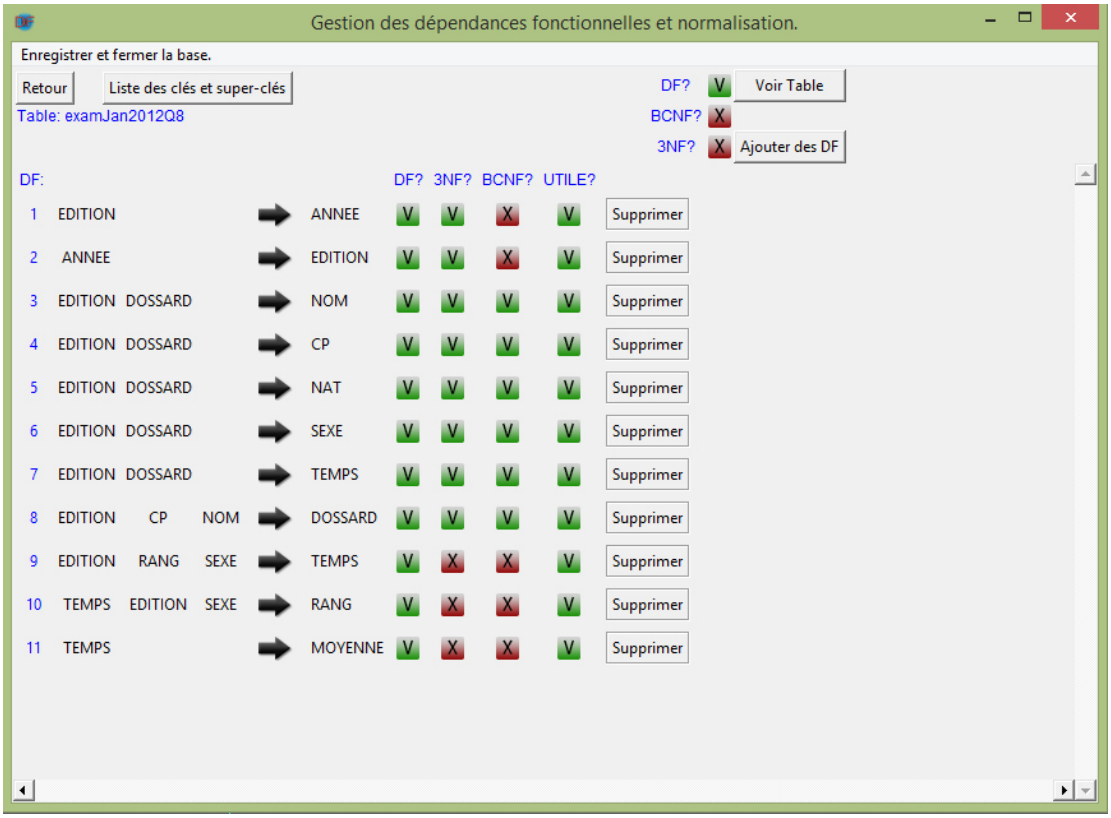

FIGURE 4 – Affichage des DF pour une relation.
Possibilité d'en ajouter, d'en supprimer.
De voir les clés et super-clés.
De voir les tuples de la relation.

