# RelationalDatabaseConnector

The `RelationalDatabase` data connector generates multiple attributes from a relational database via a JDBC DataSource. The attributes are generated such that each attribute represents a column of the query result set. The ordered values represent the rows of the result set and each attribute will contain the same number of values, including any embedded nulls in the results. Nulls are represented explicitly with objects of type EmptyAttributeValue (note, this is a change from V2, which exposed null values as Java nulls in the attribute value collections).



## Schema Name and Location

This `xsi:type` is defined by the `urn:mace:shibboleth:2.0:resolver` schema [3.3], located at http://shibboleth.net/schema/idp/shibboleth-attribute-resolver.xsd.

Prior to V3.3 supplied plugins were defined by a schema type (xsi:type) in the `urn:mace:shibboleth:2.0:resolver:dc` namespace, the schema for which is located at http://shibboleth.net/schema/idp/shibboleth-attribute-resolver-dc.xsd. This is still supported, but every element or type in the `urn:mace:shibboleth:2.0:resolver:dc` namespace has an equivalently named (but not necessarily identical) version in the `urn:mace:shibboleth:2.0:resolver` namespace. The use of the `urn:mace:shibboleth:2.0:resolver` namespace also allows a relaxation of the ordering requirements of child elements to reduce strictness.

## Data Sources and Drivers

This connector uses a JDBC javax.sql.DataSource to connect to the database. The data source can be supplied via a number of techniques, but the recommended approach is to define one using Spring syntax in *global.xml* (or similar location) and use the `<BeanManagedConnection>` element, the reason being it can be easily shared across multiple connectors. If you need the ability to reload the data source's settings, the suggested approach is to create a new Spring file to contain the bean, and add it to the set of resources in *services.xml*



No matter where or how you define the data source, it is your responsibility to obtain and install the JDBC driver you want to use. The IdP does not come with any drivers, to avoid them becoming stale.

Whatever driver you use should generally be installed to **edit-webapp/WEB-INF/lib**, after which you will need to stop your container, rebuild the warfile, and restart the container. Failure to do so will lead to ClassNotFound exceptions.

### Connection Pooling

If you want to use connection pooling, the Apache DBCP library is a popular option. As their documentation notes, you will need to add both the *commons-dbcp2.jar* and *commons-pool2.jar* files to make use of this implementation. The DBCP library provides various data source implementations that wrap an actual database driver, and you will have to add the driver itself as well. A rudimentary example is included below, but be aware that there are a lot of options available and no particular "best practice" is implied.

## Configuration Reference

### Attributes

Any of the common attributes can be specified. In addition the following attributes may be specified:

| Name | Type | Default | Description |
|---|---|---|---|
| `noResultIsError` | boolean | false | Controls whether an empty result set is an error |
| `mappingStrategyRef` | Bean ID | | Bean ID of a MappingStrategy<java.sql.ResultSet> to process the result set in a pluggable way |
| `validatorRef` [3.2] | Bean ID | | Bean ID of a Validator to control what constitutes an initialization failure (set this to "shibboleth.NonFailFastValidator" to bypass connection attempt at config load time) |
| `executableSearchBuilderRef` [3.4] | Bean ID | | Bean ID of an ExecutableSearchBuilder<ExecutableStatement> to produce the SQL query to execute |
| `multipleResultsIsError` | boolean | false | Controls whether a result set with more than one row is an error |

| | | | |
|---|---|---|---|
| `queryTimeout` | XML Duration or milliseconds | | Timeout for the queries made against the database |
| `templateEngine` | Bean ID | | Bean ID of a org.apache.velocity.app.VelocityEngine to use for processing the SQL template |
| `readOnlyConnection` | boolean | true | Whether the DataConnector should be marked as readonly. If the DataConnector is shared with a subsystem which requires write access (via a <BeanManagedConnection>) this **must** bet set to false. |

## Child Elements

Any of the common child elements can be specified. In addition, the following may be specified.

| Name | Cardinality | Description |
|---|---|---|
| `<ContainerManagedConnection>` | Exactly 1<br><br>Not permitted if the `springResource` attribute is used | Connects to a database via a JNDI DataSource defined in the container |
| `<SimpleManagedConnection>` 3.4 | | Connects to a database via a JDBC DataSource defined explicitly with a simplified syntax. |
| `<ApplicationManagedConnection>` | | Connects to a database via a JDBC DataSource defined explicitly<br>Deprecated in V3.4 |
| `<BeanManagedConnection>` | | Connects to a database via an externally specified javax.sql.DataSource |
| `<QueryTemplate>` | 0 or 1 | The template of the SQL query to send to the database |
| `<Column>` | 0 or more | A series of remapping definitions which map a column name to an IdPAttribute ID |
| `<ResultCache>` | 0 or 1 | Defines how results should be cached |
| `<ResultCacheBean>` | | Bean ID (in the element content) defining how results should be cached as an externally defined com.google.common.cache.Cache<String,Map<String,IdPAttribute>> |

## Externally (Spring) Defined Content

If the `springResource` or `springResourceRef` attributes are specified, then the configuration of the data connector bean is delegated to the supplied resources. The system will create a factory for an RDBMSDataConnector object, and look for beans in the Spring resource(s) supplied that match the types of properties supported by that type and its parent classes. Note that since these are not public, but implementation classes, they are subject to change, which creates some risk during non-patch upgrades, so you must take additional precautions to use this feature.

