FirstSpirit™
Unlock Your Content

FirstSpirit™ DynamicDatabaseAccess

FirstSpirit Version 5.0

Version	**2.8**
Status	**RELESED**
Datum	**2012-12-03**
Abteilung	FS-Core
Copyright	2012 e-Spirit AG

Dateiname
DDBA50DE_FirstSpirit_DynamicDatabaseAccess

e-Spirit AG

Barcelonaweg 14
44269 Dortmund | Germany

T +49 231 . 477 77-0
F +49 231 . 477 77-499

info@e-spirit.com
www.e-spirit.com

e-Spirit

Inhaltsverzeichnis

1 Einführung

Die Dokumentation zu FirstSpirit DynamicDatabaseAccess beschreibt das lizenzabhängige FirstSpirit Modul zur Anbindung unterschiedlicher Datenbank-Technologien. Mithilfe der FirstSpirit DynamicDatabaseAccesss-Plattform können Inhalte aus einer Datenbank über eine Webanwendung dargestellt und editiert werden.

Dabei sind unterschiedliche Anwendungsmöglichkeiten vorstellbar:

- FirstSpirit DynamicDatabaseAccess wird nur zur Anzeige der Datenbankinhalte verwendet. In diesem Fall, ist beispielsweise eine Generierung der Datensätze aus der FirstSpirit Datenquellen-Verwaltung nicht mehr notwendig (Die JSP-Seite wird in diesem Fall direkt von der Servlet-Engine ausgeliefert.)
- FirstSpirit DynamicDatabaseAccess wird zur Anzeige und zur Pflege der Datenbankinhalte verwendet. Dabei werden die Strukturen der Datenbanktabellen in FirstSpirit definiert, das Anlegen und Ändern der Datensätze erfolgt aber über die Webanwendung FirstSpirit DynamicDatabaseAccess.
- FirstSpirit™ DynamicDatabaseAccess kann natürlich auch direkt die Datenbankinhalte der FirstSpirit Datenquellen-Verwaltung ändern. Die Änderungen sind in den (dynamischen) JSP-Seiten unmittelbar verfügbar. Innerhalb einer statischen Seite werden die Änderungen erst nach einer Generierung dargestellt.

FirstSpirit DynamicDatabaseAccess verwendet Datenbank-Schemata aus der FirstSpirit Vorlagenverwaltung. Diese Schemata können innerhalb eines Projekts neu definiert werden. Für die Bearbeitung eines Datenbank-Schemas steht ein grafischer Editor zur Verfügung, mit dessen Hilfe das gewünschte Datenbankschema erstellt werden kann. Jedes Schema kann dabei auf bestehende Datenbankstrukturen zurückgreifen, oder neue Tabellenstrukturen in einer bestehenden Datenbank anlegen (weiterführende Informationen zur Verwendung von Datenbank-Schemata siehe „FirstSpirit Handbuch für Entwickler (Grundlagen)“ oder „FirstSpirit Online Dokumentation“).

Der Remote-Zugriff auf Datenbank-Schemata aus weiteren FirstSpirit-Projekten ist ebenfalls möglich (Remote-Projekte).

Sofern eine gültige Lizenz für das Modul besteht, kann über die FirstSpirit Server- und Projektkonfiguration eine Webkomponente installiert werden. Über diese

Komponente wird die Webanwendung gebaut, die über JSP-Tags und Servlets auf die entsprechende Datenbank zugreift (siehe Kapitel 2 ff.).

Achtung: Das Modul FirstSpirit DynamicDatabaseAccess erfordert mindestens den Einsatz einer JDK Version >= 1.5. Dabei muss sichergestellt werden, dass die eingesetzte Servlet-Engine diese JDK-Version unterstützt.

1.1 Thema dieser Dokumentation

Kapitel 2: Beschreibt die Installation und Konfiguration der Webkomponente über die FirstSpirit Server- und Projektkonfiguration (ab Seite 5).

Kapitel 3: Einführung in die Tag-Library zur Steuerung der Webanwendung, mit ausführlichen Beispielen zum Erstellen, Ändern und Löschen von Datensätzen und zur Formulierung von Datenbankabfragen über FirstSpirit DynamicDatabaseAccess (ab Seite 11).

2 Konfiguration

2.1 Installieren des Moduls auf dem Server

Das Modul FirstSpirit DynamicDatabaseAccess muss zunächst innerhalb der Anwendung zur Server- und Projektkonfiguration installiert werden. Im Bereich Servereigenschaften wird dazu der Menüeintrag „Module“ selektiert. Mit einem Klick auf den Button „Installieren“ öffnet sich ein Dateiauswahldialog. Hier kann die zu installierende fsm-Datei (fs-integration.fsm) ausgewählt werden. Die erfolgreich installierte Datei wird anschließend im Dialog „Server Eigenschaften“ angezeigt:

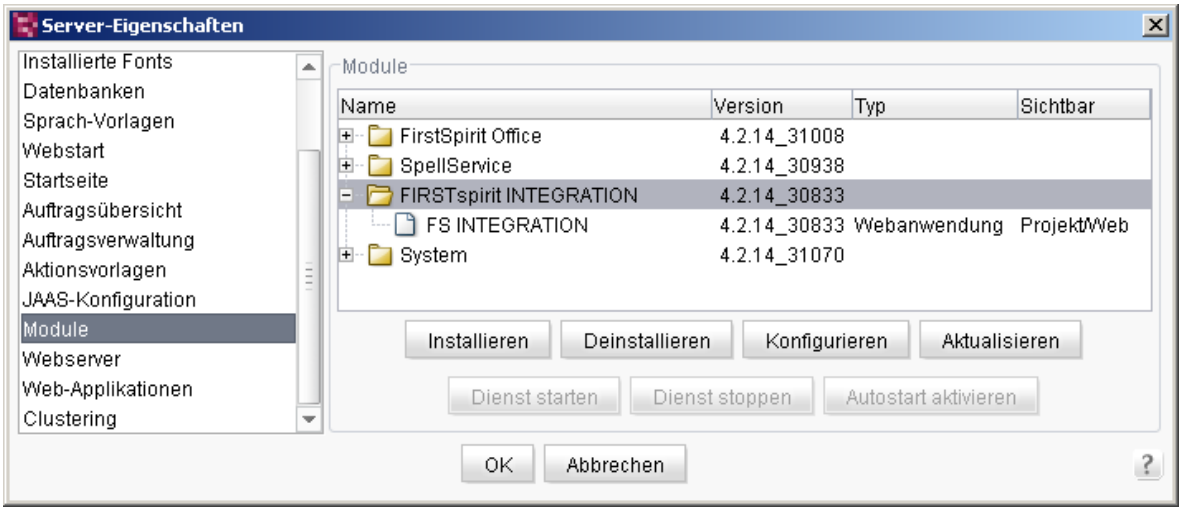

Abbildung 2-1: Installation des Moduls auf dem FirstSpirit™-Server

Bestandteil des Moduls FirstSpirit DynamicDatabaseAccess ist die Webanwendung „FS INTEGRATION“. Die Webanwendung stellt JSP-Tags und Servlets zur Verfügung, die innerhalb des Projekts verwendet und aufgerufen werden können (siehe Kapitel 3.2 Seite 14).

Die Komponente ist „Sichtbar“ für die Bereiche „Projekt/Web“. Damit handelt es sich um eine „web-lokale“ Komponente. Diese kann nach der Installation den unterschiedlichen Web-Bereichen (preview, staging, live) innerhalb der gewünschten Projekte zugefügt werden (siehe Kapitel 2.2 Seite 6).

Weitere Informationen zu diesem Dialog siehe „FirstSpirit Handbuch für Administratoren“.

2.2 Installieren der Webanwendung im Projekt

Die Webanwendung muss nun im gewünschten Projekt installiert werden. Dazu wird innerhalb der Projekteigenschaften der Menüeintrag „Web-Komponenten“ aufgerufen. In diesem Bereich können die Web-Komponenten für ein Projekt aktiviert werden.

Abbildung 2-2: Installation der Webanwendung innerhalb der Web-Bereiche

Es existieren für jedes Projekt drei unterschiedliche Web-Bereiche. Über die jeweilige Registerkarte können die Web-Komponenten für jeden Bereich einzeln aktiviert und konfiguriert werden:

Abbildung 2-3: Web-Bereiche innerhalb eines Projekts

- Vorschau (Preview): Ort für die Projektinhalte für die eine Vorschau angefordert wurde.
- QA (Staging): Ort für die generierten Projektinhalte
- Produktion (Live): Ort für die veröffentlichten Projektinhalte

Hinzufügen: Mit einem Klick auf den Button öffnet sich der Dialog „Hinzufügen“. In der Liste werden alle Web-Komponenten angezeigt, die auf dem Server installiert sind (siehe Kapitel 2.1 Seite 5).

Nach dem Hinzufügen zu einem Webbereich, besteht die Möglichkeit, die Komponenten zu konfigurieren, entweder mit einer von der Komponente erzeugten oder einer generischen GUI (siehe 2.3 Seite 7). Nach der Konfiguration müssen die Komponenten noch aktiviert werden. Dabei kann eine Komponente innerhalb eines

Projekts nur für bestimmte Bereiche aktiviert bzw. deaktiviert werden

Weitere Informationen zu diesem Dialog siehe „FirstSpirit Handbuch für Administratoren“.

2.3 Konfigurieren der Webanwendung

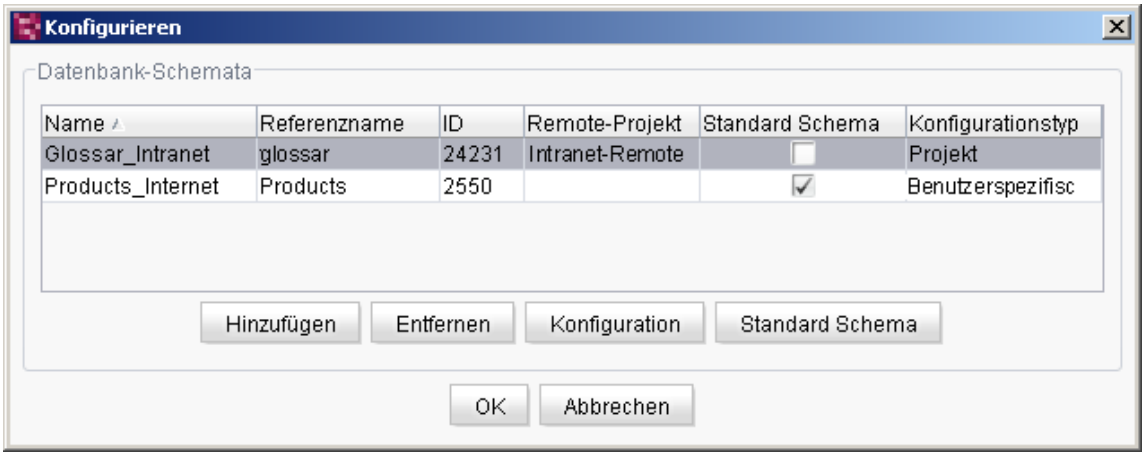

Abbildung 2-4: Konfigurieren der Webanwendung Integration

Mit einem Klick auf den Button „Konfigurieren“ (vgl. Abbildung 2-2) kann die Komponente innerhalb des Web-Bereichs konfiguriert werden. Im Konfigurationsdialog werden die Datenbank-Schemata des Projekts angezeigt, die für den jeweiligen Web-Bereich verwendet werden sollen:

Name: Enthält den symbolischen Namen für die Konfiguration eines Datenbank-Schemas. Dieser Name wird beim Hinzufügen einer Konfiguration vom Administrator vergeben.

