Data Modeling

SAP Commerce Cloud Developer Training

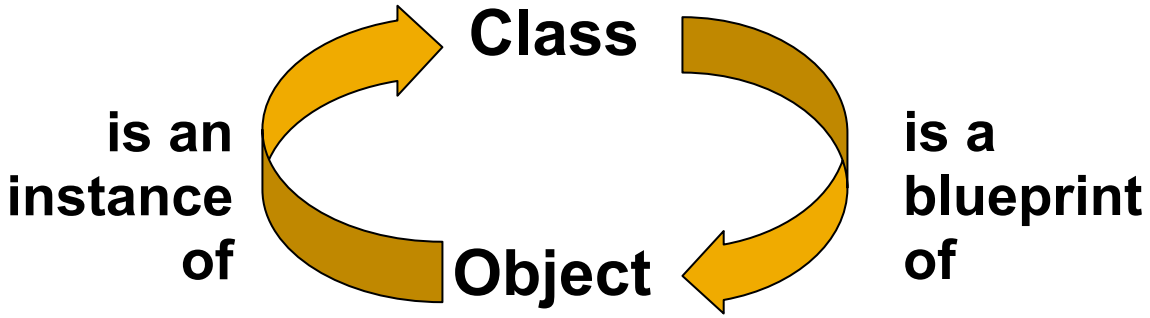
Introduction to the Type System

SAP Commerce and Java

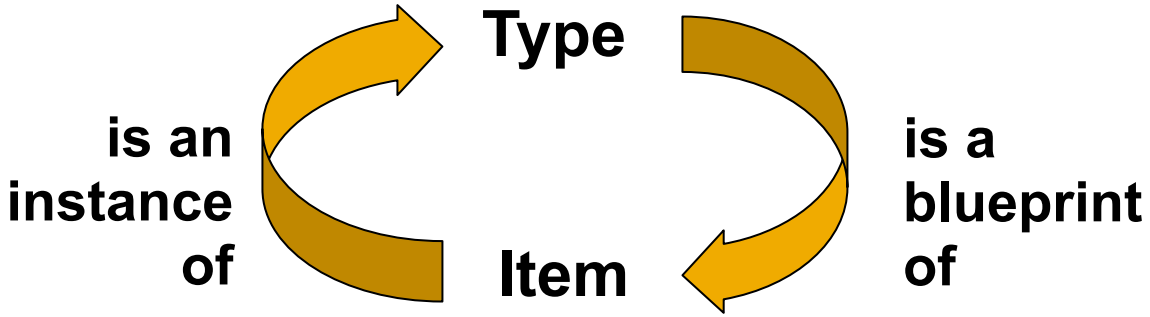
? Who is responsible for converting SAP Commerce type definitions to Java classes?

The SAP Commerce build system (ant)

