QueryProvider

Allgemein

Der QueryProvider stellt Möglichkeiten zur Verfügung, Daten aus dem Nuclos-System zu laden und in Form eines BusinessObjects oder Listen von BusinesObjects in Regeln bereitzustellen. Die folgende Liste zeigt den Funktionsumfang der Provider-Klasse und bietet Beispiele für deren Nutzung an.

Ohne die Angabe des Feldes "NuclosLogicalDeleted" werden logisch gelöschte Einträge nicht berücksichtigt. Ebenso bei NuclosLogicalDeleted.eq(Boolean.FALSE). Nur wenn explizit NuclosLogicalDeleted.eq(Boolean.TRUE) in der Query verwendet wird, können gelöschte Einträge abgefragt werden.

Alle Einträge, die nicht logisch gelöscht wurden:

```
Query<Auftrag> qAuftrag = QueryProvider.create(Auftrag.class);
qAuftrag.where(Auftrag.NuclosLogicalDeleted.eq(Boolean.FALSE))

// oder einfach nur

Query<Auftrag> qAuftrag = QueryProvider.create(Auftrag.class);
```

Alle Einträge, die logisch gelöscht wurden:

```
Query<Auftrag> qAuftrag = QueryProvider.create(Auftrag.class);
qAuftrag.where(Auftrag.NuclosLogicalDeleted.eq(Boolean.TRUE))
```

Alle Einträge anzeigen:

```
 Query<Auftrag> qAuftrag = QueryProvider.create(Auftrag.class);
 qAuftrag.where(Auftrag.NuclosLogicalDeleted.eq(Boolean.TRUE).or(Auftrag.NuclosLogicalDeleted.eq
(Boolean.FALSE)))
```

Funktionsumfang

Methode	Beschreibung

create	Mit Hilfe dieser Methode kann ein typisiertes Query-Object angelegt werden, mit dessen Hilfe Datenbankabfragen ausgeführt werden können. Dabei wird auf eine abstrakte Query-Language zurückgegriffen, Abfragen mit SQL-Syntax sind nicht möglich. Die übergebene Klasse muss vom Typ BusinessObject sein. Das Query-Object als Rückgabewert ist ebenfalls typisiert und muss dem übergebenen BusinessObject entsprechen. Zum Ausführen der Query muss die unten beschriebene execute()-Methode verwendet werden.
	
	`public static <T extends BusinessObject> Query<T> create(Class<T> type) {`
	`  return getService().createQuery(type);`
	`}`
	
	Ein Beispiel finden Sie hier.

execute	Diese Methode führt eine typisierte Query auf der Datenbank aus. Da die Suchabfrage generell mehrere Ergebnisse zurückliefern kann, ist der Rückgabewert vom Typ List. Diese ist ebenfalls typisiert und bei keinem gefundenen Treffer leer, aber nicht *null*.
	`public static <T extends BusinessObject> List<T> execute(Query<T> query) {`
	`  return getService().executeQuery(query);`
	`}`
	Ein Beispiel finden Sie hier.

get	Diese Methode ermöglicht die Suche nach einem konkreten Datenbankeintrag. Dazu muss der Typ und die Id angegeben werden. Wird kein Element gefunden, ist der Rückgabewert *null*.
	`public static <T extends BusinessObject> T get(Long id) {`
	`  return getService().get(id);`
	`}`
	Ein Beispiel finden Sie hier.

getByProcess	Diese Methode ermöglicht die Suche nach Datenbankeinträgen, die einer bestimmten Aktion zugeschrieben sind. Jede Aktion in Nuclos gehört zu einem Businessobjekt. Bei der Suche nach Einträgen muss deshalb das Businessobjekt nicht extra angegeben werden. Zwingend erforderlich dagegen ist die Angabe mindestens einer Aktion. `public static <PK, T extends Stateful & BusinessObject<PK>> List<T> getByProcess(Process<T> process, Process<T>... additionalProcesses) throws BusinessException;` Ein Beispiel finden Sie hier.
getByState	Diese Methode ermöglicht die Suche nach Datenbankeinträgen, die einen bestimmten Status besitzen. Da ein Status immer einem Statusmodell angehört, das von mehreren Businessobjekten werden kann, ist die Angabe eine Businessobjekt notwendig. Weiterhin muss mindestens ein Status der Suche übergeben werden. `public static <PK, T extends Stateful & BusinessObject<PK>> List<T> getByState(Class<T> type, State state, State... additionalStates) throws BusinessException` Ein Beispiel finden Sie hier.

Queries und SearchExpressions

Das Query-Interface besitzt folgende Methoden:

Methode	Beschreibung

and	`Query<T> and (Attribute element, Boolean ascending);`
	Mit Hilfe dieser and() - Methode lässt sich eine Mehrfachsortierung bei Abfragen realisieren. Die Reihenfolge der Angaben innerhalb der Query ergibt die Reihenfolge der Sortierung bei der Abfrage.
	`Query<T> and (SearchExpression elm);`
	Mit Hilfe dieser and() - Method lassen sich SearchExpressions miteinander verknüpfen. Der Rückgabewert ist die Query selbst, was eine Aneinanderreihung ermöglicht.

exist	Mit der Exist() - Methode lassen sich Unterabfragen einbinden.
	