Referenzname: Enthält den Referenznamen des Schemas (aus dem Projekt).

ID: Eindeutige ID des Schemas im Projekt.

Remote-Projekt: Sofern das Schema aus einem Remote-Projekt stammt, enthält diese Spalte den symbolischen Remote-Projektnamen.

Standard Schema: Die Kennzeichnung “Standard Schema” kennzeichnet initial das Schema, das zuerst zur Konfiguration hinzugefügt wurde. Für jede Konfiguration (innerhalb eines Web-Bereichs) gibt es jeweils ein Standard-Schema. Das Standard-Schema wird benötigt, wenn z.B. mithilfe der JSP-Tags manuell auf eine bestimmte Datenbank-Tabelle zugegriffen wird. Ist dort nicht explizit ein Schema definiert, wird das Standard-Schema verwendet.

Konfigurationstyp: Die Spalte kennzeichnet, ob eine Konfiguration benutzerspezifisch oder projektspezifisch definiert wurde. Standardmäßig wird eine Konfiguration projektspezifisch (Typ: „Projekt“) angelegt. Die benutzerspezifische Konfiguration wird in Kapitel 2.3.1 beschrieben.

Hinzufügen Mit einem Klick auf den Button wird ein neues Datenbank-Schema aus der Vorlagen-Verwaltung des Projekts zur Konfiguration hinzugefügt. Dabei öffnet sich ein Dialog mit den verfügbaren Schemata des lokalen Projekts. Sofern das lokale Projekt einen Remote-Zugriff auf weitere FirstSpirit™-Projekte besitzt, werden diese in getrennten Registern angezeigt:

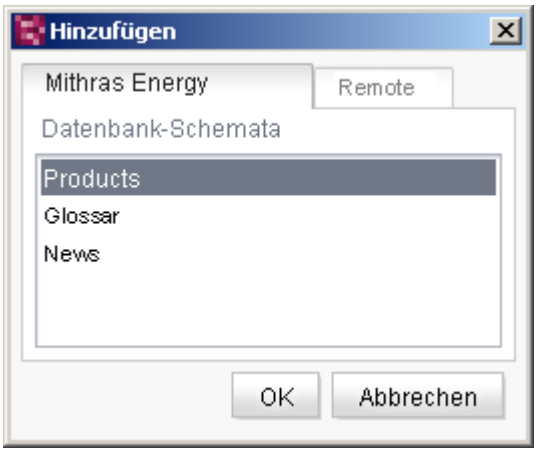

Abbildung 2-5: Auswahl eines FirstSpirit™-Schemas für die Konfiguration

Der Remote-Zugriff auf Schemata weiterer FirstSpirit™-Projekte ist nur möglich, wenn innerhalb der Remote-Projekt-Konfiguration der Remote-Typ „Remote-Schemata“ aktiviert wurde.

Entfernen Mit einem Klick auf den Button wird ein bestehendes Datenbank-Schema aus der Konfiguration entfernt.

Konfiguration Mit einem Klick auf den Button kann ein Datenbank-Schema für FS Integration konfiguriert werden (siehe 2.3.1 Seite 9).

Standard Schema Mit einem Klick auf den Button kann ein bestehendes Datenbank-Schema aus der Konfiguration zum Standard-Schema werden. Das bisherige Standard-Schema bleibt als Schema innerhalb der Konfiguration erhalten, ist jedoch nicht mehr als Standard-Schema gekennzeichnet.

2.3.1 Datenbankschema konfigurieren

Mit einem Klick auf den Button „Konfigurieren“ innerhalb des Konfigurations-Dialogs (vgl. Abbildung 2-4) öffnet sich ein weiterer Konfigurationsdialog für das selektierte Datenbankschema.

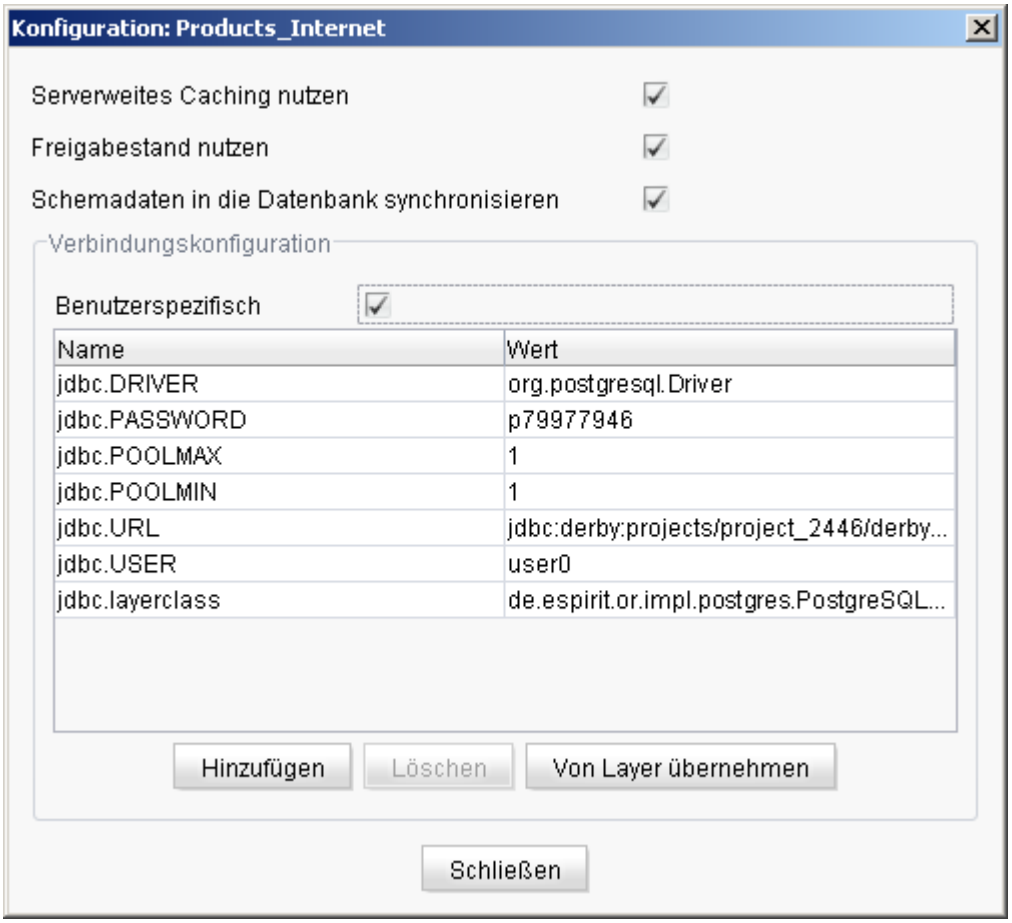

Abbildung 2-6: Konfiguration des Datenbank-Schemas

Zur Konfiguration eines Datenbank-Schemas stehen unterschiedliche Optionen und Parameter zur Konfiguration der Verbindung zur Datenbank zur Verfügung.

Serverweites Caching nutzen: Wird diese Option aktiviert, wird ein serverweites Caching der Objekte eingeschaltet (“application”-Scope). Wird die Option deaktiviert, wird ein benutzerlokales Caching (HTTP-Session-Scope) verwendet (Standard-Einstellung).

Freigabestand nutzen: Wird diese Option aktiviert, wird für alle Anfragen der Freigabestand (der Datenbankinhalte) verwendet. Wird diese Option deaktiviert, wird für alle Anfragen der aktuelle Stand zurückgeliefert.

Schemadaten in die Datenbank synchronisieren: Wird diese Option aktiviert, werden bei Initialisierung des Moduls die Daten des Schemas (im Modul) mit den Inhalten der Datenbank abgeglichen. Damit entspricht diese Funktionalität dem Speichern eines Schemas in der FirstSpirit Vorlagen-Verwaltung. Es werden aber nicht die Schema-Inhalte aus dem Projekt übernommen, sondern die Inhalte aus dem FirstSpirit Modul (xml-Datei).

Unterhalb dieser Optionen können weitere Parameter für die Verbindung zur Datenbank konfiguriert werden:

Benutzerspezifisch: Wird diese Option aktiviert, kann eine manuelle bzw. benutzerspezifische Konfiguration des Datenbank-Layers definiert werden. Bei aktivierter Option werden die aktuell im Modul (oder innerhalb der Webkonfiguration) gespeicherten Werte übernommen. Die Werte zwischen aktiviert und nicht aktiviert können potentiell nach einem Export oder nach Änderungen des Administrators abweichen.
Wird die Option deaktiviert, wird der Datenbank-Layer verwendet, der innerhalb des Projekts vom Schema verwendet wird. Die Parameter werden dann ebenfalls aus der Layer-Konfiguration des Schemas übernommen.

Zur Beschreibung der Parameter siehe „FirstSpirit Handbuch für Administratoren “ (Kap. 4.8.1. „Konfiguration von Datenquellen“ ff.).

Mit einem Klick auf den Button können weitere Parameter zur Konfiguration hinzugefügt werden.

Mit einem Klick auf den Button können Parameter aus der Konfiguration entfernt werden.

Mit einem Klick auf den Button wird statt der momentan gespeicherten Verbindungskonfiguration die Layer-Konfiguration des Projektes übernommen.

3 Syntax

3.1 Allgemeine Informationen

Das folgende Kapitel erläutert die Funktionen der Tag-Library zur Steuerung der Webanwendung. Dabei werden sowohl allgemeine, formale Definitionen beschrieben (z.B. die Festlegung eines Präfixes für die JSP-Tags) (siehe Kapitel 3.1) als auch die Verwendung der verfügbaren JSP-Tags erläutert (ab Kapitel 3.2). Weitere Kapitel enthalten aktuelle Beispiele zum Erstellen, Ändern und Löschen von Datensätzen (siehe Kapitel 3.3 und Kapitel 3.4) und eine kurze Einführung in Abfragen („Querys“) über FirstSpirit DynamicDatabaseAccess (siehe Kapitel 3.5).

