Groupe de recherche interdisciplinaire en informatique de la santé

Documentation SES - Modèles de Conception de Requêtes (MCR)

OntoRelQuery

MCR

Mohamed Amin GAIED *(Mohamed.Amin.Gaied@USherbrooke.ca)*

—

MAD/«depot»/MCR_SES, version 1.0.0.a, en date du 2025-06-18

Sommaire

Le composant MCR est un élément principal d'OntoRelQuery, permettant de concevoir et d'exécuter des requêtes ontologiques de manière structurée et réutilisable.

Cette documentation technique (SES) détaille l'architecture, l'implémentation et l'extension des MCR. Pour la documentation fonctionnelle et des exemples d'utilisation, consultez la Documentation fonctionnelle des MCR (SDC).

1. Introduction

Cette documentation technique détaille l'architecture, l'implémentation et l'extension des MCR en OntoRelQuery. Les MCR sont des modèles prédéfinis qui permettent de concevoir et d'exécuter des requêtes ontologiques de manière structurée et réutilisable.

1.1. Objectif

Les MCR ont été conçus pour:

- Standardiser les requêtes fréquemment utilisées
- Simplifier la construction de requêtes complexes
- Permettre la réutilisation de patrons de requêtes
- Offrir une abstraction de haut niveau pour l'interrogation des données ontologiques
- Faciliter la maintenance et l'évolution des requêtes

1.2. Positionnement dans l'architecture

Les MCR constituent un composant important, servant d'interface entre:

- Les graphes ontologiques (OntoGraph) - Représentation des concepts et relations ontologiques
- Les graphes relationnels (OntoRelGraph) - Mapping entre ontologie et schéma relationnel
- Le catalogue OntoRel (OntoRelCat) - Métadonnées sur les mappings ontologie-relationnel
- Le système de génération de requêtes SQL - Transformation des requêtes ontologiques en SQL

2. Architecture du module MCR

2.1. Vue d'ensemble

Le module MCR est organisé autour de plusieurs composants :

- **QueryModel** (`ca.griis.ontorelquery.model.query.QueryModel`) : Énumération définissant les différents types de modèles disponibles
- **QueryModelStrategy** (`ca.griis.ontorelquery.service.construction.QueryModelStrategy`) : Interface définissant la stratégie de construction de requêtes
- **DynamicQueryBuilder** (`ca.griis.ontorelquery.service.construction.DynamicQueryBuilder`) : Point d'entrée pour la construction de requêtes
- **Implémentations de stratégies** : Classes concrètes implémentant les différentes stratégies de construction
- `SimpleModelStrategy` - Pour les modèles simples (MS_*)
- `IterativeModelStrategy` - Pour les modèles itératifs (MI_*)
- `CombinatorialModelStrategy` - Pour les modèles combinatoires (MC_*)

2.2. Diagramme de classes

[diagram class mcr] | *figures/diagram_class_mcr.png*

```
classDiagram
    class QueryModel {
        <<enumeration>>
        +String description
        +List~String~ requiredOptions
        +boolean isAvailable
        +String getDescription()
        +List~String~ getRequiredOptions()
        +boolean isAvailable()
        +static boolean isValidQueryModel(String)
        +static QueryModel fromString(String)
    }

    class QueryModelStrategy {
        <<interface>>
        +Query buildCatalogQuery(Map~String,String~)
        +Query buildOntoRelQuery(Map~String,String~, List~String~)
    }

    class SimpleModelStrategy {
        +Query buildCatalogQuery(Map~String,String~)
        +Query buildOntoRelQuery(Map~String,String~, List~String~)
        -Query buildMsScQuery(Map~String,String~)
        -Query buildMsScpoQuery(Map~String,String~)
        -Query buildMsScpdQuery(Map~String,String~)
        -Query buildMsSchQuery(Map~String,String~)
    }

    class IterativeModelStrategy {
        +Query buildCatalogQuery(Map~String,String~)
        +Query buildOntoRelQuery(Map~String,String~, List~String~)
        -Query buildMiApQuery(Map~String,String~)
        -Query buildMiSpQuery(Map~String,String~)
        -Query buildMiPpoQuery(Map~String,String~)
    }

    class CombinatorialModelStrategy {
        +Query buildCatalogQuery(Map~String,String~)
        +Query buildOntoRelQuery(Map~String,String~, List~String~)
        -List~QueryModelStrategy~ createSubStrategies(String)
    }
```

```
    class QueryModelStrategyFactory {
        -Map~String,Class~ strategyClasses
        +QueryModelStrategy createStrategy(String, Map~String,String~)
    }

    class DynamicQueryBuilder {
        -DSLContext dsl
        -Map~String,String~ options
        -String modelType
        -String ontorelSchema
        -String schema
        -String ontoreluuid
        -OntoGraph ontoGraph
        -OntoRelGraph ontoRelGraph
        -String queryLanguage
        -QueryModelStrategy strategy
        +Query buildCatalogQuery()
        +Query buildOntoRelQuery(List~String~)
        +List~String~ extractTableIds(List~String~)
        +List~String~ getTableColumns(String)
    }

    class ExecuteQueryCommand {
        -QueryExecutor queryExecutor
        -ResultExporter resultExporter
        -DynamicQueryBuilder queryBuilder
        +Integer call()
        -List~String~ handleMi(int, DynamicQueryBuilder, OntoGraph, OntoRelGraph)
        -OntoGraph loadOntoGraph()
        -OntoRelGraph loadOntoRelGraph()
        -Map~String,String~ collectOptions(QueryModel)
    }

    class QueryExecutor {
        -DSLContext dsl
        +QueryResult execute(Query)
        +QueryResult executeAndFetch(Query)
    }

    class ResultExporter {
        +void exportResults(QueryResult, String)
        +void exportSql(Query, String)
    }

    QueryModelStrategy <|.. SimpleModelStrategy
    QueryModelStrategy <|.. IterativeModelStrategy
    QueryModelStrategy <|.. CombinatorialModelStrategy
    QueryModelStrategyFactory --> QueryModelStrategy : creates
    DynamicQueryBuilder --> QueryModelStrategy : uses
    DynamicQueryBuilder --> QueryModelStrategyFactory : uses
    ExecuteQueryCommand --> DynamicQueryBuilder : uses
    ExecuteQueryCommand --> QueryExecutor : uses
    ExecuteQueryCommand --> ResultExporter : uses
```

