Spring Data a pour mission de fournir un modèle de programmation familier et cohérent pour l'accès aux données tout en conservant les traits spéciaux du magasin de données sous-jacent.   
  
Il facilite l'utilisation des technologies d'accès aux données, des bases de données relationnelles et non relationnelles, des cadres de réduction de la carte et des services de données basés sur le cloud. Il s'agit d'un projet de parapluie qui contient de nombreux sous-projets spécifiques à une base de données.

 À l'aide de @Query

L'utilisation de requêtes nommées pour déclarer des requêtes pour des entités est une approche valide et fonctionne bien pour un petit nombre de requêtes. Comme les requêtes elles-mêmes sont liées à la méthode Java qui les exécute, vous pouvez effectivement les lier directement à l'aide de l' @Query annotation Spring Data JPA plutôt que de les annoter dans la classe de domaine. Cela libérera la classe de domaine à partir d'informations spécifiques à la persistance et co-localise la requête dans l'interface du dépôt.

## Utilisation de l'API Criteria et de l'API Metamodel pour créer des requêtes Basic Typesafe

La sémantique de base d'une requête Criteria consiste en une clause SELECT , une clause FROM et une clause WHERE optionnelle , similaire à une requête JPQL. Les requêtes de critères définissent ces clauses en utilisant des objets de langage de programmation Java, de sorte que la requête peut être créée de manière typesafe.

Les requêtes de critères sont une manière sécurisée pour exprimer une requête. Ils sont sécurisés en termes d'utilisation d'interfaces et de classes pour représenter diverses parties structurelles d'une requête telles que la requête elle-même, ou la clause de sélection, ou un ordre, etc. Ils peuvent également être sécurisés en termes de référencement Les attributs que nous verrons un peu. Les utilisateurs de l'ancienne org.hibernate.CriteriaAPI de requête Hibernate reconnaîtront l'approche générale, mais nous croyons que l'API JPA est supérieure car elle représente un regard propre sur les leçons tirées de cette API.

Les requêtes de critères sont essentiellement un graphique d'objet, où chaque partie du graphique représente une augmentation de la partie atomique de la requête (en navigant dans ce graphique). La première étape dans l'exécution d'une requête de critères consiste à créer ce graphique. L'javax.persistence.criteria.CriteriaBuilder interface est la première chose avec laquelle vous devez vous familiariser pour commencer à utiliser des requêtes de critères. Son rôle est celui d'une usine pour toutes les pièces individuelles des critères. Vous obtenez une javax.persistence.criteria.CriteriaBuilder instance en appelant la getCriteriaBuilderméthode de la javax.persistence.EntityManagerFactory

L' annotation @Query présente les avantages suivants:

* Il prend en charge JPQL et SQL.
* La requête invoquée se trouve au-dessus de la méthode de requête. En d'autres termes, il est facile de savoir ce que fait la méthode de requête.
* Il n'y a pas de convention de dénomination pour les noms de méthodes de requête.

L' annotation @Query présente les inconvénients suivants:

* Il n'y a aucun support pour les requêtes dynamiques.
* Si nous utilisons des requêtes SQL, nous ne pouvons pas modifier la base de données utilisée sans tester nos requêtes SQL fonctionnent comme prévu.

Lorsque nous pensons aux avantages et aux inconvénients de l' annotation @Query , il devient clair que les méthodes de requête qui utilisent cette stratégie ne sont pas aussi faciles à écrire ou à lire que les méthodes de requête qui utilisent la [génération de requêtes à partir de la stratégie de nom de méthode](https://www.petrikainulainen.net/programming/spring-framework/spring-data-jpa-tutorial-creating-database-queries-from-method-names/) .

Ainsi, si nous ne voulons pas utiliser la génération de requêtes à partir de la stratégie de nom de méthode, car les noms de nos méthodes de requête seraient trop longs, nous devrions utiliser l' annotation @Query .

Passons et résume ce que nous avons appris de cette publication.

## Résumé

Cette publication de blog nous a appris trois choses:

* Les requêtes de base de données spécifiées en utilisant l' annotation @Query précèdent toutes les autres stratégies de génération de requêtes.
* L' annotation @Query prend en charge les requêtes JPQL et SQL.
* Si nous ne pouvons pas créer nos requêtes de base de données à partir des noms de méthode de nos méthodes de requête, car les noms des méthodes seraient trop longs, nous devrions les créer en utilisant l' annotation @Query .

La prochaine partie de mon didacticiel Spring Data JPA décrit comment nous pouvons [créer des requêtes de base de données en utilisant des requêtes nommées](https://www.petrikainulainen.net/programming/spring-framework/spring-data-jpa-tutorial-creating-database-queries-with-named-queries/) .

Une API REST mature peut être faire beaucoup de travail, et la publication de ressources de manière souple est habituellement un équilibre.

D'une part, vous souhaitez **permettre au client de rechercher des informations de plusieurs façons souples**. D'autre part, vous ne voulez pas implémenter de nombreuses opérations.

**Un langage de recherche / Query pour l'API a** le plus de sens - il permet une opération unique et propre tout en ouvrant l'API pour une recherche puissante.

Il y a plusieurs méthode permet de manipulé la recherche dans une API REST :

#### [>> REST Query Language with Spring et JPA Criteria](http://www.baeldung.com/rest-search-language-spring-jpa-criteria)

Explore la mise en œuvre du langage de requête en utilisant les critères JPA pour la couche de persistance.

#### [>> REST Query Language with Spring Data Spécifications JPA](http://www.baeldung.com/rest-api-search-language-spring-data-specifications)

Explore Spring Data JPA Specifications pour la couche de persistance du Query Language.

#### [>> REST Query Language with Spring Data JPA et QueryDSL](http://www.baeldung.com/rest-api-search-language-spring-data-querydsl)

Ensuite - la couche de persistance est implémentée à l'aide de QueryDSL.

#### [>> REST Query Language - Advanced Search Operations](http://www.baeldung.com/rest-api-query-search-language-more-operations)

Des opérations supplémentaires et plus avancées ont été ajoutées au langage de requête de l'API.

#### [>> REST Query Language with RSQL](http://www.baeldung.com/rest-api-search-language-rsql-fiql)

Enfin, au lieu d’un custom Query Language ,nous allons utiliser Feed Item Query Language ([FIQL](http://tools.ietf.org/html/draft-nottingham-atompub-fiql-00)) and its super-set – RSQL.

#### [>> REST Query Language with Querydsl Web Support](http://www.baeldung.com/rest-api-search-querydsl-web-in-spring-data-jpa)

Querydsl Web Support de Spring est un moyen simple pour obtenir directement un prédicat de la requête HTTP et de l'utiliser pour récupérer des données.

**Pourquoi un langage de requête?**Parce que - pour toute API suffisamment complexe - la recherche / le filtrage de vos ressources par des champs très simples n'est tout simplement pas suffisant. Un langage de requête est plus flexible et vous permet de filtrer les ressources dont vous avez besoin.

<http://www.baeldung.com/intro-to-querydsl>