# Integrated relation\_json type

This type can be used to store multiple properties of the relations of a content item in solr.

In your solr mapping you can add the content like this:

<**type** name**="integrated.relation\_json"**>
 <**options**>
 <**string** key**="alias"**>solr\_alias</**string**>
 <**string** key**="relation\_id"**>integrated\_relation\_id</**string**>
 <**array** key**="properties"**>
 <**string** key**="id"**>id</**string**>
 <**string** key**="slug"**>slug</**string**>
 <**string** key**="title\_alias"**>title</**string**>
 </**array**>
 </**options**>
</**type**>

The alias is the name for the field in solr.

The relation\_id is the relationId of the relation. All the references of this relation will be stored in solr using json\_encode. In the properties you have to define the properties that will be stored in solr.

The properties work the same as integrated.fields, the key is the alias, the value is the property.

You have to use a json\_decode filter to read the values as an array. There is no twig json\_decode filter available in integrated yet, you can build your own twig filter or use php.

Example of json\_decode twig function:

**<?php

namespace** YourBundle\Twig\Extension;
**class** JsonDecodeExtension **extends** \Twig\_Extension
{

 */\*\*
 \** ***@return*** *array
 \*/* **public function** getFilters()
 {
 **return** [
 **new** \Twig\_SimpleFilter(**'json\_decode'**, [$this, **'jsonDecode'**]),
 ];
 }

 */\*\*
 \** ***@param*** *$jsonArray
 \** ***@return*** *mixed
 \*/* **public function** jsonDecode($jsonArray)
 {
 **return** json\_decode($jsonArray);
 }

 */\*\*
 \** ***@return*** *string
 \*/* **public function** getName()
 {
 **return 'json\_decode'**;
 }
}