2.3. Classes principales et leurs responsabilités

2.3.1. QueryModel

La classe `QueryModel` est une énumération qui définit tous les modèles de requêtes disponibles dans le système. Chaque modèle a une description, une liste d'options requises et un indicateur de disponibilité.

Responsabilités:

- Définir les types de modèles disponibles
- Fournir des métadonnées sur chaque modèle (description, options requises)
- Valider les noms de modèles

2.3.2. QueryModelStrategy

L'interface `QueryModelStrategy` définit le contrat pour toutes les stratégies de construction de requêtes. Elle déclare deux méthodes principales:

- `buildCatalogQuery` : Construit une requête pour interroger le catalogue OntoRelCat
- `buildOntoRelQuery` : Construit une requête pour interroger les données relationnelles en OntoRel

Responsabilités:

- Définir l'interface commune pour toutes les stratégies
- Permettre le polymorphisme dans la construction de requêtes

2.3.3. DynamicQueryBuilder

La classe `DynamicQueryBuilder` est le point d'entrée principal pour la construction de requêtes. Elle utilise le pattern Strategy pour déléguer la construction aux implémentations spécifiques.

Responsabilités:

- Sélectionner la stratégie appropriée en fonction du type de modèle
- Coordonner le processus de construction de requêtes
- Gérer les paramètres et options de requête
- Extraire les identifiants de tables à partir des résultats de requêtes

2.3.4. SimpleModelStrategy

La classe `SimpleModelStrategy` implémente la stratégie de construction pour les modèles simples (MS_*).

Responsabilités:

- Construire des requêtes pour les modèles simples
- Gérer les spécificités de chaque type de modèle simple

2.3.5. IterativeModelStrategy

La classe `IterativeModelStrategy` implémente la stratégie de construction pour les modèles itératifs (MI_*).

Responsabilités:

- Construire des requêtes basées sur des chemins entre classes ontologiques
- Gérer l'interaction avec l'utilisateur pour la sélection de chemins

2.3.6. CombinatorialModelStrategy

La classe `CombinatorialModelStrategy` implémente la stratégie de construction pour les modèles combinatoires (MC_*).

Responsabilités:

- Construire des requêtes combinant plusieurs sous-modèles
- Gérer les dépendances entre sous-modèles
- Fusionner les résultats des sous-requêtes

2.3.7. QueryModelStrategyFactory

La classe `QueryModelStrategyFactory` est responsable de la création des instances de stratégies appropriées.

Responsabilités:

- Créer des instances de stratégies en fonction du type de modèle
- Maintenir une map des types de modèles vers les classes de stratégie

2.3.8. ExecuteQueryCommand

La classe `ExecuteQueryCommand` est responsable de l'exécution des requêtes via l'interface en ligne de commande.

Responsabilités:

- Collecter les paramètres de requête auprès de l'utilisateur
- Charger les graphes ontologiques et relationnels
- Coordonner le processus d'exécution de requête
- Exporter les résultats

2.3.9. QueryExecutor

La classe `QueryExecutor` est responsable de l'exécution des requêtes SQL.

Responsabilités:

- Exécuter les requêtes SQL via JOOQ
- Transformer les résultats en objets Java

2.3.10. ResultExporter

La classe `ResultExporter` est responsable de l'exportation des résultats de requêtes.

Responsabilités:

- Exporter les requêtes SQL dans des fichiers
- Exporter les résultats dans des fichiers

