Specifying a Deployment for Platform Types

Items within SAP Commerce are made persistent by writing values into a database. Within the database, the values are stored in tables. SAP Commerce allows you to explicitly define the database tables where the values of instances of a given type will be written. This process is referred to as deployment.

Technical Background

SAP Commerce stores instances of types in tables within a database. Every instance is stored as one row in a database table.

The table within a database where instances of a type are stored is called the type deployment and is specified with the type itself. Every type needs to have a deployment to store its instances. Deployment is inherited from a type to its subtypes. The deployment that is active for a given type is the deployment specified closest to the type in the type's hierarchy. The topmost deployment is **GenericItem**, which is therefore the default deployment. This means if a type has no explicit specification of deployment, that type's instances are deployed in the same table as **GenericItem**.

This means that the default deployment of any subtype, which you extend from **GenericItem**, is the deployment of **GenericItem**. In other words: if you do not specify a deployment for a subtype of **GenericItem**, the instances of that subtype are stored in the same table as instances of **GenericItem**.

For example, the Catalog and CronJob types in SAP Commerce are subtypes of **GenericItem**. If there were no deployment specified for Catalog and CronJob, all Catalog and CronJob instances are written into one single database table. Firstly, this is not intuitive. Secondly, storing instances of many different types in one single database table causes that database table to have quite a lot of columns to store all attributes from all these types (a CronJob has different attributes than a Catalog, and both types need to store attribute values).

Note

Specify Deployment for All Types

Deployment of a large number of types in a single table can markedly decrease the performance. Therefore, by default, builds fail if you do not specify the deployment table and you may encounter errors such as:

```
[ycheckdeployments] No deployment defined for relation <RELATIONNAME> in file: <FILENAME>
```

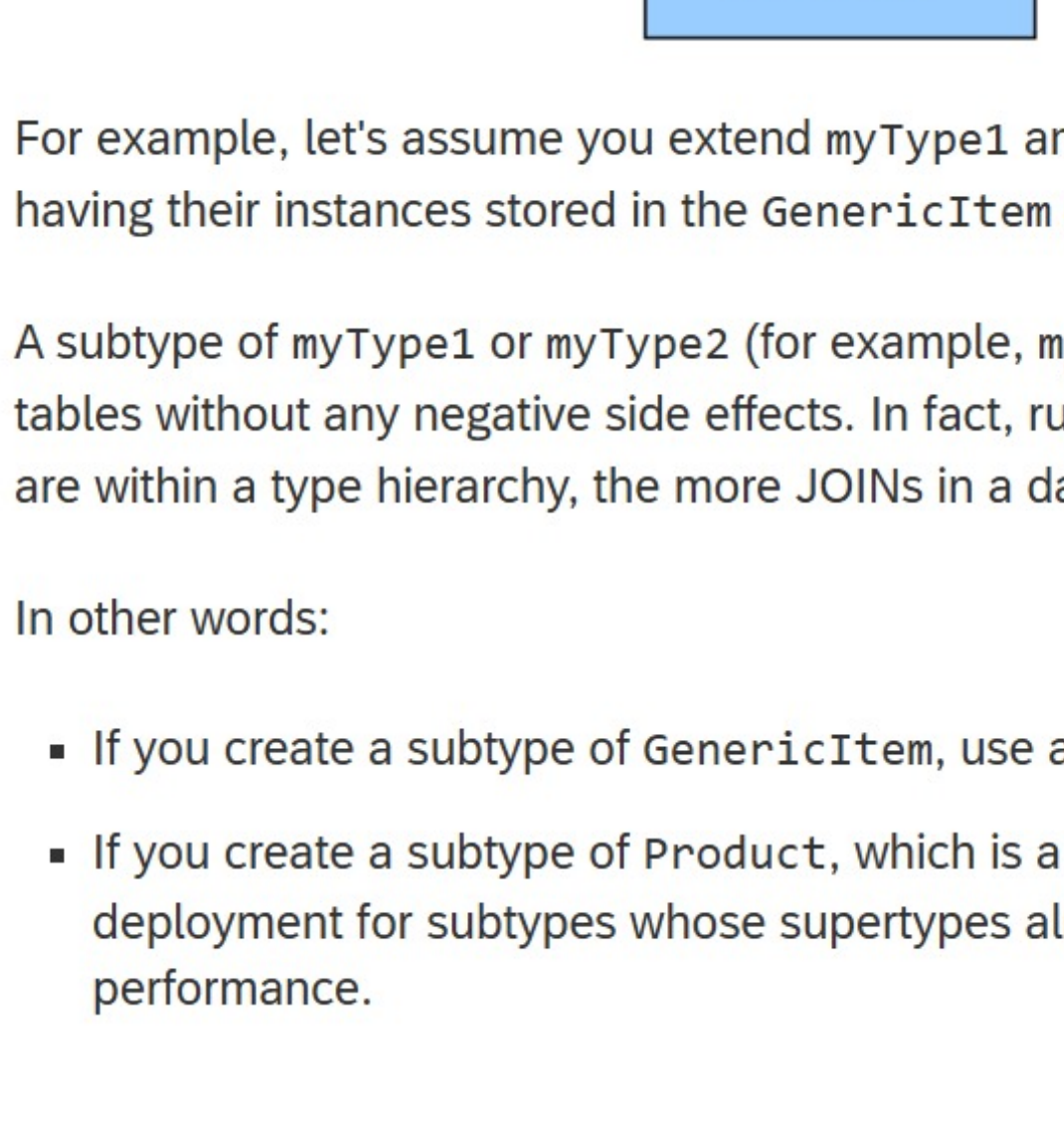
Otherwise it would be easy to forget to specify the deployment, and, as a result, some types would go to the **GenericItem** table. The build failure reminds you that deployment is not specified for some types. To change this default behaviour, set the property **build.development.mode** to **false** in the **`local.properties`** file:

```
build.development.mode=false
```