3.1.1 Präfix

Um FirstSpirit DynamicDatabaseAccess verwenden zu können, müssen in den JSP-Seiten die entsprechenden Taglibs angegeben werden. Diese Dokumentation verwendet das Präfix “fsi” für FirstSpirit™-Integration-Tags und das Präfix “c” für die Tags der Core-Funktionen der JSTL[1].

Beispiel für die Einbindung in JSP-Seiten:

```
<%@ page language="java" contentType="text/html; charset=UTF-8" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ taglib uri="fsweb" prefix="fsi" %>
```

Zu beachten ist hierbei, dass der URI für FirstSpirit immer “fsweb” lautet.

Wichtig: Wird das Präfix “fsi” auf einen anderen Wert geändert, so ist dieses Präfix für die einzelnen Tags zu verwenden, d.h. “<**myPrefix**:ref>” anstelle von “<**fsi**:ref>”.

[1] JavaServer Pages Standard Tag Library – Informationen zu JSTL siehe http://wikipedia.org/wiki/JSTL

3.1.2 Objekt-/Attributreferenzen

Im Web-Modul werden an vielen Stellen Objekt- bzw. Attributreferenzen verwendet. Dabei kann es sich abhängig von der Verwendung um Listen, einzelne Objekte oder Attribute einzelner Objekte handeln. Ein Objekt wird durch einen Punkt vom Attribut bzw. von der Attributkette getrennt.

Beispiel:

```
OBJEKT.ATTRIBUT_1[.ATTRIBUT_2][.ATTRIBUT_3]
```

Es gibt zwei Typen von Objektreferenzen: relative und absolute.

In der Regel wird mit einer absoluten Objektreferenz begonnen, z.B. für eine Iteration. Alle weiteren Operationen werden dann mit relativen Referenzen durchgeführt.

Absolute Referenzen zeichnen sich durch Klammern hinter dem symbolischen Bezeichner aus, z.B. „movie()“. Beim symbolischen Bezeichner handelt es sich um den Namen einer Tabelle aus einem Schema.

- **Alle Datensätze:** Steht zwischen der öffnenden und schließenden Klammer kein Zeichen, so werden alle Datensätze (d.h. eine Liste von Entity-Objekten) der angegebenen Tabelle zurückgeliefert. Dies ist vergleichbar mit der Rückgabe der Header-Funktion „contentSelect“ (weitere Informationen siehe FirstSpirit™-Online-Dokumentation[2]).
- **Ein Datensatz:** Soll ein spezieller Datensatz (ein Entity-Objekt) zurückgeliefert werden, so kann innerhalb der Klammern die eindeutige ID des Datensatzes angegeben werden, z.B. „movie(12)“.
- **Neuer Datensatz:** Das Sonderzeichen „*“ (Stern) wird für die Anlage eines neuen Datensatzes verwendet, z.B. „movie(*)“.

Die verfügbaren Attribute, die auf einer Objektreferenz anwendbar sind, können jeder Tabelle im Schema entnommen werden. Die Verwendung ähnelt an dieser Stelle der Verwendung des Systemobjekte „#row“ in Tabellenvorlagen (weitere Informationen siehe FirstSpirit™-Online-Dokumentation[3]).

[2] Bereich: .../vorlagenentwicklung/vorlagensyntax/funktionen/im_header/contentselect/contentselect.html
[3] Bereich .../vorlagenentwicklung/vorlagensyntax/systemobjekte/row/row_1.html

Beispiel:

```
$CMS_VALUE(#row.title)$
  $--
    gibt statisch die Spalte „title“ des aktuellen Datensatzes aus
  --$

${m.title}
  $--
    gibt dynamisch die Spalte „title“ eines Datensatzes aus
  --$
```

Beispiele für die Notation von absoluten Objektreferenzen:

Notation	Bedeutung
movie()	alle „movie”-Objekte bzw. alle Datensätze der Tabelle „movie“
movie(12)	„movie”-Objekt mit dem fachlichen Schlüssel „12” bzw. der Datensatz mit der ID „12“ aus der Tabelle „movie“
movie(37).title	Attribut „title“ des „movie”-Objekts mit dem fachlichen Schlüssel „37” bzw. die Spalte „title“ des Datensatzes mit der ID „37“ aus der Tabelle „movie“
movie(*).title	Attribut „title“ eines neuen „movie“-Objekts bzw. Spalte „title“ eines neuen Datensatzes der Tabelle „movie“

Beispiele für relative Referenzen: In den Syntaxbeispielen ist „m“ ein „movie“-Objekt, das durch die Verwendung des Tags <fsi:find> ermittelt wurde.

Notation	Bedeutung
m	ein „movie“-Objekt, welches durch „m“ repräsentiert wird bzw. ein Datensatz (Entity) der Tabelle „movie“.
m.actors	„actors“-Objekt (zu-N-Relation) von „m“ bzw. die gespeicherten Werte der Fremdschlüsselbeziehung (Bezeichnung „actors“) der Tabelle „movie“ zu einer anderen Tabelle.
m.title	Attribut „title“ von „m“ bzw. die Spalte „title“ eines Datensatzes der Tabelle „movie“.

3.1.3 Durch Tags definierte Variablen

In den Tags <fsi:getQueryDetails>, <fsi:iterateResults>, <fsi:navigation>, <fsi:navBottom>, <fsi:navFrame> und <fsi:navTop> werden Variablen definiert.

Dabei kann es zu Überschneidung mit manuell definierten Variablen in der JSP-Seite kommen. Um Überschneidungen zu vermeiden, sollten folgende Variablenbezeichner nicht (bzw. mit Bedacht) verwendet werden:

- firstFramePage
- hasMore
- lastFramePage
- navPageNo
- navUrl
- pageNo
- pageSize
- parameter
- query
- totalPages
- totalResults

Welches Tag, welche Variable definiert, kann den einzelnen Tag-Beschreibungen entnommen werden (siehe Kapitel 3.2).

3.2 JSP-Tags

3.2.1 Daten suchen und anzeigen

Mit FirstSpirit DynamicDatabaseAccess können Daten gesucht und angezeigt werden, z.B.:

- Suche nach einem einzelnen Datensatz über (<fsi:find>)
- Suche nach mehreren Datensätzen über (<fsi:search>)
- seitenweise Ausgabe der Datensätze über (<fsi:iterateResults> in Verbindung mit <fsi:search pageSize="....">)
- gesamte Ausgabe der Datensätze über (<fsi:iterateResults> in Verbindung mit (<c:forEach>)

Dazu stehen eine Vielzahl von Tags zur Verfügung:

- <fsi:setSchema> (Festlegen eines Schemas) (siehe Kapitel 3.2.1.1 Seite 15)

- <c:out> / ${...} (Ausgabe von Attributen) (siehe Kapitel 3.2.1.2 Seite 16)
- <fsi:ref> (Verweise definieren) (siehe Kapitel 3.2.1.3 Seite 17)
- <fsi:find> (Suche nach Objekten) (siehe Kapitel 3.2.1.4 Seite 20)
- <c:forEach> (Ausgabe von Listen) (siehe Kapitel 3.2.1.5 Seite 22)
- <fsi:search> (Suche nach Datensätzen) (siehe Kapitel 3.2.1.6 Seite 24)
- <fsi:query> (Abfragen definieren) (siehe Kapitel 3.2.1.7 Seite 26)
- <fsi:iterateResults> (Gesamtausgabe) (siehe Kapitel 3.2.1.8 Seite 27)
- <fsi:getQueryDetails> (Infos zur Abfrage) (siehe Kapitel 3.2.1.9 Seite 28)

Hinweis: Da die Datenbankanfragen zwischengespeichert werden, sollte für eine aktuelle Darstellung der Cache zurückgesetzt werden. Hierfür muss die Klasse „WebContext“ importiert werden:

```
<%@ page
import="de.espirit.firstspirit.opt.integration.web.WebContext" %>
```

Anschließend kann der Cache mit folgendem Aufruf zurückgesetzt werden:

```
<%
  WebContext.getWebContext(pageContext).getSession().rollback();
%>
```

3.2.1.1 Festlegen des Schemas (<fsi:setSchema>)

Mit dem JSP-Tag <fsi:setSchema> kann ein Schema festgelegt werden.

Alle Tags, die sich zwischen dem öffnenden und schließenden Tag befinden, verwenden dieses Schema.

Für das Tag ist das Attribut „schema“ anzugeben, das als Wert den eindeutigen Bezeichner des Schemas aus der Modul-Konfiguration erwartet.

Attribute:

Attribut	Erwarteter Wert	Pflichtparameter
schema	eindeutiger Bezeichner des Schemas aus der Modul-Konfiguration	Nein

Beispiel:

```
<fsi:setSchema schema="movies">
  <fsi:search resultName="mL">
    <fsi:query>
      <QUERY entityType="movie" />
    </fsi:query>

    <c:forEach items="${mL}" var="m">
      <p>
        <c:out value="${m.title}" />
      </p>
    </c:forEach>
  </fsi:search>
</fsi:setSchema>
```

Im Beispiel wird das von den Tags <fsi:search>, <fsi:query> und <c:forEach> zu verwendende Schema auf „movies“ gesetzt.

3.2.1.2 Ausgabe von Attributen (<c:out> / ${...})

Zur Ausgabe von Attributen kann entweder das <c:out>-Tag der JSTL[4] oder die JSP EL-Notation (${...}) verwendet werden.

Attribute des <c:out>-Tags:

Attribut	Erwarteter Wert	Pflichtparameter
value	auszugebendes Attribut in JSP EL-Notation (${...}) oder ein fester Wert	Ja
escapeXml	legt fest, ob die Zeichen „<“, „>“, „&“, „'“ und „"“ in HTML-Entitäten umgewandelt werden sollen. Mögliche Werte sind: „true“ oder „false“	Nein
default	Standardwert, der ausgegeben wird, wenn „value“ null zurückliefert.	Nein

[4] JavaServer Pages Standard Tag Library – Informationen zu JSTL siehe http://wikipedia.org/wiki/JSTL

Beispiel für die Attributausgabe mit dem <c:out>-Tag:

```
<fsi:find item="movie(98)" resultName="m">
      Titel: <c:out value="${m.title}" default="" />
</fsi:find>
```

Beispiel für die Ausgabe eines Festwertes mit dem <c:out>-Tag:

```
<fsi:find item="movie(98)" resultName="m">
      Fix value: <c:out value="VALUE" />
</fsi:find>
```

Beispiel für Attributausgabe mit der JSP EL-Notation:

```
<fsi:find item="movie(98)" resultName="m">
      Title: ${m.title}
</fsi:find>
```

Wichtig: In früheren Versionen wurde zur Attributausgabe das Tag <fsi:get> verwendet, welches nicht mehr unterstützt wird. Es besteht jedoch die Möglichkeit, die <fsi:get>-Syntax in eine <c:out>- bzw. JSP EL-Syntax zu überführen.