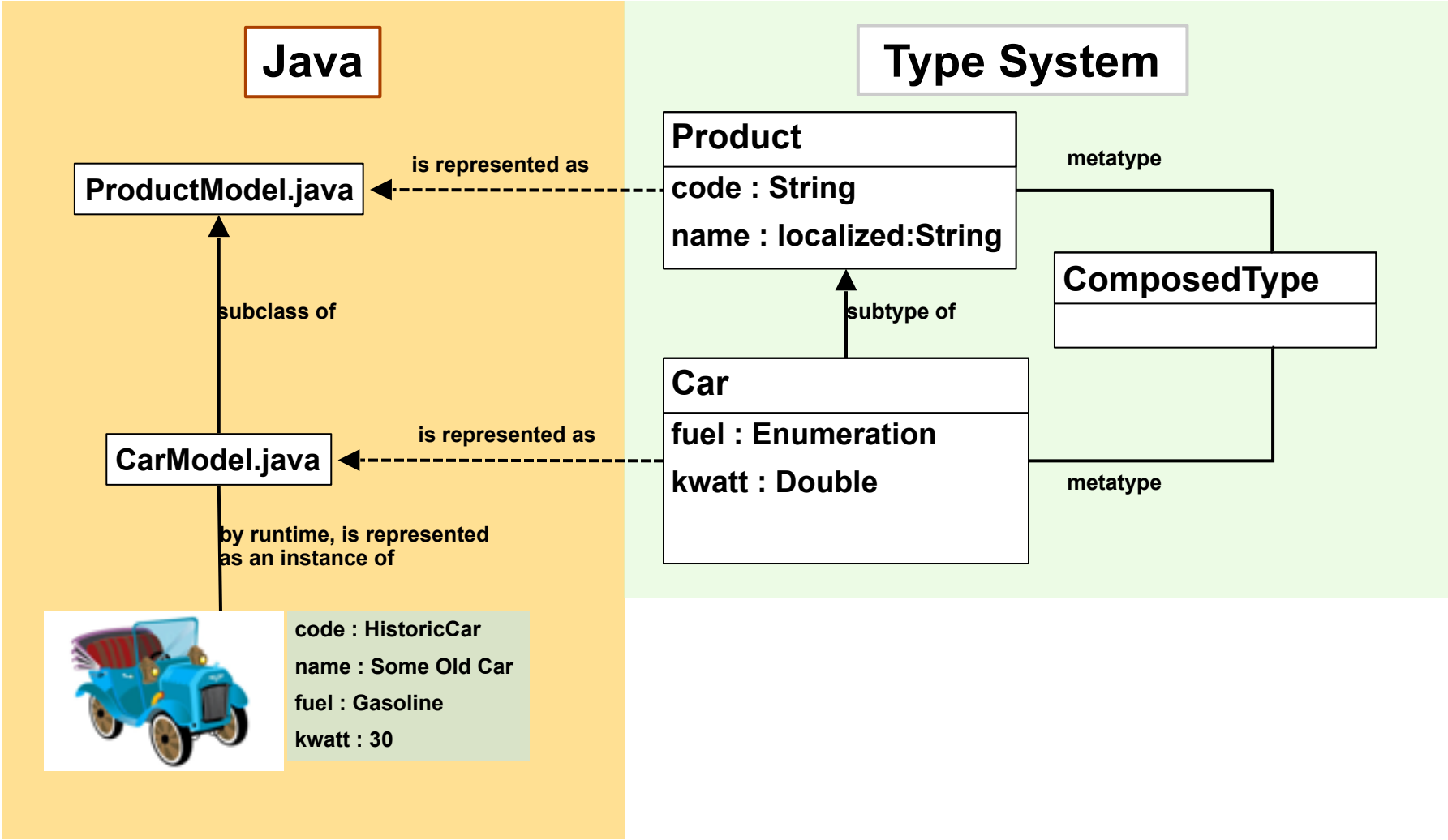
Java



SAP Commerce



Java Classes vs SAP Commerce Types



Types used in SAP Commerce

AtomicType

- Represents Java value objects which are mapped to database types
  - Java Primitive keywords: `int`
  - Java Wrapper Classes: java.lang.`Integer, java.math.BigInteger`
  - Some Reference types: `java.util.Date, java.lang.String, de.hybris.platform.core.PK`

CollectionType

- Represents a typed collection

EnumerationType

- ComposedType which describes enumerations

Types used in SAP Commerce

MapType

- Represents a typed Map

RelationType

- Used to model binary dependencies between items, representing n:m relations.

ItemType (aka ComposedType)

- Record attribute and relation meta data for each type, including unique identifier, db table, and supporting Java class. The foundation of the Commerce Suite's type system.

The sections within *extensionName*-items.xml (in order)

```
<items>

    <atomictypes>  ...     </atomictypes>

    <collectiontypes> ...  </collectiontypes>

    <enumtypes>    ...     </enumtypes>

    <maptypes>     ...     </maptypes>

    <relations>    ...     </relations>

    <!-- Composed Types -->
    <itemtypes>    ...     </itemtypes>

</items>
```

Extending the Data Model

- Create new types:
  - Define a type by extending already existing types, such as:

```
<itemtype code="Car" extends="Product">
```

  - Define “completely new types”, such as:

```
<itemtype code="Car">
```
(implicitly extends from GenericItem)

- Extend existing types:
  - Add attribute definitions to existing types (attribute injection), such as:

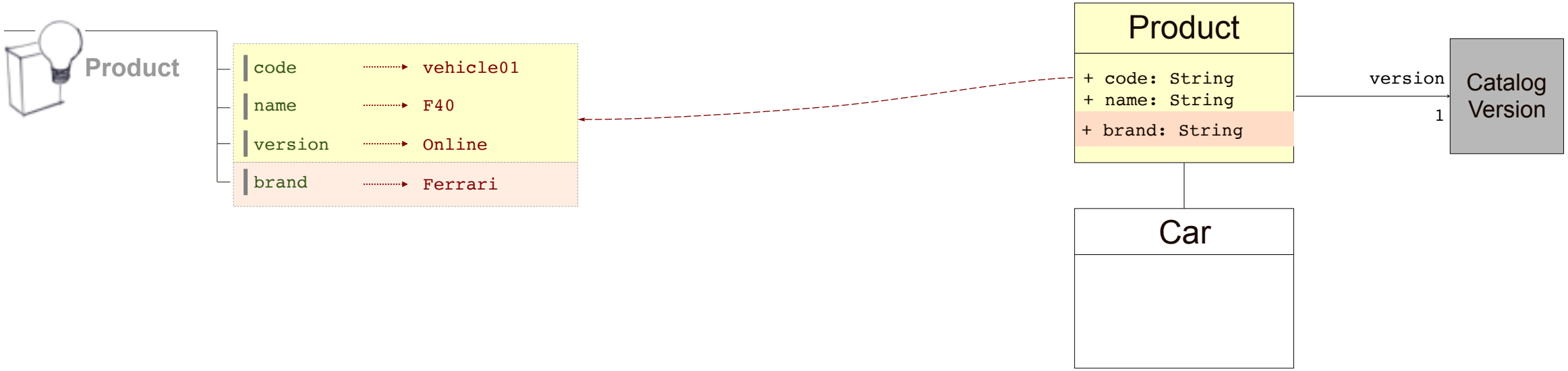
```
<itemtype code="Product" ...>
   ...
   <attributes>
      ...
      <attribute qualifier="myAttribute">
      ...
   </attributes>
<itemtype>
```

  - Redefine inherited attribute definitions from super type

```
<attribute qualifier="code" redeclare="true">
```

    - If you change the attribute's java type, the new type must extend the original type

Extending the Data Model • Extend Existing Type



- Add attribute definitions to existing types (attribute injection)

```
<items>
  <itemtypes>
    <itemtype code="Product" autocreate="false"
generate="false">
        <attributes>
            <attribute qualifier="Brand"
type="java.lang.String" />
        </attributes>
        ...
```