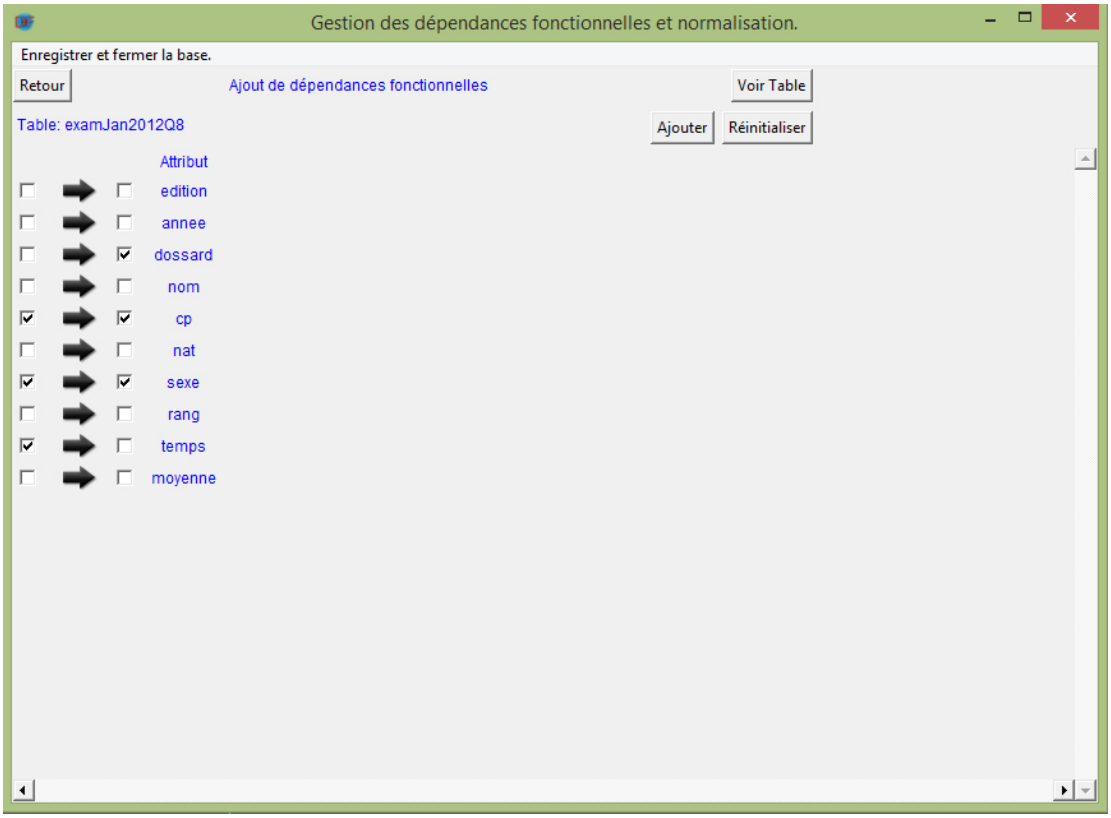

FIGURE 5 – Ajout des DF.
Comme vous pouvez le voir, c'est basé sur un système de cases à cocher facilitant l'ajout rapide de DF.