	Allgemein kann eine Subquery erstellt und eingebunden werden, deren ID als Fremdschlüssel in der äußeren Query vorhanden ist und so mit dieser verknüpft werden kann.
	
	`<P extends BusinessObject> Query<T> exist(Query<P> subQuery, Attribute element);`
	
	Weiterhin kann mit der Angabe des Vergleichsfeldes aus der Subquery angegeben werden, mit welchem Wert der Vergleich in der MainQuery stattfinden soll.
	
	`<P extends BusinessObject> Query<T> exist(Query<P> subQuery, ForeignKeyAttribute elementMainQuery,`
	`ForeignKeyAttribute elementSubQuery);`
	
	Die Subquery selbst muss nicht ausgeführt werden, sondern wird als Instanz der äußeren Query übergeben und dort ausgewertet. Sehen Sie dazu ein Beispiel: hier

orderBy	`Query<T> orderBy(Attribute element, Boolean ascending);` Mit Hilfe der Order-Methode kann eine Sortierung vorgenommen werden. Als Parameter muss das Feld (nachdem sortiert werden soll) des Businessobjekts (auf das sich die Query bezieht) und eine Sortierreihenfolge angegeben werden.
where	`Query<T> where (SearchExpression elm);` Mit Hilfe der where() - Methode kann eine Suchbedingung für die Query angegeben werden. Mit der Methode and() können diese noch erweitert werden.

Jede dieser Methoden gibt das Query-Objekt selbst zurück, was ein "Stapeln" von Bedingungen ermöglicht.

Die Methoden *where()* und *and()* erhalten als Parameter sogenannte SearchExpression, wie etwa: *Rechnung.Auftragsnr.notNull()*.

Eine SearchExpression bietet die Möglichkeit, Felder miteinander oder Felder mit konkreten Werten zu vergleichen. Damit der Programmierer nicht immer mit dem komplexen Ausdruck *new SearchExpression(...)* arbeiten muss (dient auch nicht der Übersichtlichkeit), werden alle Felder innerhalb von BOs (z.b. Auftragsnr in Rechnung) mit Vergleichsoperatoren versehen (z.b. isNull(), notNull(), gte()), die zu diesem Datentyp passen und als Rückgabewert eine komplette SearchExpression zur Verfügung stellen: *qryRechnung.where(Rechnung.Auftragsnr.notNull())*

Je nach Feldtyp werden unterschiedliche Vergleichsoperatoren angeboten. Das Feld "Auftragsnr" ist als Key eine Zahl und kann numerische Vergleiche vornehmen. Das Feld "Bemerkung" bietet als reiner String-Wert dagegen keine Vergleiche wie *gte()* oder *lte()* an.

Gemeinsam haben alle Felder folgende Vergleiche:

- eq
- neq
- isNull
- notNull

Wichtig hier: Da es sich bei Querys wie auch bei SearchExpressions um Java-Klassen handelt, werden sie immer die equals()-Methoden besitzen. Diese Methode wird in Java für Vergleiche auf Objektebene verwendet, aber bei Querys innerhalb von Nuclos ignoriert.

Etwas komplizierter wird es nun beim "Stapeln" von Suchbedingungen. Da gibt es zwei Möglichkeiten.

1. Verknüpfung mittels Query-Object.

    Wie oben beschrieben, kann das Query-Objekt nebst *where()* und *orderBy()* mit Hilfe der *and()*-Methode mehrere SearchExpressions aufnehmen und miteinander verknüpfen. Die Methode *or()* gibt es an dieser Stelle nicht.

2. Verknüpfung mittels SearchExpression.

    Die SearchExpression bietet außer den Vergleichsmöglichkeiten (Feld-Feld, Feld-Wert) auch die Möglichkeit eine andere SearchExpression "in sich aufzunehmen". Die Methoden innerhalb der SearchExpression lauten dazu:

```
public SearchExpression and(SearchExpression pParentSearchExpression) {}

public SearchExpression or(SearchExpression pParentSearchExpression) {}
```

Damit könnte folgendes Gebilde programmiert werden:

```
Query<Rechnung> qryRechnung = QueryProvider.create(Rechnung.class);

Calendar today = Calendar.getInstance();
Calendar yesterday = Calendar.getInstance();
yesterday.add(Calendar.DAY_OF_WEEK, -1);

qryRechnung.where(Rechnung.Rechnungsdatum.eq(today.getTime()).or(
                                  Rechnung.Rechnungsdatum.eq(yesterday.getTime())))
                  .and(Rechnung.Auftragsnr.notNull())
           .and(Rechnung.Rechnungsnr.notNull())
           .and(Rechnung.Waehrung.eq("EUR").or(
                Rechnung.Waehrung.eq("USD"))
          .orderBy(Rechnung.Rechnungsnr, true);

List<Rechnung> results = QueryProvider.execute(qryRechnung);
```

Gerade diese zweite Variante ermöglicht mit der *or()*-Methode das Verschachteln von Abfragen. *or()* wie auch *and()* geben als Methoden der SearchExpression immer selbige zurück, weshalb sie miteinander verknüpft und als "Block" der Query übergeben werden können.