2.4. Flux de données

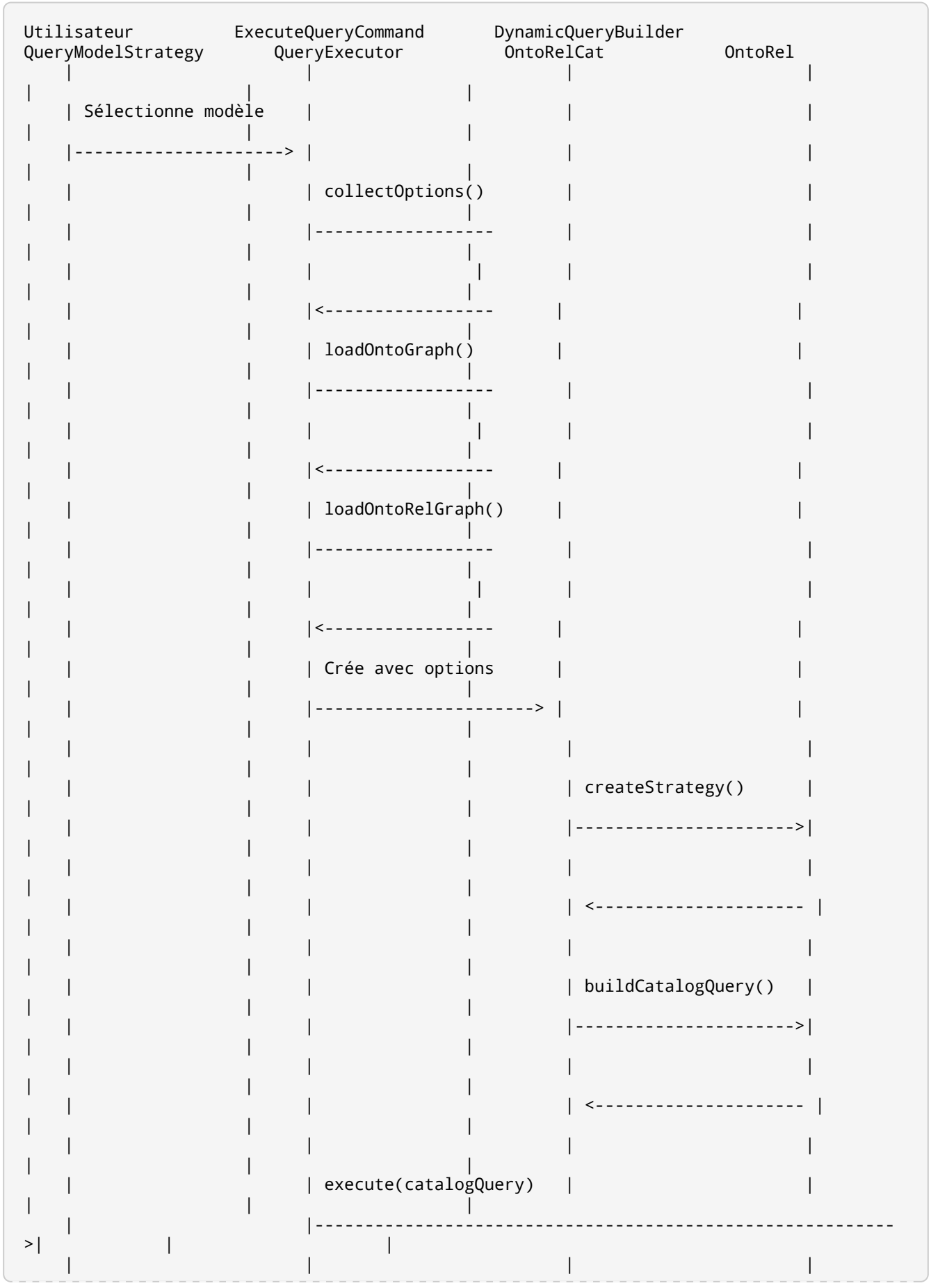
Le flux de données MCR suit généralement ces étapes:

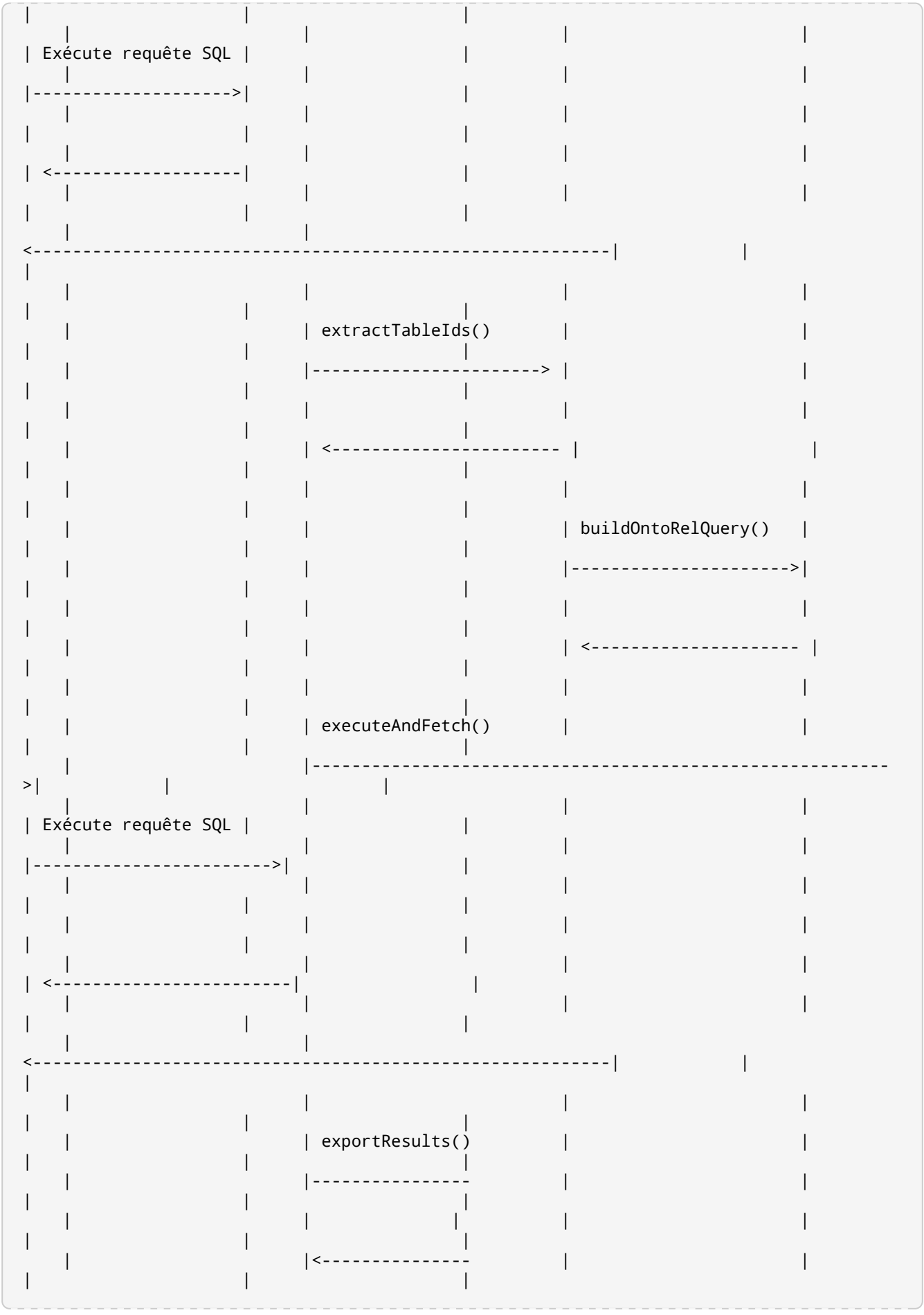
1. **Sélection du modèle** : L'utilisateur choisit un modèle de requête (MS_SC, MI_SP, etc.)
2. **Collecte des paramètres** : Les paramètres requis pour le modèle sont collectés
3. **Construction de la requête catalogue** : Une première requête est construite pour interroger le catalogue OntoRelCat
4. **Exécution de la requête catalogue** : Cette requête est exécutée pour obtenir les identifiants de tables
5. **Construction de la requête principale** : Une seconde requête est construite en utilisant les identifiants de tables
6. **Exécution de la requête principale** : Cette requête est exécutée pour obtenir les résultats en OntoRel
7. **Traitement des résultats** : Les résultats sont formatés et présentés à l'utilisateur

2.4.1. Diagramme de séquence système

Le diagramme suivant illustre les interactions entre les différents composants du système MCR lors de

l’exécution d’une requête:




Exécute requête SQL
extractTableIds()
buildOntoRelQuery()
executeAndFetch()
Exécute requête SQL
exportResults()

Ce diagramme illustre les différents flux d'exécution selon le type de modèle de requête (simple, itératif ou combinatoire). Pour chaque type, le système suit un processus spécifique pour construire et exécuter les requêtes, tout en gérant les interactions avec l'utilisateur lorsque nécessaire (notamment pour les modèles itératifs qui peuvent nécessiter une sélection de chemin).

3. Classification des MCR

Cette section présente un catalogue détaillé de tous les modèles de conception de requêtes (MCR) disponibles, avec leurs paramètres, significations, classes et méthodes de construction.

3.1. Modèles simples (MS_*)

Les modèles simples permettent de sélectionner des instances de classes ontologiques avec diverses contraintes.

3.1.1. MS_SC : Sélection simple de classe

Paramètre	Signification	Type	Obligatoire	Exemple
class	Nom de la classe ontologique à sélectionner	String	Oui	"Patient"

Classe de construction : `SimpleModelStrategy`

Méthode de construction : `buildMsScQuery(Map<String, String> options)`

Description : Ce modèle permet de sélectionner toutes les instances d'une classe ontologique spécifiée. Il génère une requête SQL qui sélectionne toutes les colonnes de la table correspondant à la classe ontologique.

Exemple de requête catalogue générée :

```
SELECT table_id
FROM Onto_class
WHERE iri = 'http://purl.obolibrary.org/obo/HOSO_0000060'
AND ontorel_uuid = 'da601f3e-729d-40b0-a76d-1661f875f375'
AND label = 'en';
```

Exemple de requête principale générée :

```
SELECT HOSO_0000060_uid as 'public provincial health insurance record'
FROM HOSO_0000060;
```

3.1.2. MS_SCPO : Sélection de classe par propriété d'objet

Paramètre	Signification	Type	Obligatoire	Exemple
class	Nom de la classe ontologique à sélectionner	String	Oui	"Patient"
objectProperty	Nom de la propriété d'objet à utiliser	String	Oui	"hasMedication"

Classe de construction : `SimpleModelStrategy`

Méthode de construction : `buildMsScpoQuery(Map<String, String> options)`

Description : Ce modèle permet de sélectionner des instances d'une classe ontologique liées à d'autres instances par une propriété d'objet. Il génère une requête SQL qui joint la table de la classe avec la table de la propriété d'objet.

