The following table outlines other recommendations for defining your data model via *-items.xml:

Rule	Description	Included in TSV
A deployment table must be defined for all Items extending *GenericItem*	Failing to declare a deployment type means that all instances of this type will be stored in the genericitems in this table with other types with no deployment table. This table can become very large in terms of rows, columns as well as indexes. This can make the database work inefficiently causing performance issues . Fixing this issue in a production environment requires a data migration script. Example: `<itemtype code="mytype" extends="GenericItem" autocreate="true" generate="true"> <attributes> <!-- deployment should be defined here attributes --> ... </attributes> </itemtype>`	Yes

A deployment table must not be defined for any Items extending any item other than *GenericItem*	**A deployment table must not be defined for any Items extending any item other than *GenericItem***
	
	Defining a deployment table for an type that already has a deployment table inherited from a super type can cause serious performance issuesbecause queries for the super type require a *UNION* clause with the new table e.g.
	`SELECT {PK} FROM {Media} WHERE {code} = ?code UNION SELECT {PK} FROM {MyMedia} WHERE {code} = ?code`
	
	Fixing this issue in a production environment requires a data migration script.
	
	Example:
	`<itemtype code="MyMedia" extends="Media" autocreate="true" generate="true"> <deployment table="mymedia" typecode="22222"/> <!-- deployment should NOT be defined --> <attributes> ... </attributes> </itemtype>`
		Yes

Rule	Description	Included in TSV
Deployment type codes must be > 10000	Type codes less that 10000 are reserved for core Hybris Commerce types. There is a strong possibility that there is a conflict with a system type code if you use a value less than 10000. Bear in mind that even if there are no conflicts with the current version of Hybris Commerce and extensions being used, an upgrade to a newer version of Hybris Commerce could cause a type code confict. There are few exceptions even for code numbers above 10000, see Specifying a Deployment for hybris Platform TypesSpecifying a Deployment for hybris Platform Types for more details. `<itemtype code="MyCustomType" extends="GenericItem" autocreate="true" generate="true"> <deployment table="mycustomtype" typecode="1000"/> < deployment typecode should be > 10000 --> <attributes> ... </attributes> </itemtype>`	Yes
Jalo class must not be defined if an existing type is being extended	It is only possible to specify a Jalo class if we are sub-typing `<itemtype code="Product" autocreate="false" generate="false" jaloclass="com.hybris.product.jalo.MyProduct"> <!-- violation here, the type Product already has a jaloc ->  <attributes> ... </attributes> </itemtype>`	Yes
A deployment table must be defined for all many-to-many relations	This is very similar to the need having explicit deployment tables for items extending the GenericItem. Without an explicit deployment table, the relations will be persisted in the *links* table. The performance degradation happens as a result of too many rows, large indexes and table locking during inserts / updates / deletes. Typically,	Yes

Rule	Description	Included in TSV
	if the *links* table is one of the five largest tables in the system, this is the place to fix it.	
Attributes should not have a persistence type of 'cmp'	This is a deprecated persistence mode that should not be used anymore `<itemtype code="MyProduct" extends="Product" autocreate="true" generate="true"> <attributes> <attribute qualifier="myAttribute" type="java.lang.String"> <persistence type="cmp"/> <!-- should only property or dynamic --> </attribute> </attributes> </itemtype>`	Yes
Mandatory fields (where optional='false') must either have initial set to 'true' or a default value defined	This ensures data consistency on the Type System level. Also note, that if you set a default value in the items.xml file, the default value will be persisted for newly created items only, existing data will not be updated.	Yes
Immutable fields (where write='false') must have initial set to 'true'	Setting write="false" means that a value can only be set when we Item is first created. Setting initial="true" effectively achieves the same. For consistency we should set initial="true" when write="false"	Yes
Boolean attributes must be defined as mandatory	It is possible to store 3 values for a Boolean attribute (true, false or null). Store a null is almost always not the desired outcome and it will usually be mapped to a true or false value. It's possible that the mapping is not always in place so there is potential for logic errors or NullPointerExceptions	Yes
Attributes should not have a persistence type of 'jalo', use 'dynamic' instead	JALO is a deprecated API and should not be used. Previously JALO attributes were used when we need some business logic to be executed before returning a result. This behaviour can now be achieved with Dynamic Attributes in the Service Layer without needing to touch JALO `<itemtype code="MyProduct" extends="Product" autocreate="true" generate="true">`	Yes