Beispiel:

```
Alte Syntax:
<fsi:find item="movie(98)" resultName="m">
      Title: <fsi:get item="m.title" />
</fsi:find>


Neue Syntax:
<fsi:find item="movie(98)" resultName="m">
      Titel: <c:out value="${m.title}" />
</fsi:find>

oder

<fsi:find item="movie(98)" resultName="m">
      Title: ${m.title}
</fsi:find>
```

3.2.1.3 Eindeutige Verweise definieren (<fsi:ref>)

Das Speichern von Werten wird innerhalb der JSP-Seiten mit HTML-Eingabekomponenten oder mit JavaScript-basierenden Editoren (z.B. TinyMCE, HTMLArea, FCK Editor u.ä.) in einem HTML-Formular realisiert. Jede HTML-Eingabekomponente verfügt über einen Parameter „name“ (Bezeichnung) und einen „value“ (Wert). Bei der Speicherung versucht das Store-Servlet, den übertragenen Wert anhand des „name“-Parameters zu speichern (siehe Kapitel 3.3).

Damit das Store-Servlet die Werte den richtigen Spalten bzw. Datensätzen in der Datenbank zuordnen kann, benötigt es einen eindeutigen Verweis auf die Spalte und den Datensatz. Um für die Speicherung einen eindeutigen Verweis zu einem Attribut bzw. Objekt zu erhalten, kann das <fsi:ref>-Tag verwendet werden.

Attribute:

Attribut	Erwarteter Wert	Pflichtparameter
Value	Objekt- oder Attributreferenz, z.B. „m.title“	Ja
resultName	Variablenname zur weiteren Verwendung in der JSP-Seite. Wird der Parameter angegeben, wird der eindeutige Verweis nicht ausgegeben.	Nein

Wichtig: Die Verwendung des <fsi:ref>-Tags ist nur für das Speichern bzw. Ändern von Werten nötig (siehe auch Beschreibung „Store-Servlet“ in Kapitel 3.3).

Das Tag unterstützt zwei Attribute: „value“ und „resultName“. Beim Attribut „value“ handelt es sich um einen Pflichtparameter. Wird der Parameter „value“ ohne „resultName“ verwendet, so wird der eindeutige Verweis direkt ausgegeben:

```
<form action="<%= application.getContextPath() %>/do.store"
      method="post">

  <fsi:search resultName="mL">
    <fsi:query>
      <QUERY entityType="movie" />
    </fsi:query>

    <c:forEach items="${mL}" var="m">
      <p>
        Title:
        <input name="<fsi:ref value='m.title' />"
               value="<c:out value='${m.title}'
                             escapeXml='true' />" />
      </p>
    </c:forEach>
  </fsi:search>

  <input type="hidden"
         name="schema"
         value="movies" />
  <input type="hidden"
         name="url_ok"
         value="$CMS_REF(#global.node)$" />
  <input type="submit"
         value="Submit" />
```

```
</form>
```

Die Angabe des Parameters „resultName“ führt dazu, dass der eindeutige Verweis nicht mehr direkt ausgegeben wird. Der Verweis muss hier manuell mit dem <c:out>-Tag bzw. der JSP EL-Syntax (${...}) ausgegeben werden:

```
<form action="<%= application.getContextPath() %>/do.store"
      method="post">

  <fsi:search resultName="mL">
    <fsi:query>
      <QUERY entityType="movie" />
    </fsi:query>

    <c:forEach items="${mL}" var="m">
      <fsi:ref value="m.title" resultName="t" />
      <p>
        Title:
        <input name="${t}"
               value="<c:out value='${m.title}'
                             escapeXml='true' />" />
      </p>
    </c:forEach>
  </fsi:search>

  <input type="hidden"
         name="schema"
         value="movies" />
  <input type="hidden"
         name="url_ok"
         value="$CMS_REF(#global.node)$" />
  <input type="submit"
         value="Submit" />
</form>
```

Ein weiterer wichtiger Aspekt der Nutzung von <fsi:ref> ist die Auswahl und Speicherung von Daten aus einer Fremdschlüsselbeziehung. In diesem Fall wird das <fsi:ref>-Tag nicht nur für das „name“-Attribut, sondern gleichzeitig für das „value“-Attribut genutzt:

```
<form action="<%= application.getContextPath() %>/do.store"
      method="post">

  <fsi:search resultName="mL">
    <fsi:query>
      <QUERY entityType="movie" />
    </fsi:query>

    <c:forEach items="${mL}" var="m">
      <p>
        <select name="<fsi:ref value='m.rating' />">

          <fsi:search resultName="rL">
            <fsi:query>
```

```
            <QUERY entityType="rating" />
          </fsi:query>

          <c:forEach items="${rL}" var="r">
            <option value="<fsi:ref value='r' />"
                    <c:if test="${m.rating == r}">
                      selected="selected"
                    </c:if>>
              ${r.abbreviation} (${r.name})
            </option>
          </c:forEach>
        </fsi:search>

      </select>
    </p>
  </c:forEach>
</fsi:search>

<input type="hidden"
       name="schema"
       value="movies" />
<input type="hidden"
       name="url_ok"
       value="$CMS_REF(#global.node)$" />
<input type="submit"
       value="Submit" />
</form>
```

3.2.1.4 Objektsuche (<fsi:find>)

Mit dem Tag <fsi:find> kann genau ein Objekt gesucht und in einer Variable gespeichert werden. Auf die Variable kann nur zwischen dem öffnenden und schließenden Tag von <fsi:find> zugegriffen werden. Das Tag verfügt über die beiden Pflichtparameter „item“ und „resultName“, sowie den optionalen Parameter „schema“.

Attribute:

Attribut	Erwarteter Wert	Pflichtparameter
item	Objekt- oder Attributreferenz, z.B. „movie(103)“	Ja
resultName	Variablenname zur weiteren Verwendung innerhalb des Tags.	Ja
schema	eindeutiger Bezeichner des Schemas aus der Modul-Konfiguration	Nein

Der Parameter „item“ dient dazu, eine Objektreferenz (z.B. movie(103)) für die Suche anzugeben. Hierbei muss beachtet werden, dass die Objektreferenz nur ein Objekt zurückliefert und nicht eine Liste von Objekten.

Mit dem Parameter „resultName“ wird der Variablenname für die weitere Verwendung angegeben.

Über den Parameter „schema“ kann ein Schema angegeben werden. Die Objektsuche findet dann nur innerhalb des definierten Schemas statt:

```
<fsi:find item="movie(104)" resultName="m" schema="movies">
  <p>
    Title: ${m.title}
  </p>
</fsi:find>
```

Anstelle des Parameters „schema“ kann auch das Tag <fsi:setSchema> verwendet werden.

Wichtig: Da die Verwendung des Attributes „schema“ (gerade bei Verschachtelungen) fehleranfällig ist, wird die Verwendung des <fsi:setSchema>-Tags empfohlen:

```
<fsi:setSchema schema="movies">
  <fsi:find item="movie(104)" resultName="m">
    <p>
      Title: ${m.title}
    </p>
  </fsi:find>
</fsi:setSchema>
```

Wird weder das Attribut „schema“ noch das Tag <fsi:setSchema> angegeben, erfolgt die Suche automatisch im Standard-Schema (vgl. Kapitel 2.3 Seite 7).

Alternativ zur Verwendung von <fsi:find> kann die Suche auch mit <fsi:search> realisiert werden:

```
<fsi:setSchema schema="movies">
  <fsi:search resultName="mL">
    <fsi:query>
      <QUERY entityType="movie" limit="1">
        <EQ attribute="fs_id"
            datatype="java.lang.Integer"
            value="104" />
      </QUERY>
    </fsi:query>

    <c:forEach items="${mL}" var="m">
      <p>
        Title: ${m.title}
      </p>
```

```
    </c:forEach>
  </fsi:search>
</fsi:setSchema>
```

3.2.1.5 Ausgabe von Listen (<c:forEach>)

Um Listen auszugeben, kann das <c:forEach>-Tag der JSTL[5] verwendet werden.

Mit dem Pflichtparameter „items“ wird dem Tag die Liste der Objekte übergeben (z.B. ${mL})). Für jedes Element der Liste wird der Inhalt des Tags einmal ausgeführt.

Soll auf ein Element der Liste mit einem festen Bezeichner zurückgegriffen werden, so lässt sich dieser mit dem optionalen Parameter „var“ angeben. Auf das Element kann dann innerhalb des Tags mit <c:out> oder JSP EL (${...}) zurückgegriffen werden (vgl. Kapitel 3.2.1.2 Seite 16).

```
<fsi:search resultName="mL">
  <fsi:query>
    <QUERY entityType="movie" />
  </fsi:query>

  <c:forEach items="${mL}" var="m">
    <p>
      ${m.title}
    </p>
  </c:forEach>
</fsi:search>
```

Attribute:

Attribut	Erwarteter Wert	Pflichtparameter
items	Liste von Objekten	Ja
var	Variablenname zur weiteren Verwendung innerhalb des Tags.	Nein

Wichtig: In früheren Versionen wurde zur Ausgabe von Listen das Tag <fsi:iterate> verwendet, welches nicht mehr unterstützt wird. Es gibt jedoch die Möglichkeit, die <fsi:iterate>-Syntax in eine <c:forEach>-Syntax zu überführen.

[5] JavaServer Pages Standard Tag Library – Informationen zu JSTL siehe http://wikipedia.org/wiki/JSTL

Beispiel:

```
Alte Syntax:
<fsi:iterate list="movie()" resultName="m">
  <p>
    <fsi:get item="m.title"/>
  </p>
</fsi:iterate>


Neue Syntax:
<fsi:search resultName="mL">
  <fsi:query>
    <QUERY entityType="movie" />
  </fsi:query>

  <c:forEach items="${mL}" var="m">
    <p>
      ${m.title}
    </p>
  </c:forEach>
</fsi:search>
```

3.2.1.6 Datensätze suchen (<fsi:search>)

Die Suche nach Datensätzen ist über das Tag <fsi:search> möglich. Alternativ dazu können über das Query-Servlet Abfragen ausgeführt bzw. Datensätze gesucht werden.

Grundlage für die Suche ist die mit dem <fsi:query>-Tag angegebene xml-Abfrage. Das <fsi:query>-Tag muss daher als erstes Tag innerhalb von <fsi:search> angegeben werden. Das Ergebnis der Abfrage steht anschließend anderen Tags, innerhalb des <fsi:search>-Tags, zur Verfügung.