Exemple de requête catalogue générée :

```
SELECT table_id, label
FROM Onto_class_axiom
WHERE ontorel_uuid= 'da601f3e-729d-40b0-a76d-1661f875f375'
AND domain_iri = 'http://purl.obolibrary.org/obo/HOSO_0000060'
AND property_iri = 'http://purl.obolibrary.org/obo/IAO_0000136'
AND range_iri = 'http://purl.obolibrary.org/obo/HOSO_0000062'
AND label = 'en';
```

Exemple de requête principale générée :

```
SELECT HOSO_0000060_uid 'public provincial health insurance record',
       HOSO_0000062_uid 'primary human health insurance beneficiary'
FROM "HOSO_0000060"
JOIN "HOSO_0000060_IAO_0000136_HOSO_0000062" ON ("HOSO_0000060_uid")
JOIN "HOSO_0000062" ON ("HOSO_0000062_uid");
```

3.1.3. MS_SCPD : Sélection de classe par propriété de données

Paramètre	Signification	Type	Obligatoire	Exemple
class	Nom de la classe ontologique à sélectionner	String	Oui	"Patient"
dataProperty	Nom de la propriété de données à utiliser	String	Oui	"hasAge"

Classe de construction : `SimpleModelStrategy`

Méthode de construction : `buildMsScpdQuery(Map<String, String> options)`

Description : Ce modèle permet de sélectionner des instances d'une classe ontologique avec leurs propriétés de données. Il génère une requête SQL qui sélectionne la table de la classe et la colonne correspondant à la propriété de données.

Exemple de requête catalogue générée :

```
SELECT table_id, label
FROM Onto_data_axiom
WHERE ontorel_uuid= 'da601f3e-729d-40b0-a76d-1661f875f375'
AND domain_iri = 'http://purl.obolibrary.org/obo/HOSO_0000019'
AND property_iri = 'http://purl.obolibrary.org/obo/OpenLHS-Core_0000059'
AND label = 'en';
```

Exemple de requête principale générée :

```
SELECT HOSO_0000019_uid 'public provincial health identifier',
       OpenLHS-Core_0000054_uid 'has value',
       OpenLHS-Core_0000054_OpenLHS-Core_0000059_Literal '@value'
FROM "HOSO_0000019"
JOIN "OpenLHS-Core_0000054" ON "OpenLHS-Core_0000054_uid" =
"HOSO_0000019"."HOSO_0000019_uid"
JOIN "OpenLHS-Core_0000054_OpenLHS-Core_0000059_Literal" ON "OpenLHS-
Core_0000054"."OpenLHS-Core_0000054_uid" = "OpenLHS-Core_0000054_uid";
```

3.1.4. MS_SCH : Sélection de classe par hiérarchie

Paramètre	Signification	Type	Obligatoire	Exemple
class	Nom de la classe ontologique à sélectionner	String	Oui	"Medication"
hierarchy	Type de relation hiérarchique	String	Oui	"subClassOf"

Classe de construction : `SimpleModelStrategy`

Méthode de construction : `buildMsSchQuery(Map<String, String> options)`

Description : Ce modèle permet de sélectionner des instances d'une classe ontologique en tenant compte des relations hiérarchiques. Il génère une requête SQL qui sélectionne les instances de la classe et de ses sous-classes ou super-classes.

Exemple de requête catalogue générée :

```
SELECT table_id, label
FROM Onto_class_inheritance
WHERE ontorel_uuid='da601f3e-729d-40b0-a76d-1661f875f375'
AND class_iri = 'http://purl.obolibrary.org/obo/OpenLHS-Core_0000065'
AND inheritance_level = 2
AND parcours = 'C'
AND label = 'en';
```

Exemple de requête principale générée :

```
SELECT t1.OpenLHS-Core_0000065_uid 'temporal information',
       t2.IOIO_0000005_uid 'human birth temporal information',
       t3.HDRN_0000003_uid 'HDRN human birth temporal information'
FROM "OpenLHS-Core_0000065" t1
JOIN "IOIO_0000005" t2 ON t1.OpenLHS-Core_0000065_uid = t2.IOIO_0000005_uid
JOIN "HDRN_0000003" t3 ON t2.IOIO_0000005_uid = t3.HDRN_0000003_uid;
```

3.1.5. MS_SCA : Sélection de classe avec agrégation

Paramètre	Signification	Type	Obligatoire	Exemple
class	Nom de la classe ontologique à sélectionner	String	Oui	"Patient"
aggregationColumn	Colonne à agréger	String	Oui (collecté interactivement)	"patient_id"
aggregationFunction	Fonction d'agrégation à appliquer	String	Oui (collecté interactivement)	"COUNT"

Classe de construction : `SimpleModelStrategy`

Méthode de construction : `buildMsScaQuery(Map<String, String> options)`

Description : Ce modèle permet de sélectionner des instances d'une classe ontologique et d'appliquer des fonctions d'agrégation. Il génère une requête SQL qui applique une fonction d'agrégation sur une colonne de la table correspondant à la classe.

