Université de Montpellier — Master Informatique

**Traitement sémantique des données (HAI824I)**

# Ontology-Based Data Access

# TP avec InteGraal

Ce TP est basé sur **InteGraal**, une bibliothèque java dédiée à l'intégration de données basée sur des règles logiques. InteGraal est développé par l'équipe BOREAL (Inria & LIRMM). Pour pouvoir effectuer ce TP sans devoir apprendre à utiliser les nombreux modules et classes d'InteGraal, nous allons passer par une **interface en ligne de commande (CLI)** qui offre les principales fonctionnalités d'InteGraal de façon simplifiée.

Le traitement de requêtes avec InteGraal peut reposer soit sur la matérialisation totale, soit sur la réécriture totale, soit sur des techniques mixtes. L'objectif de ce TP est que compreniez le fonctionnement de ces techniques.

**Pour en savoir plus sur InteGraal :**

- Dépôt Git : `https://gitlab.inria.fr/rules/integraal`
- Article démo à la conférence Bases de Données Avancées (BDA) 2023 : `https://hal-lirmm.ccsd.cnrs.fr/lirmm-04304601`

Nous prendrons comme exemple un système OBDA simplifié sur le thème de Star Wars intégrant deux bases de données :

- l'une consacrée aux personnages de Star Wars et aux objets que ces personnages utilisent (véhicules, sabres laser) ;
- l'autre consacrée aux films - cette deuxième base de données a été extraite de IMDb (`https://www.imdb.com/interfaces/`), avec des simplifications dans les tables choisies et une restriction aux films de Star Wars pour obtenir une petite base plus facile à explorer.

Les deux bases de données sont en **SQLite**, un système léger de gestion de BD relationnelles qui a l'avantage d'être intégré à Linux (et MacOS) et qui ne nécessite pas de service fonctionnant en arrière plan.

**Pour commencer :** téléchargez et décompressez le fichier TP_IG disponible sur moodle.

## 1 Structure du dossier TP_IG

On va manipuler différents types d'objets :

- des **bases de données** (ici en SQLite) ;
- des **mappings** qui permettent de passer du niveau des *données* au niveau de la *base de connaissances* (on appellera "vocabulaire ontologique" l'ensemble des prédicats utilisé dans la base de connaissances) ; d'un point de vue abstrait, chaque mapping est de la forme $q_s(x_1, \ldots, x_n) \to q_o(x_1, \ldots, x_n)$, où $q_s$ est une requête sur une base de données source et $q_o$ est une requête conjonctive utilisant le vocabulaire ontologique. Dans InteGraal, un mapping est décomposé en 2 :
  1. une **vue** (notons la $v_s$) associée à $q_s$ [mapping de bas niveau],
  2. une **règle** de la forme $v_s(x_1, \ldots, x_n) \to q_o(x_1, \ldots, x_n)$ [mapping de haut niveau] ;
- des **règles**, des **faits** et des **requêtes** utilisant le vocabulaire ontologique.

Les vues sont décrites dans des fichiers textes de suffixe `.vd`. Tous les autres éléments (règles, faits, et même des requêtes ontologiques) sont décrits dans des fichiers textes de suffixe `.dlgp` (pour "datalog +") ; dans un fichier dlgp on peut mélanger tous ces éléments, mais pour plus de clarté dans ce TP on les stockera dans différents fichiers dlgp.

▷ Ouvrez le répertoire `TP_IG` pour examiner la structure du système OBDA fourni :

- Dossier `sources` : contient les bases de données `sw_bank.db` sur les personnages de Star Wars et `sw_imdb.db` sur les films.
- `views_SW_BANK.vd` et `views_SW_BANK.vd` : les fichiers de vues sur les bases de données.
- `mappings.dlgp` : un fichier de mappings de haut niveau (liant les vues aux prédicats de l'ontologie).
- `onto_rules.dlgp` : un fichier de règles ontologiques.
- `q1.dlgp` et `q2.dlgp` : deux fichiers contenant chacun une requête utilisant le vocabulaire ontologique.

▷ Ouvrez les fichiers `.vd` et `.dlgp` et essayez d'en comprendre le contenu [1].