Die Ergebnisliste der gefundenen Elemente kann entweder vollständig (<c:forEach> – siehe Kapitel 3.2.1.5) oder seitenbasiert (<fsi:iterateResults> – siehe Kapitel 3.2.1.8) ausgegeben werden. Weitere Informationen zur Abfrage können über das Tag <fsi:getQueryDetails> ermittelt werden (siehe Kapitel 3.2.1.9).

<fsi:query> besitzt keine Pflichtparameter, es können aber die optionalen Parameter „pageSize“, „resultName“ und „schema“ verwendet werden:

- **Parameter „pageSize“:** Mit dem Parameter „pageSize“ werden die Elemente der Ergebnisliste auf verschiedene (virtuelle) Seiten aufgeteilt (vgl. Kapitel 3.2.4 Navigation, ab Seite 36). Der Wert des Parameters definiert, wie viele Elemente pro Seite dargestellt werden.
- **Parameter „resultName“:** Durch den Parameter „resultName“ wird der Variablenname für die weitere Verwendung festgelegt.
- **Parameter „schema“:** Weiterhin ist es möglich, mit dem Parameter „schema“ das Schema anzugeben, in dem gesucht werden soll:

```
<fsi:search resultName="mL" schema="movies">
  <fsi:query>
    <QUERY entityType="movie" />
  </fsi:query>

  <c:forEach items="${mL}" var="m">
    <p>
      ${m.title}
    </p>
  </c:forEach>
</fsi:search>
```

Anstelle des Parameters „schema“ kann auch das Tag <fsi:setSchema> verwendet werden:

```
<fsi:setSchema schema="movies">
  <fsi:search resultName="mL">
    <fsi:query>
      <QUERY entityType="movie" />
    </fsi:query>

    <c:forEach items="${mL}" var="m">
      <p>
        ${m.title}
      </p>
    </c:forEach>
  </fsi:search>
</fsi:setSchema>
```

Wichtig: Da die Verwendung des Attributes „schema“ (gerade bei Verschachtelungen) fehleranfällig ist, wird die Verwendung des <fsi:setSchema>-Tags empfohlen.

Attribute:

Attribut	Erwarteter Wert	Pflichtparameter
pageSize	Anzahl der Elemente, die pro Seite dargestellt werden sollen.	Nein
resultName	Variablenname zur weiteren Verwendung innerhalb des Tags.	Nein
schema	eindeutiger Bezeichner des Schemas aus der Modul-Konfiguration	Nein

Beispiel für die vollständige Ausgabe aller Elemente:

```
<fsi:search resultName="mL">
  <fsi:query>
    <QUERY entityType="movie" />
  </fsi:query>

  <c:forEach items="${mL}" var="m">
    <p>
      ${m.title}
    </p>
  </c:forEach>
</fsi:search>
```

Beispiel für seitenbasierte Ausgabe von Elementen:

```
<fsi:search pageSize="5">
  <fsi:query>
    <QUERY entityType="movie" />
  </fsi:query>

  <fsi:iterateResults resultName="m">
    <p>
      ${m.title}
    </p>
  </fsi:iterateResults>
</fsi:search>
```

3.2.1.7 Abfrage erstellen (<fsi:query>)

Das Tag <fsi:query> enthält eine Abfrage, die als Grundlage für die Suche innerhalb eines <fsi:search>-Tags verwendet wird (vgl. Kapitel 3.2.1.6 Seite 24). Abfragen verwenden die gleiche xml-Syntax, wie die Abfragen eines Schemas in der Vorlagen-Verwaltung und in der Header-Funktion „contentSelect“ (weitere Informationen siehe FirstSpirit-Online-Dokumentation[6]).

Das <fsi:query>-Tag muss als erstes Tag innerhalb von <fsi:search> angegeben werden. Das Ergebnis der Abfrage steht anschließend anderen Tags, innerhalb des <fsi:search>-Tags, zur Verfügung.

Außer dem Tag <QUERY> sind keine anderen Tags innerhalb von <fsi:query> zulässig.

Beispiel:

```
<fsi:search resultName="mL">
  <fsi:query>
    <QUERY entityType="movie">
      <ORDER>
        <ORDERCRITERIA attribute="title" descending="1" />
      </ORDER>
    </QUERY>
  </fsi:query>

  <c:forEach items="${mL}" var="m">
    <p>
      ${m.title}
    </p>
  </c:forEach>
</fsi:search>
```

[6] Bereich: .../vorlagenentwicklung/vorlagensyntax/funktionen/im_header/contentselect/contentselect.html

3.2.1.8 Ausgabe aller Elemente einer Seite (<fsi:iterateResults>)

Durch das Tag <fsi:search> können die Elemente einer Liste auf unterschiedliche (virtuelle) Seiten verteilt werden (vgl. Kapitel 3.2.1.6). Während das Tag <c:forEach> alle Elemente der Liste ausgibt bzw. durchläuft (siehe Kapitel 3.2.1.5), beschränkt sich das Tag <fsi:iterateResults> auf die Elemente der aktuellen Seite.

Das Tag verfügt über den Pflichtparameter „resultName“. Mit dem Parameter „resultName“ wird der Variablenname für die weitere Verwendung angegeben.

Durch das Tag wird die Variable „hasMore“ definiert, mit der innerhalb des Tags überprüft werden kann, ob noch Elemente folgen.

Hinweis: Bei der Verwendung von <fsi:iterateResults> ist darauf zu achten, dass die Liste des nächst höheren <fsi:search>-Tags (innerstes Tag) als Grundlage dient:

Beispiel:

```
<fsi:search resultName="mL">
  <fsi:query>
    <QUERY entityType="movie" />
  </fsi:query>

  <c:forEach items="${mL}" var="m">

    <select name="<fsi:ref value='m.rating' />">

      <fsi:search>
        <fsi:query>
          <QUERY entityType="rating" />
        </fsi:query>

        <fsi:iterateResults resultName="r">
            ${r.abbreviation} (${r.name})
        </fsi:iterateResults>
      </fsi:search>

    </select>

  </c:forEach>
</fsi:search>
```

Im Beispiel wird für <fsi:iterateResults> die Menge der Datensätze der Tabelle „rating“ verwendet.

Attribute:

Attribut	Erwarteter Wert	Pflichtparameter
resultName	Variablenname zur weiteren Verwendung innerhalb des Tags.	Ja

Variablen:

Variable	Bedeutung	Rückgabedatentyp
hasMore	Liefert zurück, ob noch weitere Elemente in der Liste enthalten sind.	boolean

3.2.1.9 Informationen zur Abfrage <fsi:getQueryDetails>

Das Tag <fsi:getQueryDetails> liefert Informationen zur aktuellen Datensatzsuche (<fsi:search>) zurück.

Innerhalb des Tags stehen die Variablen „pageNo“, „pageSize“, „parameter“, „totalPages“ und „totalResults“ zur Verfügung.

Variablen:

Variable	Bedeutung	Rückgabedatentyp
pageNo	Nummer der aktuell angezeigten (virtuellen) Seite.	Integer
pageSize	Anzahl der auf der Seite darzustellenden Elemente.	Integer
parameter	Liefert alle Suchparameter zurück.	Map
totalPages	Anzahl aller (virtuellen) Seiten (Berechnungsgrundlage: Anzahl der Datensätze und „pageSize“)	Integer
totalResults	Anzahl aller Elemente (über alle Seiten).	Integer

Beispiel:

```
<fsi:search>
  <fsi:query>
    <QUERY entityType="movie" />
  </fsi:query>

  <fsi:getQueryDetails>
      <%= totalResults %> Filme gefunden!
  </fsi:getQueryDetails>
</fsi:search>
```

3.2.2 Entfernen einer Abfrage (<fsi:clearQuery>)

Mit dem Tag <fsi:clearQuery> kann eine Aufteilung auf virtuelle Seiten (über das Query-Servlet oder <fsi:search>) aufgehoben werden. Die Abfrageergebnisse werden dann aus der HTTP-Session des Benutzers entfernt, um den für die Aufteilung benötigten Speicher wieder freizugeben.

Das Tag verfügt über den Parameter „if“. Der Parameter erwartet als Wert, die Angabe des Request-Parameter-Namens. Wird dieser optionale Parameter angegeben, prüft das Tag vor dem Entfernen der Aufteilung, ob ein solcher Request-Parameter existiert.

```
<fsi:clearQuery if="resetNavigation" />
```

Wird die Seite aufgerufen und ist beim Aufruf der Request-Parameter „resetNavigation“ vorhanden, wird die Aufteilung entfernt.

Attribute:

Attribut	Erwarteter Wert	Pflichtparameter
if	Angabe eines Request-Parameter-Namens	Nein

3.2.3 Vergleichsoperationen

Folgende Vergleichsoperationen sind verfügbar:

- <c:if> If-Then-Ausdruck (siehe Kapitel 3.2.3.1 Seite 30)
- <c:choose> If-Then-Else-Ausdruck (siehe Kapitel 3.2.3.2 Seite 32)
- <fsi:contains> Contains-Ausdruck (siehe Kapitel 3.2.3.3 Seite 33)
- <fsi:matches> Matches-Ausdruck (siehe Kapitel 3.2.3.4 Seite 34)

3.2.3.1 If-Then-Ausdruck (<c:if>)

Mit dem <c:if>-Tag der JSTL[7] kann eine Bedingung überprüft werden. Trifft die Bedingung zu („if“), so wird die Anweisung innerhalb des Tags ausgeführt („then“):

```
<c:if test="BEDINGUNG">
  RUMPF
</c:if>
```

In der Bedingung können die folgenden Vergleichsoperatoren verwendet werden:

```
== oder eq
!= oder ne
<  oder lt
>  oder gt
<= oder le
>= oder ge
```

Soll überprüft werden, ob der Inhalt eines String-Objekts „v“ die Zeichenkette „WERT“ enthält, so lautet die Syntax:

```
<c:if test="${v == 'WERT'}">
  ...
</c:if>
```

Wichtig: In älteren Versionen wurde ein if-Then-Ausdruck mit den Tags <fsi:if>, <fsi:equals> und <fsi:then> realisiert. Diese Tags werden nicht mehr unterstützt. Die alte Syntax lässt sich jedoch in die neue Syntax überführen:

Alte Syntax:

```
<fsi:if>
  <fsi:equals object1="user.login" value2="Admin" />
  <fsi:then> [X] </fsi:then>
</fsi:if>
```

Neue Syntax:

```
<c:if test="${user.login == 'Admin'}"> [X] </c:if>
```

Weiterhin können in der Bedingung noch folgende logische Operatoren verwendet werden:

```
&& oder and
|| oder or
!  oder not
```

[7] JavaServer Pages Standard Tag Library – Informationen zu JSTL siehe http://wikipedia.org/wiki/JSTL

Beispiel für die Verwendung von logischen Operatoren:

```
<c:if test="${user.login == 'Admin' && user.isActive}"> [X] </c:if>
```

Wichtig: Als logische Operatoren standen in älteren Version die Tags <fsi:and> und <fsi:or> zur Verfügung. Da die Tags nicht mehr unterstützt werden, sollten die Ausdrücke in die aktuelle Syntax überführt werden.