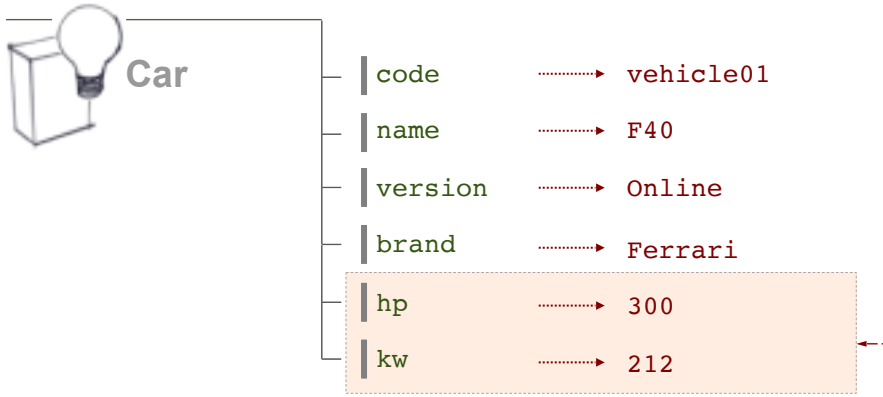
autocreate

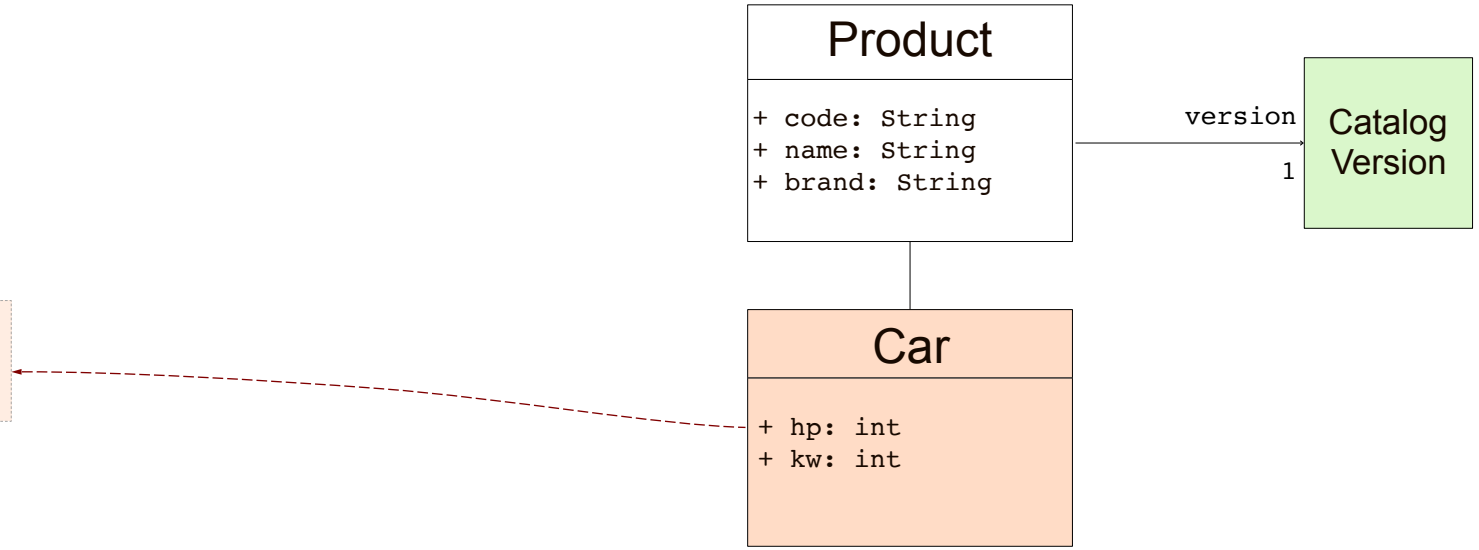
If set to **true**, indicates this is the first mention of this ItemType, which should be created during initialization. As *Product* is already defined in Commerce, set to **false**.

generate

If set to **true**, indicates this is the first mention of this ItemType, for which a model must be created. As *Product* is already defined in Commerce, set to **false**.

Extending the Data Model • New Type



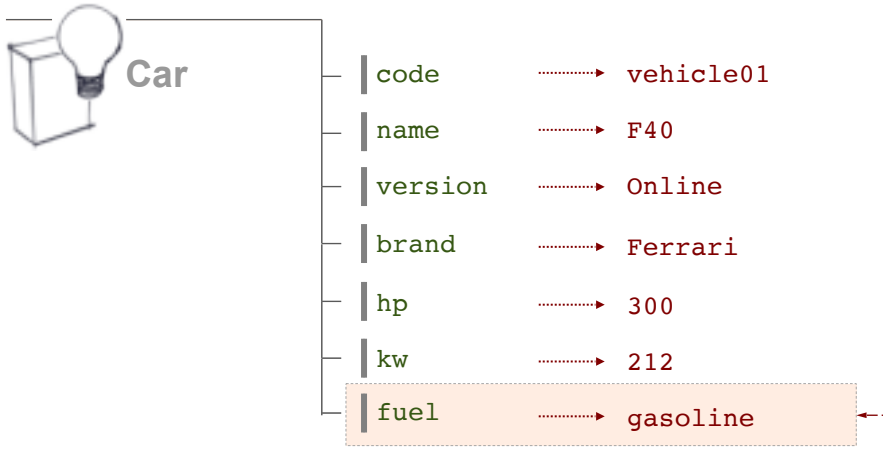


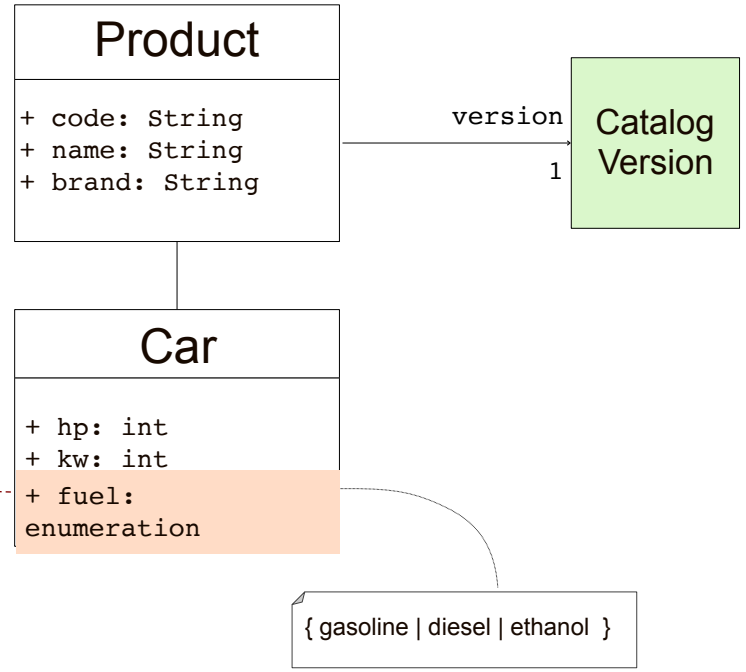

```
<items>
  <itemtypes>
    <itemtype code="Car" extends="Product" autocreate="true" generate="true">
      <attributes>
        <attribute qualifier="hp" type="java.lang.Integer">
          <description>Horsepower</description>
          <persistence type="property"/>
        </attribute>
        <attribute qualifier="kw" type="java.lang.Integer">
          <description>Kilowatt</description>
          <persistence type="dynamic"
                       attributeHandler="kwPowerAttributeHandler"/>
          <modifiers write="false" />
        </attribute>
        ...
```