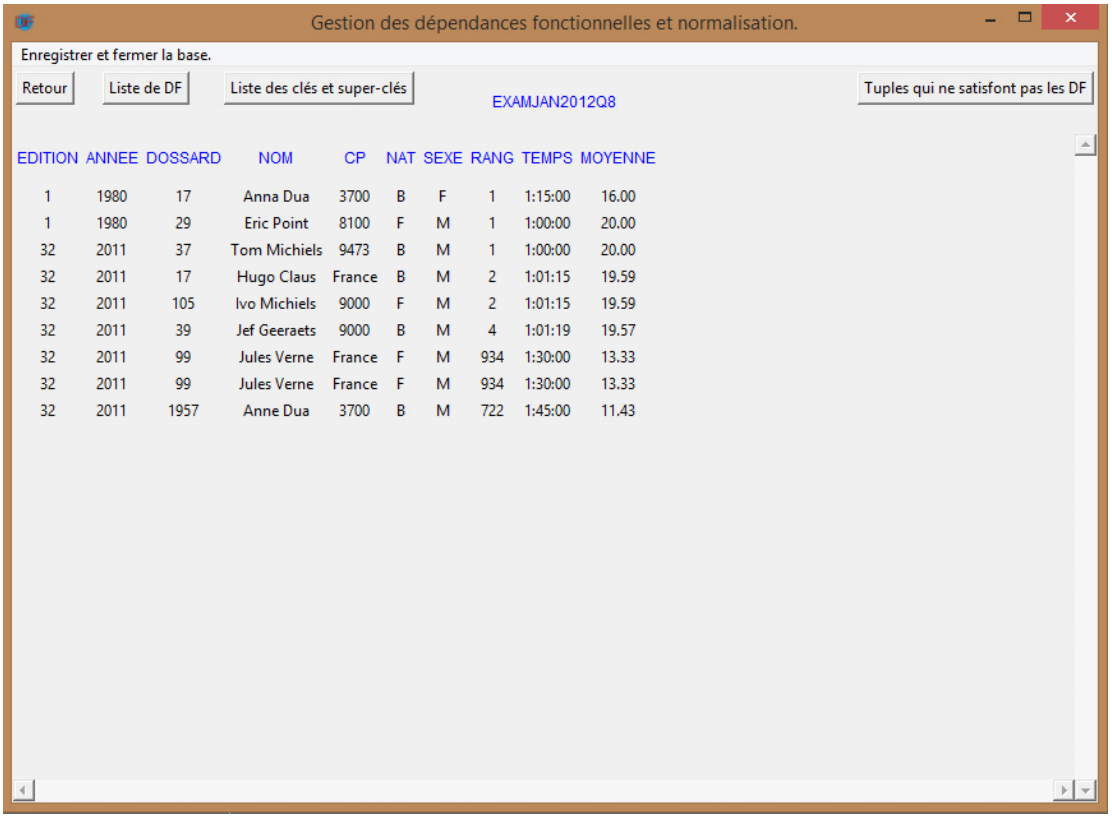

FIGURE 6 – "Voir table", affichage des tuples d'une relation.
On passe par cette page pour afficher les tuples à problèmes (le bouton ne s'affiche que s'il y a des tuples posant problèmes).
On peut aussi afficher la liste des clés-super-clés en passant par cette page.

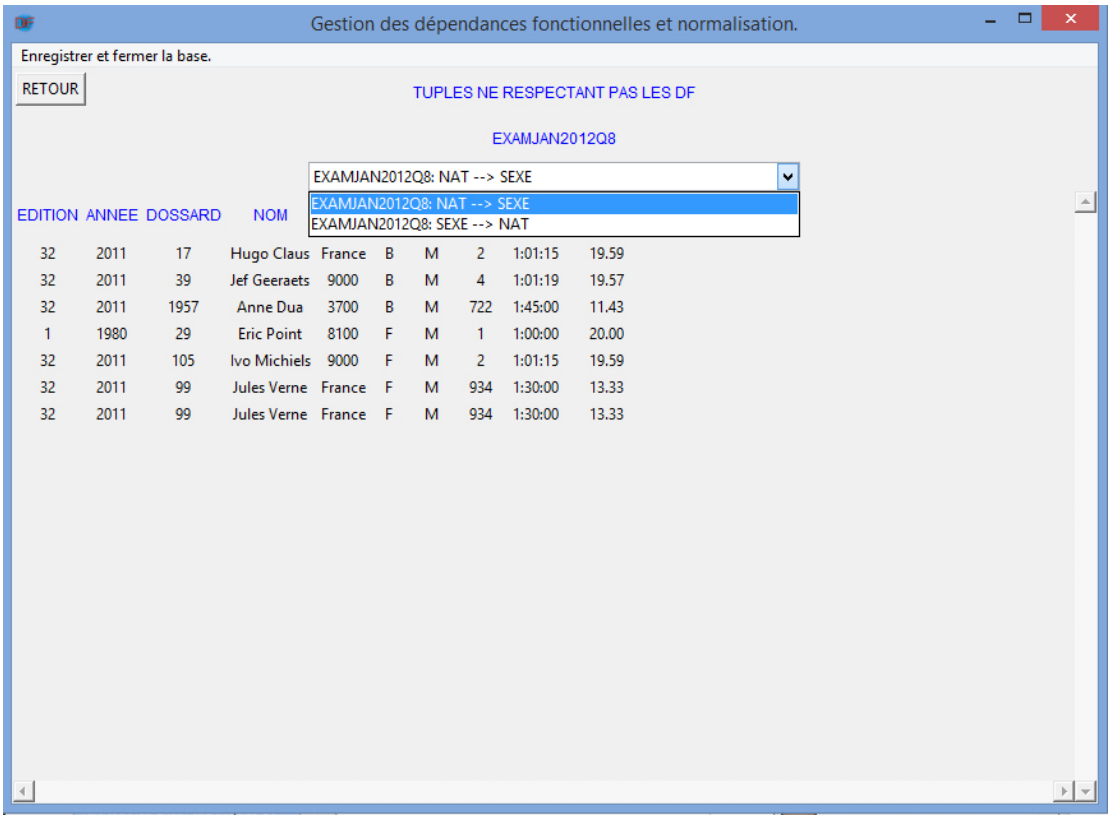

FIGURE 7 – Affichage des tuples qui ne respectent pas une DF (on peux choisir entre les DF concernées).