Exemple de requête catalogue générée :

```
SELECT table_id, label
FROM Onto_data_axiom
WHERE ontorel_uuid= 'da601f3e-729d-40b0-a76d-1661f875f375'
AND domain_iri = 'http://purl.obolibrary.org/obo/HDRN_0000003'
AND property_iri = 'http://purl.obolibrary.org/obo/HDRN_0000001'
AND label = 'en';
```

Exemple de requête principale générée :

```
SELECT PDRO_0000042.PDRO_0000042_uid as 'drug dispensing record',
       count(PDRO_0000041.PDRO_0000041_uid) as 'drug dispensing record item'
FROM PDRO_0000042
JOIN PDRO_0000042_BFO_0000051_PDRO_0000041 ON PDRO_0000042.PDRO_0000042_uid =
PDRO_0000042_BFO_0000051_PDRO_0000041.PDRO_0000042_uid
JOIN PDRO_0000041 ON PDRO_0000042_BFO_0000051_PDRO_0000041.PDRO_0000041_uid =
PDRO_0000041.PDRO_0000041_uid
GROUP BY PDRO_0000042.PDRO_0000042_uid;
```

3.1.6. MS_SCF : Sélection de classe avec filtre

Paramètre	Signification	Type	Obligatoire	Exemple
class	Nom de la classe ontologique à sélectionner	String	Oui	"Patient"
filterColumn	Colonne à filtrer	String	Oui (collecté interactivement)	"birth_date"
filterCondition	Condition de filtrage	String	Oui (collecté interactivement)	"> '1980-01-01'"

Classe de construction : `SimpleModelStrategy`

Méthode de construction : `buildMsScfQuery(Map<String, String> options)`

Description : Ce modèle permet de sélectionner des instances d'une classe ontologique en appliquant un filtre. Il génère une requête SQL qui sélectionne les instances de la classe qui satisfont une condition de filtrage.

Exemple de requête catalogue générée :

```
SELECT table_id, label
FROM Onto_data_axiom
WHERE ontorel_uuid= 'da601f3e-729d-40b0-a76d-1661f875f375'
AND domain_iri = 'http://purl.obolibrary.org/obo/HDRN_0000003'
AND property_iri = 'http://purl.obolibrary.org/obo/HDRN_0000001'
AND label = 'en';
```

Exemple de requête principale générée :

```
SELECT HDRN_0000003_uid as 'HDRN human birth temporal information',
       HDRN_0000003_HDRN_0000001_dateTime_value as 'date of birth'
FROM "HDRN_0000003"
JOIN "HDRN_0000003_HDRN_0000001_dateTime" ON "HDRN_0000003_uid" =
"HDRN_0000003_HDRN_0000001_dateTime"."HDRN_0000003_uid"
WHERE "HDRN_0000003_HDRN_0000001_dateTime_value" between '2023-01-01' and '2023-12-31';
```

3.1.7. MS_SCPOC : Sélection de classe par propriété d'objet associée à une autre classe

Paramètre	Signification	Type	Obligatoire	Exemple
domain	Nom de la classe ontologique source	String	Oui	"Patient"
objectProperty	Nom de la propriété d'objet à utiliser	String	Oui	"hasMedication"
range	Nom de la classe ontologique cible	String	Oui	"Medication"

Classe de construction : `SimpleModelStrategy`

Méthode de construction : `buildMsScpocQuery(Map<String, String> options)`

Description : Ce modèle permet de sélectionner des instances d'une classe ontologique liées à des instances d'une autre classe par une propriété d'objet. Il génère une requête SQL qui joint les tables des deux classes via la table de jointure correspondant à la propriété d'objet.

Exemple de requête catalogue générée :

```
SELECT table_id, label
FROM Onto_class_axiom
WHERE ontorel_uuid= 'da601f3e-729d-40b0-a76d-1661f875f375'
AND domain_iri = 'http://purl.obolibrary.org/obo/PDRO_0000042'
AND property_iri = 'http://purl.obolibrary.org/obo/BFO_00000051'
AND label = 'en';
```

Exemple de requête principale générée :

```
SELECT table_id, label
FROM Onto_class_axiom
WHERE ontorel_uuid= 'da601f3e-729d-40b0-a76d-1661f875f375'
AND domain_iri = 'http://purl.obolibrary.org/obo/PDRO_0000042'
AND property_iri = 'http://purl.obolibrary.org/obo/BFO_00000051'
AND label = 'en';
```

3.2. Modèles itératifs (MI_*)

Les modèles itératifs permettent de construire des requêtes basées sur des chemins entre classes ontologiques.

3.2.1. MI_AP : Tous les chemins possibles entre deux classes

Paramètre	Signification	Type	Obligatoire	Exemple
from	Nom de la classe ontologique source	String	Oui	"Patient"

Paramètre	Signification	Type	Obligatoire	Exemple
to	Nom de la classe ontologique cible	String	Oui	"Medication"
selectedPath	Chemin sélectionné par l'utilisateur	String	Non (collecté interactivement)	"Patient → hasMedication → Medication"

Classe de construction : `IterativeModelStrategy`

Méthode de construction : `buildMiApQuery(Map<String, String> options)`

Description : Ce modèle permet de trouver tous les chemins possibles entre deux classes ontologiques et de sélectionner un chemin pour construire une requête. Il utilise l'algorithme de recherche en profondeur (DFS) pour trouver tous les chemins possibles, puis construit une requête SQL basée sur le chemin sélectionné par l'utilisateur.