Rule	Description	Included in TSV
	<attributes> <attribute qualifier="myAttribute" type="java.lang.String"> <persistence type="jalo"/> <!-- should only use property or dynamic --> </attribute> </attributes> </itemtype>	
Type Names (including EnumTypes and Relations) must start with an uppercase letter	For consistency all type names should start with an uppercase letter. You may run into issues with the code generators (e.g. platformwebservices) if you do not follow this convention. <itemtype code="myProduct" extends="Product" autocreate="true" generate="true"> <!-- violation he "myProduct" should be "MyProduct" <attributes> .. </attributes> </itemtype>	Yes
Type Names must not start with the string Generated	Starting a type name with "Generated" can have some side effects because a abstract class that starts with "Generated" is created by Hybris Commerce already <itemtype code="GeneratedProduct" extends="Product" autocreate="true" generate="true"> <!-- violation he "GeneratedProduct" will cause some side effects --> <attributes> .. </attributes> </itemtype>	Yes
Item attribute names must start with a lowercase letter	For consistency all item type attribute names should start with an lowercase letter. You may run into issues with the code generators (e.g. platformwebservices) if you do not follow this convention.	Yes
Relationship qualifier names must start with a lowercase letter	For consistency all relation type attribute names should start with a lowercase letter. You may run into issues with the code generators (e.g. platformwebservices) if you do not follow this convention.	Yes

Rule	Description	Included in TSV
Any side of a relation that has cardinality='many' should not have ordered='true' unless absolutely necessary	Order="true" has a significant impact on performance when reading and writing from the database. The queries required to provide a defined order to the items that are returned are much more complex and expensive that an unordered query. Using ordered="true" should only be used when absolutely necessary.	Yes
Any side of a relation that has cardinality='many' should have collectiontype='set' unless absolutely necessary	The set guarantees that every relation is persisted only once. For example if a Product is related to a Categoryit is usually desirable to store this relation as one row in the database.	Yes

CatalogVersion attribute should be marked unique for Catalog aware types	The CatalogVersion attribute forms part of the unique key for Catalog Synchronization. To ensure that the same uniqueness rules are applied for other processes (e.g. ImpEx) we need to set the CatalogVersion attribute to unique="true"
	`<itemtype code="MyCatalogType" extends="GenericItem"`
	`autocreate="true" generate="true">`
	`<deployment table="MyCatalogType"`
	`typecode="XXXX"/>`
	`<custom-properties>`
	`<property name="catalogItemType">`
	`<value>java.lang.Boolean.TRUE</va`
	`</property>`
	`<property`
	`name="catalogVersionAttributeQualifier">`
	`<value>"catalogVersion"</value>`
	`</property>`
	`<property`
	`name="uniqueKeyAttributeQualifier">`
	`<value>"uid"</value>`
	`</property>`
	`</custom-properties>`
	`<attributes>`
	`<attribute qualifier="uid" generate=`
	`autocreate="true" type="java.lang.String">`
	`<persistence type="property" />`
		No

Rule	Description	Included in TSV
	<modifiers optional="false" unique="true" /> </attribute> <attribute qualifier="catalogVersion" type="CatalogVersion"> <modifiers optional="false" /> <!-- violation here because missing unique="true" --> <persistence type="property" /> </attribute> </attributes> </itemtype>	
Unique attributes should match the catalog unique attribute key (uniqueKeyAttributeQualifier)	The unique attributes defined for the type and the attributes defined in "uniqueKeyAttributeQualifier" should have the same attributes. "uniqueKeyAttributeQualifier" defines the unique attributes for Catalog Synchronization whereas processes using the Service Layer (including the Catalog Sychronization persistence) use the unique attributes on the type. <itemtype code="MyCatalogType" extends="GenericItem" autocreate="true" generate="true"> <deployment table="MyCatalogType" typecode="XXXX"/> <custom-properties> <property name="catalogItemType"> <value>java.lang.Boolean.TRUE</v </property> <property name="catalogVersionAttributeQualifier"> <value>"catalogVersion"</value> </property> <property name="uniqueKeyAttributeQualifier"> <value>"uid"</value> </property> </custom-properties> <attributes> <attribute qualifier="uid" generate= autocreate="true" type="java.lang.String"> <persistence type="property" /> <modifiers optional="false" unique="false" /> <!-- violation here because missin unique="true" -->	No