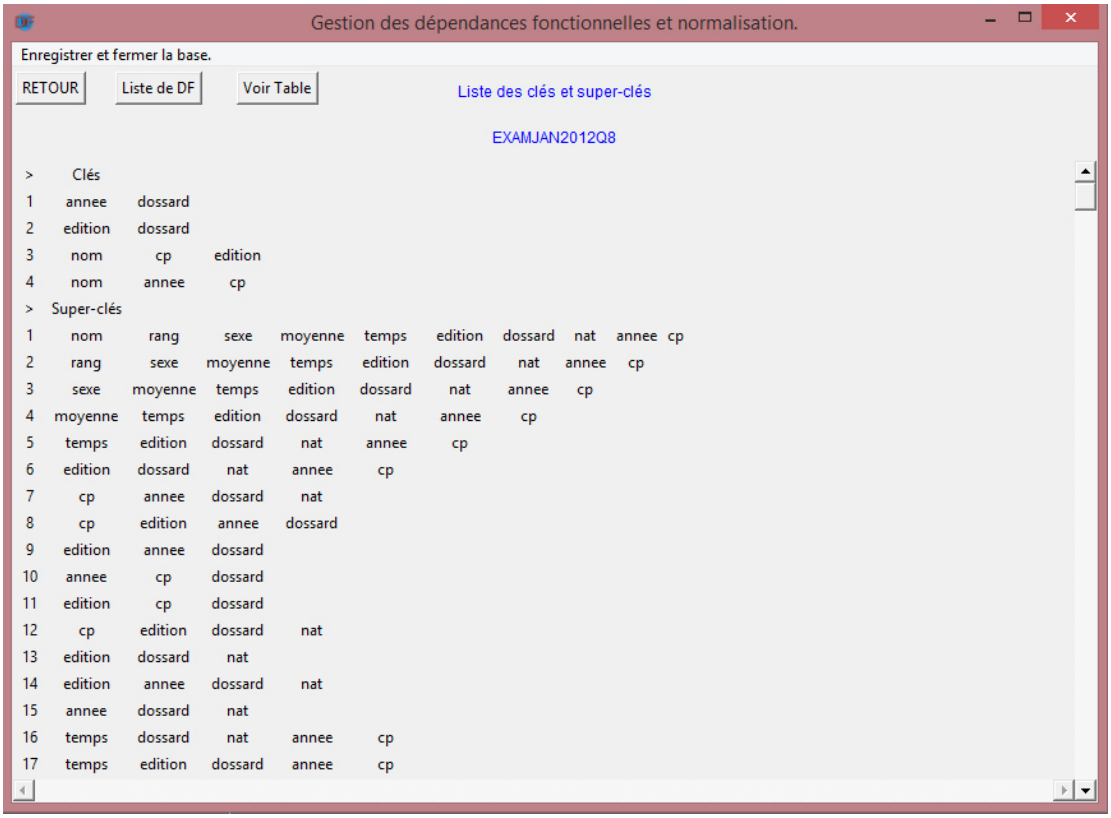

FIGURE 8 – Affichage des clés et super-clés pour une relation.
L'affichage de la page peut prendre un certain temps, explication dans la partie "problème".