Setting this value to false is useful for some legacy projects that keep all items in default tables (genericitems, links).

Keeping individual types' instances deployed in different tables keeps the number of columns down to a minimum.

By consequence, SAP recommends specifying a deployment for direct subtypes of **GenericItem** only. Subtypes of subtypes of **GenericItem** usually do not need a specific deployment as the number of database table columns in the deployment is not likely to get out of hand. In constrast, specifying a deployment for a subtype whose supertype has a deployment already is likely to reduce database performance (especially during long and complex database transactions, such as synchronization between catalog versions).



For example, let's assume you extend `myType1` and `myType2` from `GenericItem`. Then it is recommended for `myType1` and for `myType2` to have a specific deployment (to avoid having their instances stored in the `GenericItem` database table).

A subtype of `myType1` or `myType2` (for example, `myType3` in the diagram) is not likely to also need a specific deployment. The instances of `myType3` fit into the `myType2` type's tables without any negative side effects. In fact, running FlexibleSearch statements on `myType2` requires JOINs to include the `myType3` type as well. The more deployments there are within a type hierarchy, the more JOINs in a database statement are necessary, and the longer complex database actions take to complete.

In other words:

- If you create a subtype of `GenericItem`, use a deployment.
- If you create a subtype of `Product`, which is a subtype of `GenericItem` already, using a specific deployment is discouraged by SAP. It is technically possible to use a deployment for subtypes whose supertypes already have an individual deployment, but it is not recommended. The JOINs required to construct database statements reduce performance.

Solution: Defining a Deployment in SAP Commerce

Note

Database Limitations Apply

This form of specifying a deployment is affected by database limitations, which you have to comply with. Therefore, Platform only allows a deployment string of 24 characters maximum.

For example, Oracle databases only allow a maximum of 30 characters overall for the `table` attribute value.

This 30-character limit includes table prefixes. So if you use a table prefix of `myDataBase` (for a total of 10 characters), you have only 20 characters left for the `table` attribute value.