Alte Syntax:

```
<fsi:if>
  <fsi:and>
    <fsi:equals object1="user.login" value2="Admin" />
    <fsi:equals object1="user" object2="User(12)" />
  </fsi:and>
  <fsi:then> [X] </fsi:then>
</fsi:if>
```

Neue Syntax:

```
<fsi:ref value="User(12)" resultName="u" />
<c:if test="${user.login == 'Admin' && user == u}"> [X] </c:if>
```

Wichtig: Die Prüfung von zwei Objekten bzw. Werten auf Gleichheit (in älteren Version) mit dem Tag <fsi:equals> ist nicht mehr möglich. Die Prüfung auf Gleichheit kann über den logischen Operator „==“ realisiert werden. Zur Überführung der alten in die neue Syntax, vergleiche die folgenden Beispiele:

Alte Syntax:

```
<fsi:if>
  <fsi:or>
      <fsi:equals object1="user.login" value2="Admin" />
      <fsi:equals object1="user" object2="User(12)" />
  </fsi:or>
  <fsi:then> [X] </fsi:then>
</fsi:if>
```

Neue Syntax:

```
<fsi:ref value="User(12)" resultName="u" />
<c:if test="${user.login == 'Admin' || user == u}"> [X] </c:if>
```

3.2.3.2 If-Then-Else-Ausdruck (<c:choose>)

Über das Tag <c:choose> der JSTL[8] kann ein If-Then-Else-Ausdruck realisiert werden. Das Tag <c:choose> verfügt über keinerlei Parameter.

Es kann:

- einen oder mehrere <c:when>-Tags und
- kein oder ein <c:otherwise>-Tag enthalten.

<c:when> verfügt über den Pflichtparameter „test“, mit dem eine Bedingung angegeben wird. Trifft die Bedingung zu, so wird der Inhalt von <c:when> ausgeführt.

Das <c:otherwise>-Tag muss als letztes Tag innerhalb von <c:otherwise> angegeben werden. Der Inhalt des Tags wird ausgeführt, wenn keine der angegebenen Bedingungen der <c:when>-Tag zutrifft.

```
<c:choose>
  <c:when test="BEDINGUNG_1">
    RUMPF_1
  </c:when>

  <c:when test="BEDINGUNG_2">
    RUMPF_2
  </c:when>

  ...

  <c:otherwise>
    RUMPF_N
  </c:otherwise>
</c:choose>
```

[8] JavaServer Pages Standard Tag Library – Informationen zu JSTL siehe http://wikipedia.org/wiki/JSTL

3.2.3.3 Contains-Ausdruck (<fsi:contains>)

Über das JSP-Tag <fsi:contains> kann geprüft werden, ob ein Wert oder Objekt in einer Liste enthalten ist.

Mit dem Pflichtparameter „list“ wird eine Liste für den Vergleich an das Tag übergeben. Als Wert für den Pflichtparameter „resultName“ ist der Variablenname für die weitere Verwendung anzugeben.

Die Alternativparameter „value“ und „object“ enthalten den zu vergleichenden Wert bzw. das zu vergleichende Objekt.

Wichtig: Die beiden Parameter „value“ und „object“ können nicht gleichzeitig angegeben werden.

Zusätzlich kann mit dem optionalen Parameter „schema“ das Schema für den Vergleich angegeben werden.

Wichtig: Da die Verwendung des Attributes „schema“ (gerade bei Verschachtelungen) fehleranfällig ist, wird die Verwendung des <fsi:setSchema>-Tags empfohlen (vgl. Kapitel 3.2.1.1 Seite 15).

Beispiel für die Prüfung, ob ein Wert in einer Liste enthalten ist:

```
<fsi:contains list='l'
              value='WERT'
              resultName="hasW" />
<c:if test="${hasW}">
  [X]
</c:if>
```

Beispiel für die Prüfung, ob ein Objekt in einer Liste enthalten ist:

```
<fsi:contains list='m.actors'
              object='person(93)'
              resultName="hasP93" />
<c:if test="${hasP93}">
  [X]
</c:if>
```

Attribute:

Attribut	Erwarteter Wert	Pflichtparameter
resultName	Variablenname zur weiteren Verwendung innerhalb des Tags.	Ja
list	Objekt- oder Attributreferenz der Liste, beispielsweise „m.actors“.	Ja
object	Vergleichs-Objektreferenz oder Vergleichs-Attributreferenz, z. B. „user(12)“.	Alternativparameter zu „value“
value	Vergleichswert	Alternativparameter zu „object“
schema	eindeutiger Bezeichner des Schemas aus der Modul-Konfiguration	Nein

3.2.3.4 Matches-Ausdruck (<fsi:matches>)

Über das JSP-Tag <fsi:matches> kann geprüft werden, ob die Schnittmenge zweier Listen nicht leer ist.

Mit dem Pflichtparameter „resultName“ wird der Variablenname für die weitere Verwendung angegeben.

Für die Übergabe der Listen existieren zwei Alternativparameter. Der erste Parameter wird mit dem Postfix „1“ gekennzeichnet und der zweite mit „2“. Für die Angabe einer Objektreferenz zu einer Liste ist das Präfix „list“ und für die Angabe einer Werteliste „values“ zu verwenden.

Beispiel: Es soll geprüft werden, ob zwei Wertelisten eine Schnittmenge haben. Die erste Liste enthält die Werte „a“, „b“ und „c“ und die zweite Liste die Werte „b“, „c“ und „d“. Da es sich bei beiden Listen um Wertelisten handelt, ist für beide das Präfix „values“ zu verwenden. Für die erste Liste wird das Postfix „1“ und für die zweite „2“ verwendet. Daraus resultiert für die erste Werteliste der Parameter „values1“ und für die zweite Werteliste der Parameter „values2“.

Die Werte der Liste für die Parameter „values1“ und „values2“ werden kommasepariert angegeben. Da einige Java-Klassen beim Aufruf der Methode

.toString() ein „["-Zeichen voranstellen, bzw. ein „]“-Zeichen anhängen, kann die Liste auch optional zwischen diesen beiden Zeichen eingefasst werden.

Beispiel:

```
<fsi:matches values1="a,b,c"
             values2="[b,c,d]"
             resultName="hasIntersection" />
<c:if test="${hasIntersection}">
  [X]
</c:if>
```

Weiterhin ist es möglich mit dem Parameter „schema“ das Schema anzugeben, in dem gesucht werden soll:

```
<fsi:matches values1="a,b,c"
             values2="[b,c,d]"
             resultName="hasIntersection"
             schema="movies" />
<c:if test="${hasIntersection}">
  [X]
</c:if>
```

Attribute:

Attribut	Erwarteter Wert	Pflichtparameter
resultName	Variablenname zur weiteren Verwendung innerhalb des Tags.	Ja
list1	Objekt- oder Attributreferenz der Liste, z. B. „m.actors“.	Alternativparameter zu „value1“
value1	Vergleichswert	Alternativparameter zu „list1“
list2	Objekt- oder Attributreferenz der Liste, z. B. „m.actors“.	Alternativparameter zu „value2“
value2	Vergleichswert	Alternativparameter zu „list2“
schema	eindeutiger Bezeichner des Schemas aus der Modul-Konfiguration	Nein

3.2.4 Navigation

Mit dem Tag <fsi:search> können einzelne Datensätze auf virtuelle Seiten verteilt werden. Zur Navigation zwischen den einzelnen virtuellen Seiten bietet FirstSpirit DynamicDatabaseAccess Tags zur Erzeugung einer Navigationsleiste an.

Eine Navigationsleiste besteht aus drei Bereichen:

- Erste virtuelle Seite bzw. obere Navigationsgrenze (fsi:navTop)
- Teilbereich bzw. Navigationsausschnitt virtueller Seiten, ausgehend von der aktuellen Seite (fsi:navFrame)
- Letzte virtuelle Seite bzw. untere Navigationsgrenze (fsi:navBottom)

Beispiele von Navigationsleisten (die aktuelle Seite ist jeweils fett hervorgehoben):

- Anzeige der ersten und letzten virtuellen Seite und des Teilbereichs:
  [1] … [6] [7] **[8]** [9] [10] … [21]
- Anzeige der letzten virtuellen Seite und des Teilbereichs
  (die erste Seite wird im Teilbereich angezeigt):
  [1] **[2]** [3] [4] [5] … [21]
- Anzeige der ersten virtuellen Seite und des Teilbereichs
  (die letzte Seite wird im Teilbereich angezeigt):
  [1] … [17] [18] [19] **[20]** [21]
- Anzeige des Teilbereichs
  (die erste und die letzte Seite werden im Teilbereich angezeigt):
  [1] [2] [3] [4] [5]

3.2.4.1 Navigationsleiste (<fsi:navigation>)

Das Tag <fsi:navigation> erzeugt eine Navigationsleiste. Innerhalb des Tags <fsi:navigation> können die Tags <fsi:navTop>, <fsi:navFrame> und <fsi:navBottom> verwendet werden.

Über den Parameter „frameSize“ kann definiert werden, wie viele virtuellen Seite im Teilbereich (<fsi:navFrame) dargestellt werden sollen. Wird der Parameter nicht angegeben, so wird der Standardwert „5“ verwendet.

Attribute:

Attribut	Erwarteter Wert	Pflichtparameter
frameSize	Anzahl der in <fsi:navFrame> darzustellenden virtuelle Seiten (Ganzzahl); Standardwert: „5“	Nein

Beispiel:

```
<fsi:search pageSize="2">
  <fsi:query>
    <QUERY entityType="movie" />
  </fsi:query>

  <fsi:navigation frameSize="3">

    <fsi:navTop resultName="isVisible">
      <c:if test="${isVisible}">
        <a href="${navUrl}">[${navPageNo}]</a> ...
      </c:if>
    </fsi:navTop>

    <fsi:navFrame resultName="isVisible">
      <c:choose>
        <c:when test="${!isVisible}">
          <b>[${navPageNo}]</b>
        </c:when>
        <c:otherwise>
          <a href="${navUrl}">[${navPageNo}]</a>
        </c:otherwise>
      </c:choose>
    </fsi:navFrame>

    <fsi:navBottom resultName="isVisible">
      <c:if test="${isVisible}">
        ... <a href="${navUrl}">[${navPageNo}]</a>
      </c:if>
    </fsi:navBottom>

  </fsi:navigation>
  ...
</fsi:search>
```

3.2.4.2 Erste / Letzte virtuelle Seite (<fsi:navTop>, <fsi:navBottom>)

Die beiden JSP-Tags <fsi:navTop> und <fsi:navBottom> stehen innerhalb des Tags <fsi:navigation> zur Verfügung.