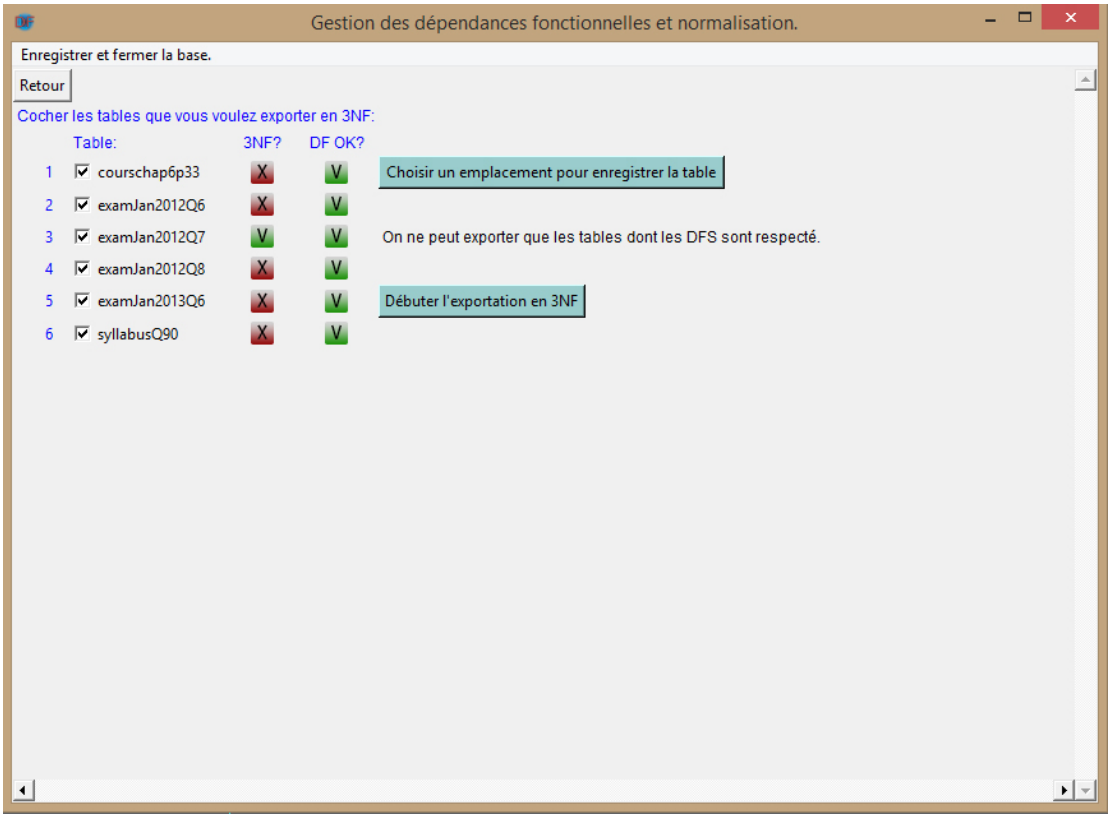

FIGURE 9 – Affichage du début de l'exportation, on choisit les tables à exporter et où elles seront exportées.

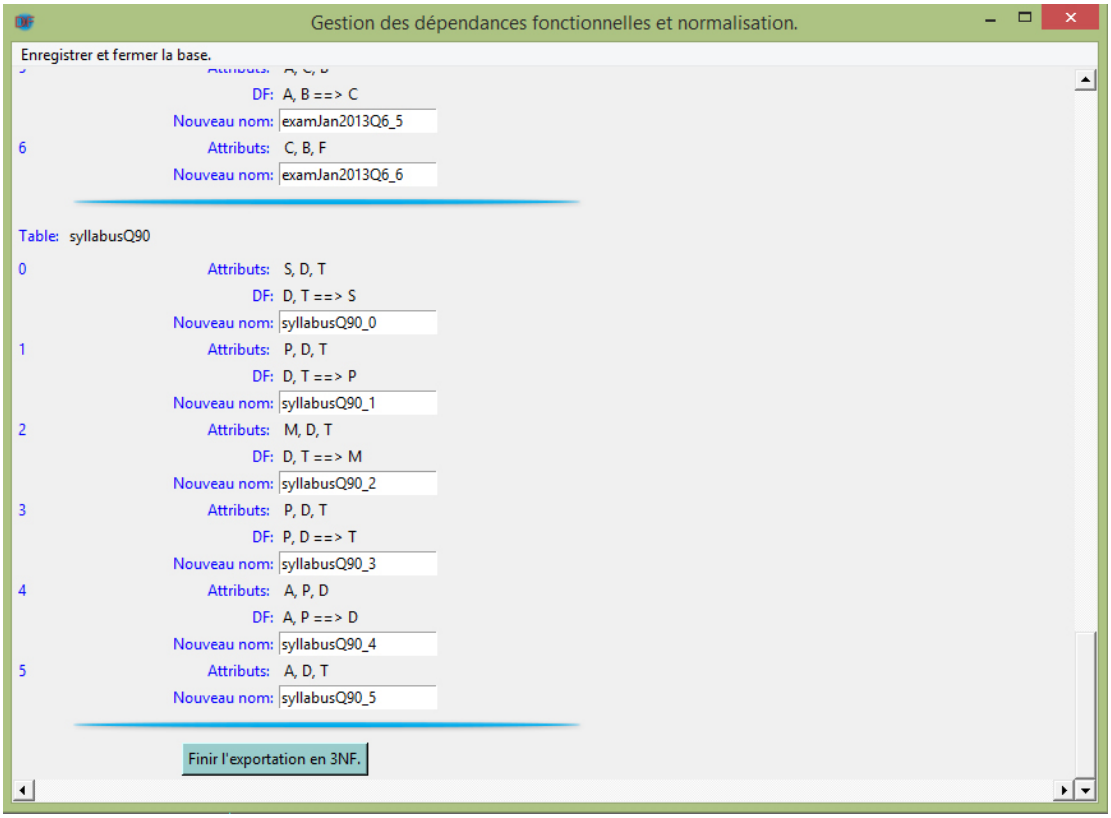

FIGURE 10 – 2ème étape, On choisit le nom des nouveaux schémas de table et on les exporte.
Une barre a été placée entre chaque relation pour faciliter la lecture.