In practice, the RDBMS Data Connector may be supplied with beans of the following types:

- DataSource
- ExecutableSearchBuilder<ExecutableStatement>
- com.google.common.cache.Cache<String,Map<String,IdPAttribute>>
- Validator
- MappingStrategy<ResultSet>

In addition native bean IDs can be injected as follows:

1. The data source can be specified as an externally defined bean via the `<BeanManagedConnection>` element (as a recommended replacement for either the `<ContainerManagedConnection>` or `<ApplicationManagedConnection>` elements).
2. (3.4+) The builder for the SQL query can be specified as an externally defined bean via the `executableSearchBuilderRef` attribute (as a replacement for the `<QueryTemplate>` element).
3. The mapping of column names can be specified as an externally defined bean via the `mappingStrategyRef` attribute (as a replacement for the `<Column>` elements).
4. The caching of results can be specified as an externally defined bean via the `<ResultCacheBean>` element (as a replacement for the `<ResultCache>` element).
5. Rarely, a non-default Velocity engine can be injected via the `templateEngine` attribute.

# Examples

**Simple DataConnector entirely in custom syntax**

```
 <DataConnector id="myDatabase" xsi:type="RelationalDatabase">
   <FailoverDataConnector ref="BackupDataseConnector"/>
   <SimpleManagedConnection
       jdbcDriver="org.hsqldb.jdbc.JDBCDriver" jdbcURL="jdbc:hsqldb:mem:RDBMSDataConnectorStore"
       jdbcUserName="SA" jdbcPassword="secret" />
   <QueryTemplate>
       <![CDATA[
         SELECT * FROM people WHERE userid='$resolutionContext.principal'
       ]]>
   </QueryTemplate>

   <Column columnName="homephone" attributeID="phonenumber" />

   <ResultCache elementTimeToLive="PT10S"/>
</DataConnector>
```

**Simple Data Connector using external bean**

```
<DataConnector id="myDatabase" xsi:type="RelationalDatabase" mappingStrategy="MappingBeanId">
  <BeanManagedConnection>DataConnectorBeanId</BeanManagedConnection>
   <QueryTemplate>
       <![CDATA[
         SELECT * FROM people WHERE userid='$resolutionContext.principal'
       ]]>
   </QueryTemplate>
  <ResultCacheBean>ResultCacheBeanId</ResultCacheBean>
</DataConnector>
```

The example below demonstrates a number of approaches:

- Use of a Spring file to define the various low-level objects, which could be referenced via `<DataConnector" xsi:type="RelationalDatabase" springResources="....." />`
- Use of a Spring file to define a data source which could be referenced via `<BeanManagedConnection>dataSource</BeanManagedConnection>`
- Use of the DBCP pooling library to wrap a database driver in a simple pool.

**Example of a springResources file**

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:p="http://www.springframework.org/schema/p"
    xmlns:c="http://www.springframework.org/schema/c"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema
/beans/spring-beans-3.1.xsd">

        <bean id="dataSource" class="org.apache.commons.dbcp2.BasicDataSource" destroy-method="close" p:
driverClass="org.mariadb.jdbc.Driver"
        p:jdbcUrl="jdbc:mysql://mysql.example.org:3306/shibboleth" p:user="admin" p:password="secret"
        p:maxTotal="20"
        p:maxIdle="5"
        p:maxWaitMillis="2000"
        p:testOnBorrow="true"
        p:validationQuery="select 1"
        p:validationQueryTimeout="5" />

        <!-- The rest of these beans would be unneeded for a simple BeanManagedConnection. -->

        <bean id="cacheBuilder" class="com.google.common.cache.CacheBuilder" factory-method="from">
                <constructor-arg value="expireAfterAccess=10s,maximumSize=25" />
        </bean>

        <bean id="cache" class="com.google.common.cache.Cache" factory-bean="cacheBuilder" factory-method="
build" />

        <bean class="net.shibboleth.idp.attribute.resolver.dc.rdbms.impl.FormatExecutableStatementBuilder">
                <constructor-arg index="0" value="SELECT * FROM people WHERE userid='%s'" />
        </bean>

        <bean id="mappings" class="net.shibboleth.idp.attribute.resolver.dc.rdbms.impl.
StringResultMappingStrategy"
        p:noResultAnError="true" p:multipleResultsAnError="true">
        <property name="resultRenamingMap">
            <map>
                <entry key="homephone" value="phonenumber" />
            </map>
        </property>
        </bean>
</beans>
```