To specify a deployment, add a `<deployment table="tablename" typecode="typecode_number"/>` tag to the type definition in the **`items.xml`** file:

1. Open the **`items.xml`** in your extension.
2. Locate the type definition where you want to specify a deployment, such as

```
<itemtype code="MyItem" extends="GenericItem">
   <attribute qualifier="myAttribute" ... >
   ...
</itemtype>
```

3. Add the `<deployment>` tag nested into the item definition. You need to specify a value for the `deployment` and the `typecode` attributes :

```
<deployment table="mytype_deployment" />
```

- The `deployment` attribute specifies the table name into which the instances of the type are written, such as `table="mytype_deployment"`
- The `typecode` attribute must specify a unique number to reference the type. The value of the `typecode` attribute must be a positive integer between 0 and 32767 (2^15-1) and must be unique throughout SAP Commerce as it is part of the PK generation mechanism. Typecode values between 0 and 10000 are reserved for SAP Commerce-internal use. Typecode values larger than 10000 are generally free for you to use but there are lots of exceptions to that rule.

  Note

  Not all typecode values larger than 10000 are free for you to use. There are many exceptions. Here are some examples:

  - **commons** extension (132xx)
  - **processing** extension (327xx)
  - Legacy **xprint** extension (244xx,245xx)
  - **b2bcommerce** extension (100xx)

  For a full list of exceptions, see the *`<HYBRIS_BIN_DIR>`*`/platform/ext/core/resources/core/unittest/reservedTypecodes.txt` file.

The entire type definition might look like this:

```
<item code="MyItem" extends="GenericItem">
   <deployment table="mytype_deployment" typecode="12345"/>
   <attribute name="myAttribute" ... >
   ...
</item>
```

Using a typecode that is already is use causes SAP Commerce to fail the build with the error message **due to duplicate deployment code**, as in:

```
[java] java.lang.IllegalArgumentException: cannot merge namespace ((customerreview)) into ((<merged>)) due to duplicate deployment code '1052' :
        de.hybris.platform.persistence.customerreview_CustomerReview::((customerreview))::YDeployment[customerreview.items.xml:51(ItemTypeTagListe
      de.hybris.platform.persistence.europe1_DiscountRow::((europe1))::YDeployment[europe1.items.xml:150(ItemTypeTagListener)]
```

Specifying or Changing Deployment for Relations

There are two important facts related to deployment for relations.

Firstly, you must specify a deployment for m:n relation, otherwise Platform will not build. Previously it was allowed to define an m:n relation which had no deployment. The relation was then maintained by the Links table. If there were multiple relations without a specified deployment, they all resided in the Links table. This caused bad performance and is not a good practice. Therefore, whenever you try to define an m:n relation without a deployment, initialisation fails with the following error:

```
[ycheckdeployments] No deployment defined for relation <RELATIONNAME> in file: <FILENAME>
```

Secondly, bear in mind that Platform does not allow changing an existing deployment. This is done in order to protect data. If there were already some records in current deployment of a type or relation, and the deployment would get changed afterwards, then access would be lost to all those records.

This has a special consequence in conjunction with the previous paragraph (specifying a deployment), because if an m:n relation did not have a deployment before, and now (according to best practices) someone wants to assign it a deployment, this change will not be performed.

Every time Platform refuses to make a deployment change, one of the following messages is printed:

- If there was no deployment before:

```
Addition of the deployment for type <TYPECODE> from <OLDDEPLOYMENT> to <NEWDEPLOYMENT> will not be performed; be aware that the old deployment
```

- If there was another deployment defined before:

```
Modification of the deployment for type <TYPECODE> from <OLDDEPLOYMENT> to <NEWDEPLOYMENT> will not be performed; be aware that the old deploym
```