0.4 Implémentation

On va présenter quelques aspects technique du projet :

0.4.1 Présentation rapide de la structure

Main.py Contient tout ce qui est interface graphique de l'application.

test.py Contient les tests faits sur l'application (Vérification du bon fonctionnement des interactions avec la base de données, et des méthodes relatives au dépendances fonctionnelles).

genBase.py Permet de générer une base de données pour pouvoir l'utiliser avec l'interface graphique.

dfproject Package contenant les algorithmes proprement dits du projet.

bddAction.py Permet de réaliser des actions sur la base de données comme :

Récupérer les **noms des colonnes** d'une table.

Récupérer la **structure d'une table**.

Récupérer la **liste des tables** présentent dans la base de données

Récupérer le **nombre d'attributs** d'une tables

Récupérer le **nombre de tuples** d'une table.

Récupérer le **nombre de DF** relatives à une table.

Récupérer le **nom des colonnes** et **les tuples** d'une table.

Copier une table vers une autre base de données.

Copier une table vers une autre base avec un nouveau schéma.

utility.py Permet de trier un texte ou de limiter sa taille.

fctDept.py Permet l'ajout de dépendance fonctionnelle à la base de données et la représentation de dépendance fonctionnelle.

Ajout de DF de la base de données

Mise à jour de DF de la base de données

Suppression de DF de la base de données

Récupérer une **liste des DF d'une relation**.

Afficher les DF au format texte pour les tests.

L'algorithme de **closure**.

L'objet **DF** une fois créé permet :

Si deux DF sont **égales**.

Si une DF est **réflexive**.

Si une DF est résultante de la **transitivité** de plusieurs autres.

De faire une **copie** d'une DF.

De voir si une DF est **vide**.

relation.py Permet de voir l'influence des DF sur une relation :

Voir les DF qui satisfont ou non une relation.

Voir les DF ayant des **attributs non existants** dans la relation.

Lister les **clés** et **super-clés**.

Lister les tuples qui rendent une DF fausses.

Lister les DF qui sont des conséquences logiques d'autres DF.

Dire si une relation est en **BCNF**/**3NF**

Dire pourquoi une relation n'est pas en **BCNF**/**3NF**

Faire une **décomposition 3NF** d'une relation.

0.4.2 Quelques algorithmes.

Décomposition 3NF d'une relation

Entrée : Une relation, la liste des attributs de la relation, la liste des DF de la relation (uniquement les DF qui ne sont pas des conséquences logique d'autres DF).
Sortie : Une liste d'ensembles d'attributs couplés chaque fois à une DF, qui sont une décomposition 3NF de la relation.
Algorithme :
Pour chaque DF de la relation, on considère une partie de décomposition composer de la DF obtenue et des attributs de celle-ci.
On fini l'algorithme avec l'ajout d'une clé quelconque combinée avec une DF nul.

Récupération des clés et super-clés.

Entrée : Une relation et la liste des DF de la relation (uniquement les DF qui ne sont pas des conséquences logiques d'autres DF).
Sortie : 2 listes d'ensembles d'attributs qui sont les clés et les super-clés d'une relation.
Algorithme :
On liste d'abord toutes les super-clés et on les place dans *liste-temps*. Pour cela, on utilise un algorithme récursif :

Entrée :

liste-df Une liste de DF.

set-attribut Un ensemble d'attributs.

liste-superkey Une liste de super-clés

Sortie : /

Effet : Remplis la liste des super-clés.

Algorithme :
Pour chaque attribut **faire** :

Pour chaque DF **faire** :

Une copie de la liste d'attributs.

Retirer l'attribut observé de la liste fraichement copiée.

Si l'attribut observé est à droite de la DF observée **et** que la liste d'attributs fraichement copiée contient tous les attributs à gauche de la DF **et** que la liste d'attributs fraichement copiée n'est pas déjà listée comme super-clé **faire** :

- Ajouter la liste d'attributs fraichement copiée dans la liste de super-clés.
- Recommencer l'algorithme avec comme entrée :
  - *liste-df* La liste de DF.
  - *set-attribut* La liste d'attributs fraichement copiée.
  - *liste-superkey* La liste des super-clés.

Ensuite, on va parcourir les ensembles d'attributs de *liste-temps* : pour chacun, si il n'est pas minimum dans *liste-temps* alors c'est une super-clé, sinon s'il ne contient aucune DF (attribut de gauche et de droite en même temps) alors c'est une clé.

Donner les tuples qui font qu'une DF n'est pas respectée dans une relation.

Entrée :

Une liste des attributs de la relation.

La DF observée.

Sortie : Une liste des tuples de la relation qui font que la DF n'est pas respectée.