Rule	Description	Included in TSV
	</attribute> <attribute qualifier="catalogVersion" type="CatalogVersion"> <modifiers optional="false" unique="true" /> <persistence type="property" /> </attribute> </attributes> </itemtype>	
Database indexes should be defined for the unique attributes of type	There should be a covering index defined that includes all the unique attributes for type. There are Service Layer interceptors that validate uniqueness for each type, which generate a query for all the unique attributes. Failure to add an index can cause serious performance issues. Furthermore, the validation of the unique attributes in the Service Layer cannot guarantee that there won't be duplicate records in the database, only a unique index/constraint can do this. <itemtype code="MyType" extends="GenericItem" autocreate="true" generate="true"> <deployment table="MyType" typecode="XXX <attributes> <attribute qualifier="uid" generate= autocreate="true" type="java.lang.String"> <persistence type="property" /> <modifiers optional="false" unique="true" /> </attribute> </attributes> <indexes> <!-- violation here because missing inde uid --> </indexes> </itemtype>	No
Catalog aware types must not be part of a one-to-many relation when the many side is not a Catalog aware type	Creating a one-to-many relation where the one side of the relation is a Catalog aware type and the many side is not a Catalog aware type causes an issue when synchronizing. The reference to the Catalog aware type will be transferred from the Staged to the Online version. The Staged version will not longer have a reference.	No

Rule	Description	Included in TSV
	<relation code="Product2Author" generate="true" localized="false" autocreate="true"> <sourceElement type="Product" cardinality="one" qualifier="product"/> <!-- violati here, MyItem is not a catalog aware type and it can reference a single Product (not a Staged and Online version --> <targetElement type="MyItem" cardinality="many" qualifier="myItems" ordered="true </relation>	
All types should have unique attributes defined	Every type should have a unique identifier(s) defined or should inherit the unique attributes from a super type. This allows the data to be easily imported/exported into different systems and prevents duplicate entries in the database. <itemtype code="MyType" extends="GenericItem" autocreate="true" generate="true"> <deployment table="MyType" typecode="XXX <attributes> <!-- missing a unique attri -> <attribute qualifier="name" generate="true" autocreate="true" type="java.lang.St <persistence type="property" /> <modifiers optional="false" unique="false" /> </attribute> </attributes> </itemtype>	No
Unoptimized attributes should not be used	Using dontOptimize="true" stores the data for this type in a secondary table defined by the "propertytable" attribute of the deployment or the props table if no propertytable is defined. Generally it is not recommended to store attribute values in this way because it is more expensive to store and retrieve the values because joins are required. <itemtype code="MyType" extends="GenericItem" autocreate="true" generate="true"> <deployment table="MyType" typecode="XXX propertytable="MyTypeProps"/>	No

Rule	Description	Included in TSV
	<attributes> <attribute qualifier="name" generate="true" autocreate="true" type="java.lang.String"> <persistence type="property" /> <modifiers optional="false" unique="false" dontOptimize="true"/> <!-- defining dontOptimize stores this value in the property table for this type (MyTypeProps) of the props table if none is defined --> </attribute> </attributes> </itemtype>	

Extending Existing Type vs Sub-Type

When extending an existing type to add new attributes, we can either add the attributes to the existing type

e.g.

```
<itemtype code="Product" autocreate="false" generate="false">
<attributes>
     <attribute qualifier="newAttribute"
type="java.lang.String">
     <modifiers read="true" write="true" search="true"
optional="true"/>
      <persistence type="property"/>
</attribute>
</itemtype>
```

Or we can sub-type e.g.

```
<itemtype code="MyProduct" extends="Product" autocreate="true"
generate="true">
<attributes>
     <attribute qualifier="newAttribute"
type="java.lang.String">
     <modifiers read="true" write="true" search="true"
optional="true"/>
      <persistence type="property"/>
</attribute>
</itemtype>
```

Whether to sub-type or extend the existing type should be based on the following decisions:

Change	Existing or Sub-Type	Reason
Adding an attribute for all instances	Existing	If all instances of the existing type could have a value for this data then the attribute belongs on existing type. Creating an unnecessary sub-type can create issues because a business user could create an instance of the super type that doesn't have all the required/expected attributes e.g. a user could create instances of items that are not compatible with the business logic For example, if we sub-type Category and create new type with a mandatory attribute called "enabledForFrontend". Business users can still create a standard Category item that doesn't have the required attribute that the front end is expecting. In addition the Hybris Commerce APIs expect the existing type so the project code will be cluttered by "if instanceof then (cast)" constructs if a subtype is used.
Adding an attribute for specific instances	Sub-Type	If only some instances have data for a particular attribute then we should sub-type.
Re-declaring an existing attribute e.g. change mandatory to optional	Sub-Type	Re-declaring requires sub-typing