## 2 Les bases de données du TP

La base de données `sw_bank.db` contient 2 tables :

- `vehicle`
  - `character` : the name of the driving character
  - `vehicleType` : the type of vehicule driven by the character
- `lightsaber`
  - `character` : the name of the character using the weapon
  - `saber` : the name of the lightsaber
  - `color` : the color of the lightsaber

La base de données `sw_imdb.db` contient 3 tables :

- `title`
  - `tconst` : alphanumeric unique identifier of the title
  - `primaryTitle` : the more popular title
  - `startYear` : release year of a title
  - `runtimeMinutes` : primary runtime of the title, in minutes
- `person`
  - `nconst` : alphanumeric unique identifier of the name/person
  - `primaryName` : name by which the person is most often credited
  - `birthYear` : in YYYY format
  - `deathYear` : in YYYY format if applicable, else NULL
- `casting`
  - `tconst` : alphanumeric unique identifier of the title
  - `nconst` : alphanumeric unique identifier of the name/person
  - `category` : the category of job that person was in ('actor', 'actress', 'director' or 'composer')
  - `character` : the name of the character played if applicable, else NULL

▷ Vous pouvez explorer ces bases de données en utilisant SQLite, qui est en général déjà installé sous Linux et MacOS : tapez la commande `sqlite3` dans un terminal pour le vérifier.

1. Documentation provisoire ici : `https://rules.gitlabpages.inria.fr/integraal-website/features/`

**Tutoriel sur SQLite :** `https://www.tutorialspoint.com/sqlite/index.htm`

▷ En particulier, essayez ceci (en vous plaçant dans le répertoire qui contient la base de données) :

`sqlite3` (ceci ouvre l'invite de commande de SQLite : vous pouvez ensuite taper soit des commandes commençant par un point, soit des commandes SQL qui doivent impérativement se terminer par un point virgule)
`.open xxx.db` (ceci ouvre la base de données xxx ou la crée si elle n'existe pas)
`.tables` (ceci affiche les noms des tables de la base)
`.schema` (ceci affiche les requêtes SQL de création du schéma de la base et permet de voir quelles sont les tables avec les noms et types des attributs)
`.header on` (ceci met les noms des attributs quand les réponses sont affichées)
`SELECT * FROM table_name;` (en mettant le nom de la table voulue : ceci affiche le contenu de la table ; ne pas oublier le point virgule à la fin des commandes SQL)
`.quit` (ceci quitte SQLite)

# 3 Prise en main de l'environnement

▷ Téléchargez le jar exécutable du CLI et placez le dans le dossier `TP_IG` :
`https://gitlab.inria.fr/rules/integraal` : aller dans README.md, "Apps built on top of InteGraal" puis lien "CLI".

▷ Ouvrez un terminal de commandes et placez-vous dans le dossier `TP_IG` (ceci évitera d'avoir à indiquer un chemin d'accès aux fichiers lors des commandes).

▷ Démarrez le CLI : `java -jar integraal-cli.jar`
Pour voir les commandes disponibles, taper `help` et pour avoir plus d'information sur une commande taper `help` *`nom_commande`*.

Commençons par les commandes de l'environnement : en mémoire, on a des bases de faits, des bases de règles et des bases de requêtes. A un moment donné, il y a une base de faits courante, une base de règle courante et une base de requêtes courante, chacune pouvant être vide. Les options `-f=`, `-r=`, et `-q=` sont respectivement utilisées pour choisir une base de faits, une base de règles ou une base de requêtes. Ci-dessous on prend l'exemple de `-f=` mais on peut à la place (ou en plus) spécifier `-r=` ou `-q=`.

Voici un récapitulatif des principales commandes de manipulation du système :

`list` : liste les noms de bases en mémoire. Il peut y avoir plusieurs bases de chaque sorte en mémoire mais une seule courante (indiquée par *)

`peek` : donne un aperçu des 3 bases courantes (éventuellement vides).

`peek -f=`*`nom_base_faits`* : pour un aperçu de cette base uniquement *(attention pas d'espace avant ou après le =).*

`inspect -f=`*`nom_base_faits`* : pour voir en entier cette base (ou toutes les bases courantes, voir help inspect).

`create -f=`*`nom_base_faits`* : crée une base de faits vide.

`remove -f=`*`nom_base_faits`* : supprime la base de faits (de l'environnement, pas le fichier).

`set -f=`*`nom_base_faits`* : pour faire de cette base la base courante.

`mapping` *`fichier.vd`* `-f=`*`nom_base_faits`* : connecte le fichier de vue à la base de faits (qui doit déjà exister).

`load` *`fichier.dlgp`* `-f=`*`nom_base_faits`* : charge le fichier dans la base de faits (et la crée si elle n'existe pas).

▷ Commençons par charger les fichiers qui nous intéressent.

**1. Créer une base de faits vide (de nom "views") et la connecter aux vues**

```
create -f=views
mapping views_SW_BANK.vd  -f=views
mapping views_SW_IMDB.vd  -f=views
```

On aurait pu indiquer `set -f=views` après la création de "views" pour en faire la base de faits courante, ce qui aurait évité ensuite d'écrire `-f=views` ensuite.