Effet : /

Algorithme :
On va lister les tuples de la relation en les ordonnant par rapport aux attributs à gauche de "–>" de la DF.
On définit *value-rhs* comme non existant.
Ensuite, on va parcourir la liste et regarder la valeur présente dans la colonne relative à l'attribut à droite de "–>" de la DF :

(a) Si cette valeur est la même que celle présente dans *value-rhs*, alors on l'a deux fois pour un tuple. Donc la DF n'est pas respectée et on ajoute le tuple dans la liste des tuples qui font que la DF n'est pas respectée.

(b) On la garde en mémoire dans *value-rhs*.

0.5 Difficultés

0.5.1 Interface graphique

1. `Scrollbar`.
   L'ajout de **la possibilité de scrooler** dans l'environnement graphique m'a posé problème car ce n'est pas supporté de base pour une `Frame`. Pour réussir à en utiliser un je suis passé par l'utilisation du `canvas` sur lequel j'ai placé des `Scrollbar`s. J'ai ensuite placé une `Frame` dans le `canvas` et je lui ai appliqué un `grid` avec lequel j'ai placé mes éléments. La gestion de la molette de la souris est quant à elle encore très aléatoire en fonction des `OS` (fonctionne correctement sous Windows mais pas sous Linux.).
2. **Affichage des images**.
   Les images ne s'affichaient pas car elles se faisaient détruire par le `garbage collector` de Python. Pour résoudre le problème, j'ai affecté les objets images comme des variables d'instance ; elles ne peuvent donc plus être supprimées par le gestionnaire de mémoire.
3. **Affichage liste des clés, super-clés**.
   Due au grand nombre de clés et super-clés à afficher, l'affichage peut prendre un certain temps à se générer. On crée un Label pour chaque attribut affiché, que l'on place ensuite dans un tableau (*grid*) ce qui a l'air de prendre du temps en *tkinter* (*tkinter* ne comportant pas de support spécifique pour des tableaux contenant du texte, j'ai choisi de garder l'option du *grid*).

0.5.2 Implémentation

J'ai été confronté à un problème concernant la transitivité de dépendance fonctionnelle. En effet, j'ai commencé le projet sans utiliser l'algorithme de `closure` et cela n'était pas suffisant (la transitivité de plus de deux DF en même temps n'était pas détectée.). J'ai donc décidé de l'implémenter afin de réduire le temps de calcul et d'avoir des résultats fiables.
Pour le **calcul des clés et super-clés**, nous avons eu des problèmes pour savoir si notre algorithme était fonctionnel du fait que le calcul de clés nous rapportait un grand nombre de clés et super-clés pendant les tests. On a donc décidé de demander au groupe de Julien Delplanque et Janou Brohée leurs

résultats en se basant sur des tests identiques, afin d'affiner nos algorithmes jusqu'à leur parfait fonctionnement.

0.6 Répartition du travail

Le projet a entièrement été réalisé par Maximilien Charlier. Corentin Ducruet s'étant chargé de prendre de l'avance sur le projet de génie logiciel. Les 10 premiers jours on étaient consacrés au développent des algorithmes et de la base du projet (qui était testée en parallèle).
Une fois la structure de base faite, je me suis attaqué au développement de la partie graphique qui, elle aussi, a duré environ 10 jours.

0.7 Apport du projet.

J'ai appris à utiliser des fonctions avancées de *git* en créant une branche "*beta*" lors du développement et en jouant avec les fusion de branche et les retours en arrière (`git reste -hard`).
J'ai remarqué que je n'avais pas bien compris la partie *Normalisation* du cours théorique, notamment ce qu'était une relation en BCNF/3NF, une DF singulière, la notion de clé/super-clé (Le cours comporte beaucoup de contre exemples, mais pas assez d'exemple, par exemple, on sait ce qu'est une relation n'étant pas en 3NF mais pas ce qu'est une relation en 3NF).
Concernant *python*, il s'avère que contrairement à ce que je pensais, il est très puissant et rapide. Le *non-typage* du langage me rebutait, mais ce n'est actuellement plus le cas. C'est en fait une grande puissance une fois maitrisée.
Le projet apporte aussi beaucoup du fait de l'utilisation d'*sqlite3* qui aide fortement pour son utilisation dans le projet de *génie logiciel* car on connait le fonctionnement des *Cursor*s ce qui facilite la compréhension en *java* (car beaucoup plus compliqué que en python).

0.8 Ressources utilisées

0.8.1 Logiciel

Sublime text (Windows et Unix) Pour la rédaction du code python.