Exemple de requête principale générée :

```
SELECT p.HOSO_0000060_uid as 'public provincial health insurance record',
       b.HOSO_0000062_uid as 'primary human health insurance beneficiary'
FROM "HOSO_0000060" p
JOIN "HOSO_0000060_IAO_0000136_HOSO_0000062" j ON j.HOSO_0000060_uid =
p.HOSO_0000060_uid
JOIN "HOSO_0000062" b ON b.HOSO_0000062_uid = j.HOSO_0000062_uid;
```

3.2.2. MI_SP : Chemin le plus court entre deux classes

Paramètre	Signification	Type	Obligatoire	Exemple
from	Nom de la classe ontologique source	String	Oui	"Patient"
to	Nom de la classe ontologique cible	String	Oui	"Medication"

Classe de construction : `IterativeModelStrategy`

Méthode de construction : `buildMiSpQuery(Map<String, String> options)`

Description : Ce modèle permet de trouver le chemin le plus court entre deux classes ontologiques et de construire une requête basée sur ce chemin. Il utilise l'algorithme de Dijkstra pour trouver le chemin le plus court, puis construit une requête SQL basée sur ce chemin.

Exemple de requête principale générée :

```
SELECT p.HOSO_0000060_uid as 'public provincial health insurance record',
       b.HOSO_0000062_uid as 'primary human health insurance beneficiary'
FROM "HOSO_0000060" p
JOIN "HOSO_0000060_IAO_0000136_HOSO_0000062" j ON j.HOSO_0000060_uid =
p.HOSO_0000060_uid
JOIN "HOSO_0000062" b ON b.HOSO_0000062_uid = j.HOSO_0000062_uid;
```

3.2.3. MI_PPO : Chemin par propriété d'objet spécifique

Paramètre	Signification	Type	Obligatoire	Exemple
from	Nom de la classe ontologique source	String	Oui	"Patient"
to	Nom de la classe ontologique cible	String	Oui	"Doctor"

Paramètre	Signification	Type	Obligatoire	Exemple
objectProperty	Nom de la propriété d'objet à utiliser	String	Oui	"hasDoctor"

Classe de construction : `IterativeModelStrategy`

Méthode de construction : `buildMiPpoQuery(Map<String, String> options)`

Description : Ce modèle permet de trouver un chemin entre deux classes ontologiques en utilisant une propriété d'objet spécifique. Il construit une requête SQL qui joint les tables des deux classes via la table de jointure correspondant à la propriété d'objet spécifiée.

Exemple de requête principale générée :

```
SELECT p.HOSO_0000060_uid as 'public provincial health insurance record',
       b.HOSO_0000062_uid as 'primary human health insurance beneficiary'
FROM "HOSO_0000060" p
JOIN "HOSO_0000060_IAO_0000136_HOSO_0000062" j ON j.HOSO_0000060_uid =
p.HOSO_0000060_uid
JOIN "HOSO_0000062" b ON b.HOSO_0000062_uid = j.HOSO_0000062_uid;
```

3.3. Modèles combinatoires (MC_*)

Les modèles combinatoires permettent de combiner plusieurs modèles de requête pour former des requêtes complexes.

3.3.1. MC_COMBINATORIAL : Combinaison de plusieurs modèles

Paramètre	Signification	Type	Obligatoire	Exemple
subModels	Liste des sous-modèles à combiner, séparés par des virgules	String	Oui	"MS_SC,MI_SP"
[subModel].[paramètre]	Paramètre pour un sous-modèle spécifique	String	Oui (pour chaque sous-modèle)	"MS_SC.class=Patient"

Classe de construction : `CombinatorialModelStrategy`

Méthode de construction : `buildCatalogQuery(Map<String, String> options)` et `buildOntoRelQuery(Map<String, String> options, List<String> tableIds)`

Description : Ce modèle permet de combiner les résultats de plusieurs modèles de requêtes. Il construit des sous-requêtes pour chaque sous-modèle, puis les combine en une seule requête en utilisant des expressions de table communes (CTE) avec la clause WITH en SQL. Cette approche permet d'organiser efficacement des requêtes complexes en les décomposant en parties plus petites et plus gérables.

Exemple de requête principale générée :

```
WITH
  records AS (
    SELECT HOSO_0000060_uid as record_id, 'public provincial health insurance record' as
record_label
    FROM HOSO_0000060
  ),
  beneficiaries AS (
    SELECT p.HOSO_0000060_uid as record_id,
           b.HOSO_0000062_uid as beneficiary_id,
           'primary human health insurance beneficiary' as beneficiary_label
    FROM "HOSO_0000060" p
    JOIN "HOSO_0000060_IAO_0000136_HOSO_0000062" j ON j.HOSO_0000060_uid =
```

```
p.HOSO_0000060_uid
    JOIN "HOSO_0000062" b ON b.HOSO_0000062_uid = j.HOSO_0000062_uid
  )
SELECT r.record_id, r.record_label, b.beneficiary_id, b.beneficiary_label
FROM records r
LEFT JOIN beneficiaries b ON r.record_id = b.record_id;
```

4. Performances et optimisation

Cette section présente les caractéristiques de performance des différents types de modèles MCR et fournit des recommandations pour optimiser leur utilisation.