Utilisez `list` et `peek` pour voir le résultat. La base de faits "views" est connectée aux vues mais elle est vide.

**2. Charger les 2 fichiers de règles dans deux bases de règles différentes,** pour bien distinguer les mappings de haut niveau et les règles ontologiques

```
load mappings.dlgp -r=mappings
load onto_rules.dlgp  -r=rules
```

Utilisez `list` et `peek` pour voir le résultat.

**Remarque** On aurait pu charger les deux fichiers de règles dans la même base de règles (qu'on pourrait appeler "all-rules") :

```
load mappings.dlgp onto_rules.dlgp -r=all_rules
```

Ou bien charger d'abord un fichier dans la base all-rules puis en ajouter un autre à cette même base avec l'option "append" :

```
load mappings.dlgp -r=all_rules
load onto_rules.dlgp -r=all_rules -m=append
```

**3. Evaluer une requête.** Chargeons par exemple la requête du fichier $q_1$ et tentons de l'évaluer :

```
load q1.dlgp -q=q1
evaluate -f=views
```

La requête $q_1$ n'a aucune réponse, ce qui est normal puisque la base de faits, bien que connectée à des vues, est toujours vide.

# 4 Raisonnements : saturation et réécriture

## 4.1 Saturation

**4. Activer les mappings pour peupler la base de faits initiale** .

On va d'abord activer les mappings de notre base de règles "mappings" :

- ceux associés à SW_BANK qui produisent des faits de prédicat *pilot*, *hasSaber* et *hasColor*.
- ceux associés à SW_IMDB qui produisent des faits de prédicat *hasName* et *hasTitle*.

Vérifiez que votre base de faits courante est "views" puis activez les règles de mappings :

```
saturate -r=mappings -o=materialized
```

On utilise la commande "saturate" qui sert à saturer une base de faits avec des règles, mais ici c'est un peu particulier : la base de faits est implicitement définie par l'ensemble des réponses aux vues mais on ne stocke pas les faits définis sur les prédicats de vues. On ne met dans la base de faits que les faits qui utilisent le vocabulaire ontologique (c'est-à-dire ceux produits par les règles de la base "mappings". Ici, on a mis le résultat dans une base "materialized" (si on n'indique pas `-o`, les faits produits sont ajoutés à la base courante).

Voyons ce que contient la base "materialized" :

```
list
peek -f=materialized
```

Pour voir la base en entier, on peut soit utiliser `inspect -f=materialized` (ouvre un "pager" qu'on quitte avec " :q"), soit la sauvegarder dans un fichier (puis ouvrir ce fichier) : `save materialized.dlgp -f=materialized`

**Remarque.** On peut aussi sauvegarder toutes les bases courantes dans un seul fichier dlgp : `save toute_ma_kb.dlgp`

**5. Saturer la base de faits obtenue avec les règles ontologiques.** On met le résultat dans une base de faits appelée "saturated".

```
saturate -f=materialized -r=rules -o=saturated
```

**6. Evaluer des requêtes sur la base de faits saturée.** Assurez vous que $q_1$ soit la requête courante. La commande `evaluate -f=saturated` doit maintenant trouver 5 réponses.

**Remarque** La commande `assert` permet d'ajouter un fait, une règle ou une requête à la base courante, ou dans une autre base, qu'on crée éventuellement. Voici des façons alternatives d'évaluer $q_1$ (utilisez `peek` pour voir ce qui se passe).

- On asserte $q_1$ dans la base courante :
  ```
  assert "?(X) :- hasSaber(X,Y)."
  evaluate -f=saturated
  ```
- On asserte $q_1$ dans une base q1 :
  ```
  assert -q=q1  "?(X) :- hasSaber(X,Y)."
  evaluate -q=q1 -f=saturated
  ```

▷ Procédez de même avec la requête $q_2$ : `?(X) :- jedi(X).` Sur la base matérialisée, vous n'obtiendrez aucune réponse, mais vous devriez en obtenir 4 sur la base saturée.

**Réécriture** On va maintenant évaluer les requêtes avec une approche de virtualisation. Vous pouvez garder vos bases de faits existantes, mais on n'utilisera que la base de faits `views`.