Git for Windows

`Adobe©Photoshop©CC` (Licence au nom de Charlier Maximilien.)

TexStudio(Windows) pour la rédaction du rapport.

0.8.2 Git

Utilisation d'un serveur OVH mutualisé (pro) pour y placer un git barre (dépôt git distant) et y accéder avec une clé SSH.

0.8.3 Librairie Python

Python 2.7

sqlite3

OS Afin d'utiliser les bons types de chemin pour les fichiers.

Tkinter Environnement graphique.
Installation : Ubuntu : http://doc.ubuntu-fr.org/tkinter

P.I.L. Python Image Librairie (Pour afficher des images.)

0.8.4 Littérature

1. La chapitre 4 du livre "*The Theory of Relational Database*" de *D. Maier* Pour l'algorithme de *closure*.
2. Le cours théorique de *base de données 1* de *Jef WIJSEN*.
3. Le cours de SQL d*Alain BUYS*.
4. Pour la récupération du schéma d'une table : https://www.gaia-gis.it/spatialite-3.0.0-BETA/spatialite-cookbook-fr/html/tables.html
5. Chapitre 3 du cours de base de données de l'ULG disponible sur le site de l'ULG pour comprendre ce qu'était une relation BCNF.
6. Chapitre 7 du cours de base de données de l'ULB disponible sur le site de l'ULB pour la décomposition en 3NF.

0.8.5 Image

1. Logo de l'UMONS (issue du site de l'UMONS).
2. Logo de la faculté des Sciences (issue du site de la faculté des Sciences).
3. Image réalisé personnellement : FIGURE 11

0.9 Remerciement.

Je remercie **Charline Simons** pour la correction orthographique du rapport.

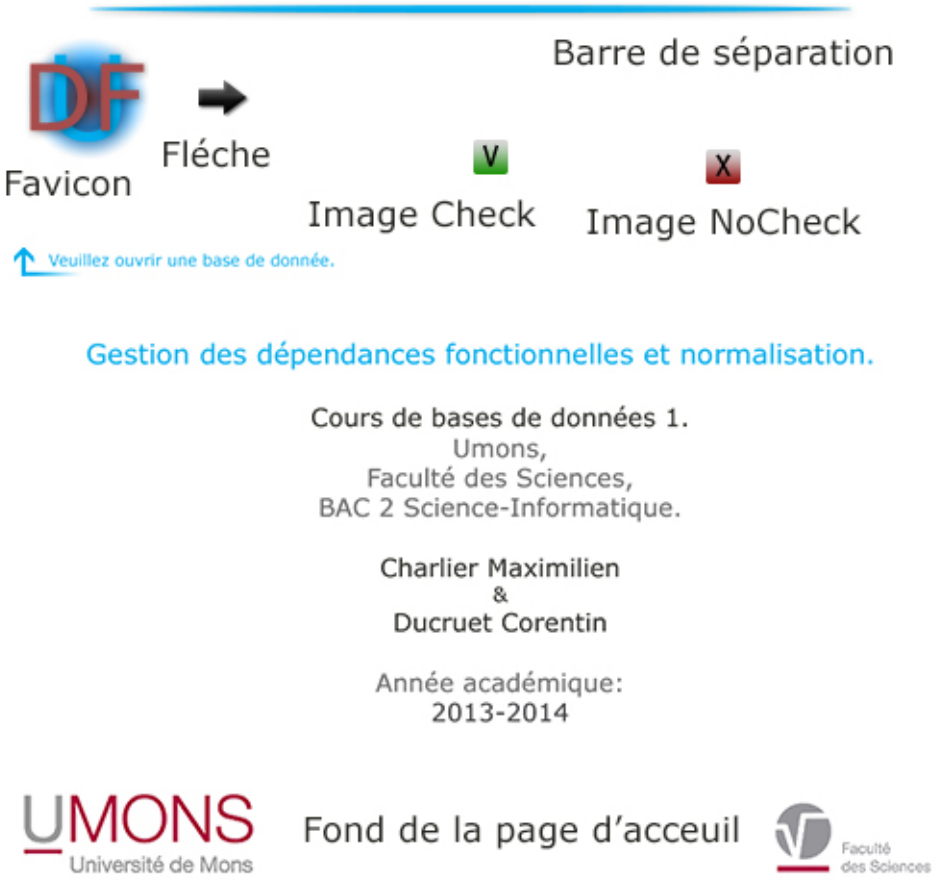

FIGURE 11 – Image utilisé pour l'interface graphique.