Über den Parameter „resultName“ kann der Variablenname für die weitere Verwendung definiert werden. Mit dieser Variable kann überprüft werden, ob die erste (<fsi:navTop>) oder letzte (<fsi:navBottom>) virtuelle Seite nicht innerhalb des Teilbereiches (<fsi:navFrame>) dargestellt wird. Die Variable liefert „true“ zurück, wenn die erste bzw. letzte virtuelle Seite nicht im Teilbereich dargestellt wird. So kann beispielsweise eine doppelte Ausgabe verhindert werden, wenn die erste bzw. letzte Seite bereits im Teilbereich dargestellt wird.

```
...
<fsi:navTop resultName="isVisible">
  <c:if test="${isVisible}">
    ...
  </c:if>
</fsi:navTop>
...
```

Innerhalb der beiden Tags stehen die Variablen „navPageNo“ und „navUrl“ zur Verfügung. Mit „navPageNo“ kann die Nummer und mit „navUrl“ der URL der virtuellen Zielseite ausgegeben werden. Zusätzlich kann die Anzahl der virtuellen Seiten mit „navPageSize“ ausgegeben werden

Variablen:

Variable	Bedeutung	Rückgabedatentyp
navPageNo	Nummer der virtuelle Zielseite	Integer
navUrl	URL der virtuellen Zielseite (inkl. Parameter)	String
navPageSize	Anzahl der virtuellen Seiten	Integer

Beispiel:

```
<fsi:search pageSize="2">
  <fsi:query>
    <QUERY entityType="movie" />
  </fsi:query>

  <fsi:navigation frameSize="3">

    <fsi:navTop resultName="isVisible">
      <c:if test="${isVisible}">
        <a href="${navUrl}">[${navPageNo}]</a> ...
      </c:if>
    </fsi:navTop>

    <fsi:navFrame resultName="isVisible">
      <c:choose>
        <c:when test="${!isVisible}">
          <b>[${navPageNo}]</b>
        </c:when>
        <c:otherwise>
          <a href="${navUrl}">[${navPageNo}]</a>
        </c:otherwise>
      </c:choose>
    </fsi:navFrame>

    <fsi:navBottom resultName="isVisible">
      <c:if test="${isVisible}">
        ... <a href="${navUrl}">[${navPageNo}]</a>
      </c:if>
    </fsi:navBottom>

  </fsi:navigation>

  ...

</fsi:search>
```

3.2.4.3 Teilbereich aller virtuellen Seiten (<fsi:navFrame>)

Mit dem Tag <fsi:navFrame>, das innerhalb des Tags <fsi:navigation> zur Verfügung steht, werden alle virtuellen Seiten im Teilbereich dargestellt.

Die Anzahl der anzuzeigenden Seiten wird durch das Attribut „frameSize“ festgelegt.

Über den Parameter „resultName“ kann der Variablenname für die weitere Verwendung festgelegt werden. Mit dieser Variable kann überprüft werden, ob die im Teilbereich darzustellende Seite, nicht der aktuellen Seite entspricht. Die Variable liefert „true“ zurück, wenn die vom Teilbereich anzuzeigende Seite, nicht die aktuelle Seite ist.

```
...
<fsi:navFrame resultName="isVisible">
  <c:if test="${isVisible}">
    ...
  </c:if>
</fsi:navFrame>
...
```

Innerhalb des Tags sind die beiden Variablen „navPageNo“ und „navUrl“ verfügbar. Mit „navPageNo“ kann die Nummer und mit „navUrl“ der URL, der vom Teilbereich anzuzeigenden, virtuellen Seite ausgegeben werden. Zusätzlich kann die Anzahl der virtuellen Seiten mit „navPageSize“ ausgegeben werden.

Variablen:

Variable	Bedeutung	Rückgabedatentyp
navPageNo	Nummer der virtuelle Zielseite	Integer
navUrl	URL der virtuellen Zielseite (inkl. Parameter)	String
navPageSize	Anzahl der virtuellen Seiten	Integer

Beispiel:

```
<fsi:search pageSize="2">
  <fsi:query>
    <QUERY entityType="movie" />
  </fsi:query>

  <fsi:navigation frameSize="3">

    <fsi:navTop resultName="isVisible">
      <c:if test="${isVisible}">
        <a href="${navUrl}">[${navPageNo}]</a> ...
      </c:if>
    </fsi:navTop>

    <fsi:navFrame resultName="isVisible">
      <c:choose>
        <c:when test="${!isVisible}">
          <b>[${navPageNo}]</b>
        </c:when>
        <c:otherwise>
          <a href="${navUrl}">[${navPageNo}]</a>
        </c:otherwise>
      </c:choose>
    </fsi:navFrame>

    <fsi:navBottom resultName="isVisible">
      <c:if test="${isVisible}">
```

```
        ... <a href="${navUrl}">[${navPageNo}]</a>
      </c:if>
    </fsi:navBottom>

  </fsi:navigation>

  ...

</fsi:search>
```

3.3 Datensätze ändern / erstellen (Store-Servlet)

Nach der Konfiguration des FirstSpirit Moduls DynamicDatabaseAccess steht automatisch ein Servlet für die Speicherung von Daten zur Verfügung – das „Store-Servlet“.

Mit Hilfe des Servlets können neue Datensätze angelegt oder vorhandene Datensätze geändert werden.

Wichtig: Ist für das Schema die Konfigurationsoption „Freigabestand nutzen“ aktiviert, können keine Daten gespeichert werden (siehe Kapitel 2.3.1 Seite 9)!

Üblicherweise wird das Speichern von Werten in JSP-Seiten über HTML-Eingabekomponenten oder über JavaScript-basierende Editoren (z.B. TinyMCE, HTMLArea, FCK Editor u.ä.) in einem HTML-Formular realisiert. Jede HTML-Eingabekomponente verfügt über einen Parameter „name“ (Bezeichnung) und einen „value“ (Wert). Bei der Speicherung versucht das Store-Servlet, den übertragenen Wert anhand des „name“-Parameters zu speichern.

Damit das Store-Servlet die Werte den richtigen Spalten bzw. Datensätzen in der Datenbank zuordnen kann, benötigt es einen eindeutigen Verweis auf die Spalte und den Datensatz. Um einen eindeutigen Verweis zu einem Attribut bzw. Objekt zu erhalten, kann das <fsi:ref>-Tag verwendet werden (siehe Kapitel 3.2.1.3).

Beispiel:

```
<input type="text"
       name="<fsi:ref value='m.title' />"
       value="<c:out value='${m.title}'
                     escapeXml='true' />" />
```

Zur Anlage eines neuen Datensatzes wird der Schlüsselbegriff „*“ und der Name der Tabelle des Schemas, als Grundlage zur Bildung der absoluten Objektreferenz verwendet:

```
TABELLENNAME(*)
```

Für die einzelnen Eingabefelder wird der absoluten Objektreferenz, der gewünschte Spaltenname angehängt, z.B.:

```
<input type="text"
       name="<fsi:ref value='movie(*).name' />"
       value="" />
```

Der Aufruf des Store-Servlets erfolgt mit:
„<%= application.getContextPath() %>/do.store".

In einem HTML-Formular wird das Tag <form> verwendet:

```
<form method="post"
      action="<%= application.getContextPath() %>/do.store">
  ...
</form>
```

Wichtig: Für die HTTP-Übertragungsmethode ist „post" zu verwenden, da die Länge der Parameternamen und -werte sehr lang sein kann!

Zusätzlich werden die folgenden Request-Parameter gesondert behandelt:

- „schema" (siehe Kapitel 3.3.1)
- „url_ok" (siehe Kapitel 3.3.2)
- „url_error" (siehe Kapitel 3.3.2)
- „pw_params" (siehe Kapitel 3.3.4)
- „url_pw_error" (siehe Kapitel 3.3.4)

3.3.1 Angabe eines Schemas

Mit dem optionalen Parameter „schema" kann der Name einer Schema-Konfiguration angegeben werden, die zur Speicherung genutzt werden soll. Wird der Parameter nicht angegeben, wird das Standard-Schema verwendet.

```
<input type="hidden" name="schema" value="movies" />
```

3.3.2 Behandlung bei erfolgreicher bzw. fehlerhafter Speicherung

Der Parameter „url_ok“ ermöglicht die Angabe eines URL, der nach erfolgreicher Speicherung angezeigt werden soll. Der Parameter „url_error“ wird genutzt, um einen URL anzugeben, der bei einem Fehler angezeigt werden soll.

```
<input type="hidden"
       name="url_ok"
       value="$CMS_REF(pageref:"saved")$" />
<input type="hidden"
       name="url_error"
       value="$CMS_REF(pageref:"error")$" />
```

Durch die Verwendung des Schlüsselbegriffes „forward:“ im Parameter url_error, kann das Verhalten im Fehlerfall angepasst werden. Wird "forward:" vor den URL gesetzt, wird ein Forward ausgeführt, sonst ein Redirect (Standardwert):

```
<input type="hidden"
      name="url_error"
      value="forward:..." />
```

Wird bei der Speicherung der Daten eine Exception ausgelöst, so ist diese nach dem Aufruf der Fehlerseite verfügbar. Auf die Exception kann durch den JSP-Ausdruck `request.getAttribute("exception");` zugegriffen werden.

3.3.3 Formatieren von Datums- und Zahlenfeldern

Für Datums- und Zahlenfeldern kann ein Format zum Parsen der Eingabe angegeben werden. Hierfür muss hinter dem Tag <fsi:ref> ".format" angegeben werden:

```
<input type="name" name="<fsi:ref value='movie(*).release_date' />"
value="01/01/70" size="50" maxlength="25" />
<input type="hidden" name="<fsi:ref value='movie(*).release_date'
/>.format" value="MM/dd/yy" />
```

3.3.4 Speichern von Passwörtern

Bei der Verwaltung von Benutzerdaten wird häufig das Login-Passwort in der Datenbank hinterlegt. Bei einer Änderung durch den Benutzer, sollte das neue Passwort sinnvollerweise zweimal angeben werden (um Tippfehler abzufangen).