**7. Réécrire une requête avec les règles ontologiques puis les règles de mapping** Pour illustrer la réécriture de requête, on va réécrire $q_2$ en deux étapes : d'abord avec les règles ontologiques, puis avec les règles de mappings. Ensuite, on évaluera la réécriture obtenue sur la base de faits virtuelle (autrement dit, en exécutant directement les requêtes SQL sur les bases de données). Voici le détail de différentes étapes :

1. Charger `q2.dlgp` ou asserter `"?(X) :- jedi(X)."` et l'appeler q2.
2. S'assurer avec `list` que q2 est la base de requêtes courante
   (au besoin faire `set -q=q2`).
3. Réécrire q2 avec la base de règles "rules" en mettant le résultat dans une requête q2-rew-rules :
   `rewrite -r=rules -o=q2-rew-rules`
4. Voir que l'évaluation de q2-rew-rules sur la base de faits views ne donne rien (l'option `-u` dit que la requête est une UCQ et pas juste un ensemble de CQ) :
   `evaluate -q=q2-rew-rules -f=views -u`
5. Réécrire q2-rew-rules avec la base de règles "mappings" en mettant le résultat dans une requête q2-rew-rules-mappings
6. Examiner le résultat (avec `inspect`) : on a 13 CQ.

**8. Evaluer la réécriture obtenue sur la base de faits virtuelle** Evaluer la requête q2-rew-rules-mappings sur la base de faits "views" : `evaluate` vous permet d'obtenir les réponses à chacune des CQ (voir que seules celles qui ne comportent que des prédicats de vues ont une réponse) et `evaluate -u` évalue la requête globalement comme une UCQ. Vous devez retrouver les mêmes réponses qu'à l'étape 6.

▷ Ecrivez les requêtes qui retrouvent tous les sith (*sith(X)*) ou les sages jedi (*sageJedi(X)*) et répondez à ces requêtes avec les deux techniques : soit en les évaluant sur la base de faits saturée soit en les réécrivant puis en évaluant leur réécriture sur la base de faits virtuelle.

# 5 A vous de jouer !

Vous allez maintenant étendre le système d'intégration.

## 5.1 Complétion des classes de personnages

Les règles du fichier `onto_rules.dlgp` introduisent les classes *jedi*, *sageJedi* et *sith*.

▷ Ajoutez une classe *character* (personnage) et les règles qui définissent les relations de spécialisation entre ces classes : les jedi et les sith sont des types de personnages, ainsi que les jedi sages qui sont aussi des jedi.

▷ Mettez à jour la base "rules" en rechargeant le fichier `onto_rules.dlgp` (supprimer la base avec `remove`, puis charger à nouveau le fichier avec `load`). Répondez à la requête : quels sont tous les personnages ? (avec la méthode que vous voulez).

## 5.2 Premier ajout de mappings

Nous nous intéressons maintenant aux mappings depuis la base de données `sw_imdb.db`. On veut d'abord récupérer l'année de sortie des films.

▷ Ajoutez une vue `view_year` dans le fichier `views_SW_IMDB.vd` qui permet d'extraire l'année de sortie d'un film : voir les attributs `tconst` et `startYear` de la table `title`. L'argument correspondant à l'année sera de type `xsd:integer`.

▷ Ajoutez une règle de mapping dans le fichier `mappings.dlgp` qui à partir de cette vue crée un prédicat hasYear/2.

▷ Rechargez le fichier de vues et le fichier de règles de mappings.

▷ Répondez à la requête suivante qui demande tous les titres de films sortis en 1977 :
`?(T):- hasTitle(IDF,T), hasYear(IDF,1977).`
Vous devez obtenir 1 réponse.

### 5.3 Lier les acteurs aux films

▷ Ajoutez de quoi peupler le prédicat binaire castsIn(IDA,IDF) où IDA est un identifiant d'acteur ou actrice et IDF est un identifiant de film. Vous pouvez utiliser dans la requête SQL la syntaxe suivante : `category in ('actor', 'actress')` pour éviter d'avoir à créer deux vues.

▷ Répondez à une requête qui demande les noms des acteurs qui ont joué dans un (le) film sorti en 1977.

### 5.4 Lier les acteurs aux personnages

▷ Ajoutez de quoi peupler le prédicat plays(IDA, P) où IDA est un identifiant d'acteur ou actrice et P est un personnage.

▷ Répondez aux requêtes suivantes :

- Quels sont les (noms des) acteurs qui ont joué des jedi ?
- Quels sont les (noms des) acteurs qui jouent Dark Wador (Darth Wader en anglais et "darth-vader" dans la base de données).

### 5.5 Lier les personnages aux films

On veut retrouver quels personnages apparaissent dans quels (titres de) films. Pourquoi la requête suivante n'est-elle pas correcte (au sens où elle ne retourne pas ce qu'il faut) :

```
?(P,T) :- plays(IDA,P), castsIn(IDA,IDF), hasTitle(IDF,T).
```

Modifiez votre système de façon à pouvoir répondre correctement à ce type de requête (qui demande à connaître les liens entre personnages et films).