Using a Custom Property Table

If you want to use an own propertytable, you have to specify an own `extensionname-advanced-deployment.xml` in the resource folder of your extension. To use the table **testtable**, you would need the content of this file like this:

```
<model name="hybris" description="...">
      <package name="de.hybris.jakarta.session" description="all session beans">
         <package name="de.hybris.jakarta.session.property">

            <object name="Property">
               <object-mapping>
                  <table name="testtable"/>
               <index name="ITEMPK">
                  <index-key attribute="itemPK"/>
               </index>
               <index name="NAMEIDX">
                  <index-key attribute="name"/>
               </index>
               </object-mapping>
               <attribute name="itemPK" type="HYBRIS.PK" primkey-field="true">
                  <attribute-mapping persistence-name="ITEMPK" null-allowed="false"/>
               </attribute>
               <attribute name="itemTypePK" type="HYBRIS.PK" >
                  <attribute-mapping persistence-name="ITEMTYPEPK" null-allowed="false"/>
               </attribute>
               <attribute name="name" type="java.lang.String" primkey-field="true">
                  <attribute-mapping persistence-name="NAME" null-allowed="false"/>
               </attribute>
               <attribute name="langPK" type="HYBRIS.PK" primkey-field="true">
                  <attribute-mapping  persistence-name="LANGPK" null-allowed="false"/>
               </attribute>
               <attribute name="realName" type="java.lang.String">
                  <attribute-mapping persistence-name="REALNAME"/>
               </attribute>
               <attribute name="type1" type="int">
                  <attribute-mapping persistence-name="TYPE1"/>
               </attribute>
               <attribute name="valueString1" type="HYBRIS.LONG_STRING">
                  <attribute-mapping database="sqlserver" persistence-name="VALUESTRING1" persistence-type="nvarchar(1800)"/>
                  <attribute-mapping database="oracle" persistence-name="VALUESTRING1" persistence-type="varchar2(4000)"/>
                  <attribute-mapping persistence-name="VALUESTRING1"/>
               </attribute>
               <attribute name="value1" type="java.io.Serializable">
                  <attribute-mapping persistence-name="VALUE1"/>
                  <attribute-mapping database="oracle" persistence-name="VALUE1" persistence-type="LONG RAW"/>
               </attribute>
            </object>

         </package>

</package>
</model>
```

SAP Commerce contains the **Maintenance** › **Deployment** page in Administration Console that gives an overview of the typecodes and the deployments in use.

Advanced Deployment

To see how SAP Commerce types are mapped to different databases, see the file **`bin/platform/ext/core/resources/core-advanced-deployment.xml`**.

```
    <database-schema database="hsqldb" primary-key="primary key" null="" not-null="not null" >
        <type-mapping type="java.lang.String" persistence-type="VARCHAR(255)" />
        <type-mapping type="String" persistence-type="VARCHAR(255)" />

        <type-mapping type="java.lang.Float" persistence-type="float"  />
        <type-mapping type="java.lang.Double" persistence-type="double"  />
        <type-mapping type="java.lang.Byte" persistence-type="smallint"  />
        <type-mapping type="java.lang.Character" persistence-type="smallint"  />
        <type-mapping type="java.lang.Short" persistence-type="smallint"  />
        <type-mapping type="java.lang.Boolean" persistence-type="tinyint"  />
        <type-mapping type="java.lang.Long" persistence-type="bigint"  />
        <type-mapping type="java.lang.Integer" persistence-type="int"  />

        <type-mapping type="float" persistence-type="float default 0"  />
        <type-mapping type="double" persistence-type="double default 0"  />
        <type-mapping type="byte" persistence-type="smallint default 0"  />
        <type-mapping type="char" persistence-type="smallint default 0"  />
        <type-mapping type="short" persistence-type="smallint default 0"  />
        <type-mapping type="boolean" persistence-type="tinyint default 0"  />
        <type-mapping type="long" persistence-type="bigint default 0"  />
        <type-mapping type="int" persistence-type="int default 0"  />

        <type-mapping type="java.util.Date" persistence-type="timestamp" />
        <type-mapping type="java.math.BigDecimal" persistence-type="DECIMAL(30,8)" />
        <type-mapping type="java.io.Serializable" persistence-type="longvarbinary" />

        <type-mapping type="HYBRIS.LONG_STRING" persistence-type="LONGVARCHAR" />
        <type-mapping type="HYBRIS.JSON" persistence-type="LONGVARCHAR" />
        <type-mapping type="HYBRIS.COMMA_SEPARATED_PKS" persistence-type="LONGVARCHAR" />
        <type-mapping type="HYBRIS.PK" persistence-type="BIGINT" />

    </database-schema>
```

Related Information

Third-Party Databases