4.1. Métriques de complexité par type de modèle

Type de modèle	Complexité temporelle	Utilisation mémoire	Nombre de jointures	Cas d'utilisation optimal
Modèles simples (MS_*)	O(n)	Faible	1-2	Requêtes directes sur une seule classe
Modèles itératifs (MI_*)	$O(n^2)$ pour MI_AP, O(n log n) pour MI_SP	Moyenne à élevée	Variable (selon le chemin)	Exploration de chemins entre classes
Modèles combinatoires (MC_*)	$O(n \times m)$ où m est le nombre de sous-modèles	Élevée	Somme des jointures des sous-modèles	Requêtes complexes nécessitant plusieurs sous-requêtes

4.2. Optimisation des requêtes

4.2.1. Modèles simples (MS_*)

- Utilisez des filtres (WHERE) spécifiques pour réduire le volume de données traitées
- Préférez MS_SC à MS_SCPO lorsque les propriétés d'objet ne sont pas nécessaires
- Pour MS_SCH, limitez la profondeur de hiérarchie pour éviter des jointures excessives

4.2.2. Modèles itératifs (MI_*)

- Préférez MI_SP à MI_AP pour les graphes denses avec de nombreux chemins possibles
- Utilisez MI_PPO lorsque vous connaissez déjà la propriété d'objet à utiliser
- Limitez la longueur des chemins pour éviter des performances dégradées

4.2.3. Modèles combinatoires (MC_*)

- Limitez le nombre de sous-modèles à 3-5 pour maintenir des performances acceptables
- Organisez les sous-modèles du plus spécifique au plus général
- Utilisez des CTE (WITH) pour optimiser l'exécution des sous-requêtes
- Assurez-vous que les clés de jointure entre sous-modèles sont indexées

4.3. Benchmarks

Des tests de performance ont été réalisés sur un ensemble de données de test (MIMIC - MPHPO Projet Riddle) avec environ 240 000 enregistrements :

Type de modèle	Temps d'exécution moyen	Utilisation mémoire moyenne
MS_SC	< 100ms	< 10MB
MS_SCPO	100-300ms	10-20MB
MS_SCPD	100-300ms	10-20MB
MS_SCH (profondeur 2)	300-500ms	20-30MB
MS_SCH (profondeur 5)	500-1000ms	30-50MB
MI_AP	500-2000ms	50-100MB
MI_SP	300-800ms	30-50MB
MI_PPO	200-500ms	20-40MB
MC_COMBINATORIAL (2 sous-modèles)	500-1500ms	50-100MB
MC_COMBINATORIAL (5 sous-modèles)	1500-3000ms	100-200MB

5. Extension MCR

Cette section explique comment étendre le système MCR avec de nouveaux modèles de requêtes.

5.1. Architecture d'extension

Le système MCR est conçu pour être extensible. Pour ajouter un nouveau modèle, vous devez :

1. Ajouter une nouvelle entrée dans l'énumération `QueryModel`
2. Créer une nouvelle stratégie ou étendre une stratégie existante
3. Enregistrer la stratégie dans `QueryModelStrategyFactory`
4. Implémenter les méthodes de construction de requêtes

5.2. Exemple : Création d'un nouveau modèle simple

Voici un exemple de création d'un nouveau modèle simple MS_SCPF (Sélection de classe par propriété filtrée) :

1. Ajout dans l'énumération `QueryModel` :

```
public enum QueryModel {
    // Modèles existants...

    MS_SCPF("Sélection de classe par propriété filtrée",
            Arrays.asList("class", "objectProperty", "filterCondition"),
            true);

    // Reste de l'énumération...
}
```

1. Implémentation dans `SimpleModelStrategy` :

```
public class SimpleModelStrategy implements QueryModelStrategy {
    // Méthodes existantes...

    private Query buildMsScpfQuery(Map<String, String> options) {
        String classIri = options.get("class");
        String objectPropertyIri = options.get("objectProperty");
        String filterCondition = options.get("filterCondition");

        // Construction de la requête catalogue
        DSLQuery query = dsl.select(field("table_id"), field("label"))
```

```
            .from(table("Onto_class_axiom"))
            .where(field("ontorel_uuid").eq(ontorelUuid))
            .and(field("domain_iri").eq(classIri))
            .and(field("property_iri").eq(objectPropertyIri))
            .and(field("label").eq("en"));

        return query;
    }

    // Implémentation de buildOntoRelQuery pour ce modèle...
}
```

1. Enregistrement dans `QueryModelStrategyFactory` :

```
public class QueryModelStrategyFactory {
    private Map<String, Class> strategyClasses;

    public QueryModelStrategyFactory() {
        strategyClasses = new HashMap<>();
        // Enregistrements existants...
        strategyClasses.put("MS_SCPF", SimpleModelStrategy.class);
    }

    // Reste de la classe...
}
```

5.3. Bonnes pratiques pour l'extension

- Suivez les conventions de nommage existantes (MS_* pour les modèles simples, etc.)
- Documentez clairement les paramètres requis et leur signification
- Assurez-vous que les nouvelles requêtes sont optimisées
- Ajoutez des tests unitaires pour les nouveaux modèles
- Mettez à jour la documentation après l'ajout de nouveaux modèles

6. References

- Documentation fonctionnelle des MCR (SDC)
- Documentation du projet OntoRelQuery
- Spécifications JOOQ : https://www.jooq.org/doc/
- Patterns de conception de requêtes : "SQL Design Patterns" par Vadim Tropashko