Das Store-Servlet bietet die Möglichkeit, die doppelte Angabe von Passwörtern zu überprüfen. Hierzu wird der Parameter „pw_params“ verwendet. Dabei wird das HTML-Eingabefeld zweimal in der Seite eingefügt, wobei das Attribut „name“ für beide Eingabefelder den gleichen Wert haben muss. Als Wert für „pw_params“ wird der (identische) Name der beiden Eingabefelder („name“-Attribut des <input>-Tags) angegeben:

```
<fsi:ref value="user(*).password" resultName="pwd" />

Neues Passwort: <input type="text" name="${pwd}" value="" />
Wiederholung:   <input type="text" name="${pwd}" value="" />

<input type="hidden" name="pw_params" value="${pwd}" />
```

Über den Parameter „url_pw_error“ kann ein URL angegeben werden, der angezeigt wird, wenn die beiden Werte nicht übereinstimmen:

```
<input type="hidden"
       name="url_pw_error"
       value="$CMS_REF(pageref:"passwordmismatch")$" />
```

Request-Parameter:

Parameter	Erwarteter Wert
schema	eindeutiger Bezeichner des Schemas aus der Modul-Konfiguration
pw_params	identischer Name der beiden Eingabefelder („name“-Attribut eines <input>-Tags)
url_ok	URL (bei erfolgreicher Speicherung)
url_error	URL (bei fehlerhafter Speicherung)
url_pw_error	URL (bei abweichenden Werten der beiden Passwortfelder)

Einfaches Beispiel:

```
$CMS_IF(#global.preview)$
  <form method="post"
        action="<%=
          application.getContextPath()
        %>/do.store">
    <table>
      <tr>
        <td valign="top">*</td>
        <td valign="top">
          <input type="text"
                 name="<fsi:ref value='User(*).Name' />"
                 value="" />
        </td>
        <td>
          <input type="hidden"
                 name="url_ok"
                 value="$CMS_REF(#global.node)$" />
          <input type="submit"
                 value="Add" />
        </td>
      </tr>
    </table>
  </form>
$CMS_END_IF$
```

3.4 Datensätze löschen (Delete-Servlet)

Über das Delete-Servlet können Datensätze gelöscht werden. Das Servlet ist direkt nach der Installation von FirstSpirit DynamicDatabaseAccess verfügbar.

Wichtig: Ist für das Schema die Konfigurationsoption „Freigabestand nutzen“ aktiviert, so können keine Daten gelöscht werden (siehe Kapitel 2.3.1 Seite 9)!

Für das Löschen eines Datensatzes in der Datenbank, benötigt das Delete-Servlet einen eindeutigen Verweis auf den Datensatz. Der eindeutige Verweis muss mit dem Request-Parameter „delete“ an das Delete-Servlet übergeben werden. Um einen eindeutigen Verweis zu einem Attribut bzw. Objekt zu erhalten kann das <fsi:ref>-Tag verwendet werden (siehe Kapitel 3.2.1.3).

Beispiel:

```
...
<c:forEach items="${list}" var="item">
  ...
  <input type="hidden"
         name="delete"
         value="<fsi:ref value='item' />" />
  ...
</c:forEach>
...
```

Der Aufruf des Delete-Servlets erfolgt mit:

```
"<%= application.getContextPath() %>/do.delete"
```

In einem HTML-Formular wird das Tag <form> verwendet:

```
<form method="post"
      action="<%= application.getContextPath() %>/do.delete">
  ...
</form>
```

Wichtig: Für die HTTP-Übertragungsmethode ist „post“ zu verwenden, da die Länge der Parameternamen und -werte sehr lang sein kann!

Zusätzlich werden die folgenden Request-Parameter gesondert behandelt:

- „schema“ (siehe Kapitel 3.3.1)
- „url_ok“ (siehe Kapitel 3.3.2)
- „url_error“ (siehe Kapitel 3.3.2)

Request-Parameter:

Parameter	Erwarteter Wert
delete	Eindeutige Referenz zum zu löschenden Datensatz, z.B. `<fsi:ref value='item' />`
schema	eindeutiger Bezeichner des Schemas aus der Modul-Konfiguration
url_ok	URL (bei erfolgreicher Speicherung)
url_error	URL (bei fehlerhafter Speicherung)

Einfaches Beispiel:

```
<fsi:search resultName="list">

  <fsi:query>
    <QUERY entityType="User">
      <ORDER>
        <ORDERCRITERIA attribute="Name" descending="0" />
      </ORDER>
    </QUERY>
  </fsi:query>

  <table>
    <c:forEach items="${list}" var="item">
      <tr>
        <td valign="top">${item.fs_id}</td>
        <td valign="top">${item.Name}</td>

        $CMS_IF(#global.preview)$
          <form method="post"
                action="<%=
                  application.getContextPath()
                %>/do.delete">
            <td>
              <input type="hidden"
                     name="delete"
                     value="<fsi:ref value='item' />" />
              <input type="hidden"
                     name="url_ok"
                     value="$CMS_REF(#global.node)$" />
              <input type="submit"
                     value="Del" />
            </td>
          </form>
        $CMS_END_IF$

      </tr>
    </c:forEach>
  </table>
</fsi:search>
```

3.5 Abfragen (Query-Servlet)

Zusätzlich zu den Tags <fsi:search> und <fsi:query> können Datenbankabfragen auch mit dem Query-Servlet durchgeführt werden. Das Ergebnis einer solchen Abfrage wird in der Benutzersession gespeichert und kann mit <fsi:iterateResults> ausgegeben werden.

Der Aufruf des Query-Servlets erfolgt mit:

```
 "<%= application.getContextPath() %>/do.query"
```

In einem HTML-Formular wird das Tag <form> verwendet:

```
<form method="post"
      action="<%= application.getContextPath() %>/do.query">
  ...
</form>
```

Wichtig: Für die HTTP-Übertragungsmethode ist „post“ zu verwenden, da die Länge der Parameternamen und -werte sehr lang sein kann!

Analog zum <fsi:search>-Tag (Parameter „pageSize“) verfügt auch das Query-Servlet über einen Parameter, um Datensätze auf verschiedene virtuelle Seiten zu verteilen. Dieser Request-Parameter heißt „page_size“. Als Wert erwartet der Parameter die Angabe, wie viele Datensätze auf einer virtuellen Seite dargestellt werden sollen. Wird der Parameter nicht angegeben, wird der Standardwert 10 (Datensätze pro virtueller Seite) verwendet.

```
<input type="hidden" name="page_size" value="1" />
```

Für die Darstellung einer Navigationsleiste (zur Navigation durch die einzelnen virtuellen Seiten) stehen die folgenden Tags zur Verfügung:

- <fsi:navigation> (siehe Kapitel 3.2.4.1 Seite 36)
- <fsi:navTop> (siehe Kapitel 3.2.4.2 Seite 38)
- <fsi:navFrame> (siehe Kapitel 3.2.4.3 Seite 39)
- <fsi:navBottom> (siehe Kapitel 3.2.4.2 Seite 38)

Mit dem Request-Parameter „query_xml“ wird dem Query-Servlet das XML einer Abfrage übergeben. Die Syntax des XMLs ist identisch zur Syntax, die innerhalb der Header-Funktion „contentSelect“ in Vorlagen verwendet werden kann (weitere Informationen siehe FirstSpirit-Online-Dokumentation[9]):

[9] Bereich: .../vorlagenentwicklung/vorlagensyntax/funktionen/im_header/contentselect/contentselect.html

```
<input type="hidden" name="query_xml" value="&lt;QUERY
entityType=&quot;person&quot;&gt;&lt;FILTERPARAM
parameter=&quot;person&quot; datatype=&quot;java.lang.String&quot;
value=&quot;%&quot; /&gt;&lt;LIKE attribute=&quot;surname&quot;
parameter=&quot;person&quot; /&gt;&lt;/QUERY&gt;" />
```

Wichtig: Um Probleme mit dem HTML-Tag <input> zu vermeiden, sind die XML- bzw. HTML-Zeichen „<“, „>“ und „“ der Abfrage zu quotieren, d.h. „<“, „>“ und „"“!

Werden in einer XML-Abfrage Parameter verwendet (Tag <FILTERPARAM>), so können die einzelnen Werte der Parameter an das Query-Servlet übergeben werden. Die Syntax setzt sich aus dem Schlüsselbegriff „param.“ und dem Wert des Parameters „parameter“ des Tags <FILTERPARAM> zusammen:

```
<input type="hidden" name="query_xml" value="&lt;QUERY
entityType=&quot;person&quot;&gt;&lt;FILTERPARAM
parameter=&quot;person&quot; datatype=&quot;java.lang.String&quot;
value=&quot;%&quot; /&gt;&lt;LIKE attribute=&quot;surname&quot;
parameter=&quot;person&quot; /&gt;&lt;/QUERY&gt;" />

<input type="text" name="param.person" value="" />
```

Zusätzlich werden die folgenden Request-Parameter gesondert behandelt:

- „schema“ (siehe Kapitel 3.3.1)
- „url_ok“ (siehe Kapitel 3.3.2)
- „url_error“ (siehe Kapitel 3.3.2)

Request-Parameter:

Parameter	Erwarteter Wert
query_xml	XML-Abfrage, z.B. "<QUERY entityType="person"><FILTERPARAM parameter="person" datatype="java.lang.String" value="%" /><LIKE attribute="surname" parameter="person" /></QUERY>"
page_size	Anzahl der Elemente, die pro Seite dargestellt werden sollen; Standardwert: „10“
schema	eindeutiger Bezeichner des Schemas aus der Modul-Konfiguration
url_ok	URL (bei erfolgreicher Speicherung)

url_error	URL (bei fehlerhafter Speicherung)

Einfaches Beispiel:

```
<fsi:iterateResults resultName="p">
  <p>
    ${p.surname}, ${p.name}
  </p>
</fsi:iterateResults>
...
<form action="<%= application.getContextPath() %>/do.query"
      method="post">

  <input type="hidden"
         name="query_xml" value="&lt;QUERY
entityType=&quot;person&quot;&gt;&lt;FILTERPARAM
parameter=&quot;person&quot; datatype=&quot;java.lang.String&quot;
value=&quot;%&quot; /&gt;&lt;LIKE attribute=&quot;surname&quot;
parameter=&quot;person&quot; /&gt;&lt;/QUERY&gt;" />

  <input type="text"
         name="param.person"
         value="" />

  <input type="hidden"
         name="page_size"
         value="1" />

  <input type="hidden"
         name="schema"
         value="movies" />

  <input type="hidden"
         name="url_ok"
         value="$CMS_REF(#global.node)$" />

  <input type="submit"
         value="Submit" />
</form>
```

4 Rechtliche Hinweise

Das Modul „FirstSpirit DynamicDatabaseAccess“ ist ein Produkt der e-Spirit AG, Dortmund, Germany.

Für die Verwendung des Moduls gilt gegenüber dem Anwender nur die mit der e-Spirit AG vereinbarte Lizenz.

Details zu möglicherweise fremden, nicht von der e-Spirit AG hergestellten, eingesetzten Software-Produkten, deren eigenen Lizenzen und gegebenenfalls Aktualisierungs-Informationen, finden Sie auf der Startseite jedes FirstSpirit™-Servers im Bereich „Rechtliche Hinweise".