? What is the difference between *property* and *dynamic*?

? In what conditions will *dynamic* be used?

Extending the Data Model • Enumerated Types

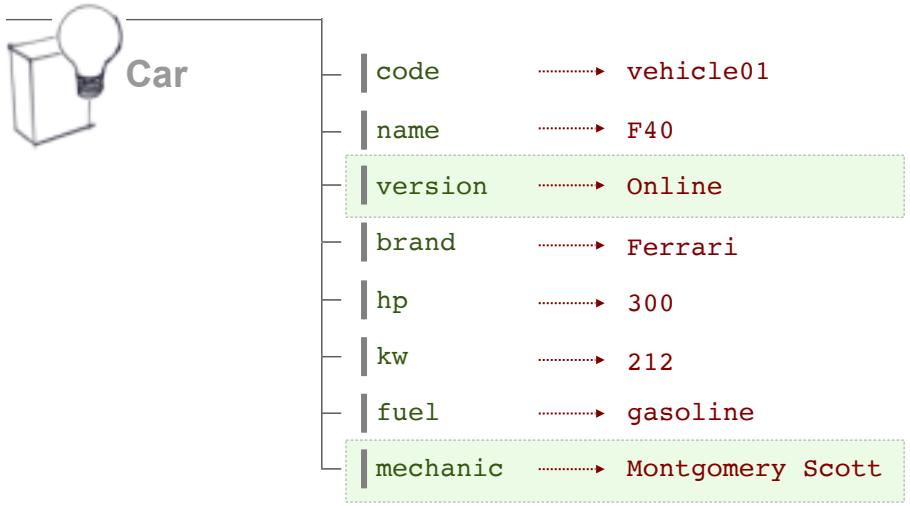


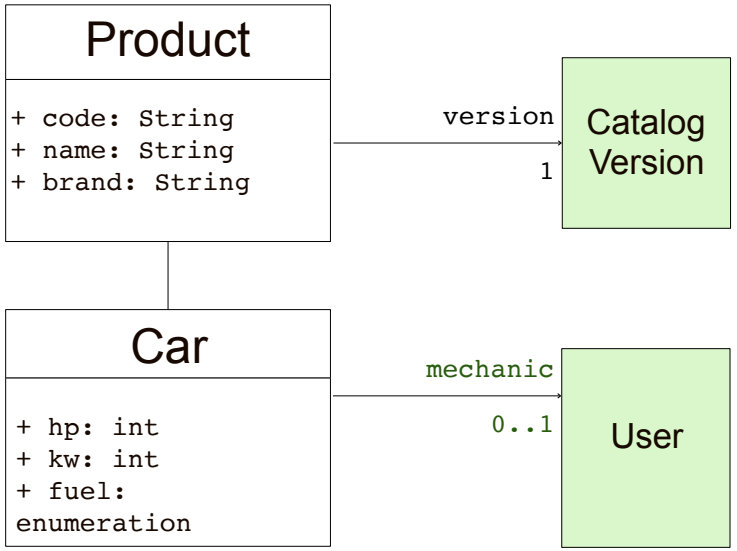



```
<items>
  <enumtypes>
    <enumtype code="FuelEnumeration" generate="true" autocreate="true"
              dynamic="true">
      <value code="diesel"></value>
      <value code="gasoline"></value>
      <value code="ethanol"></value>
    </enumtype>
  </enumtypes>

  <itemtypes>
    <itemtype code="Car" extends="Product" autocreate="true" generate="true">
      <attributes>
        ...
        <attribute qualifier="fuel" type="FuelEnumeration">
          <persistence type="property"></persistence>
        </attribute>
```

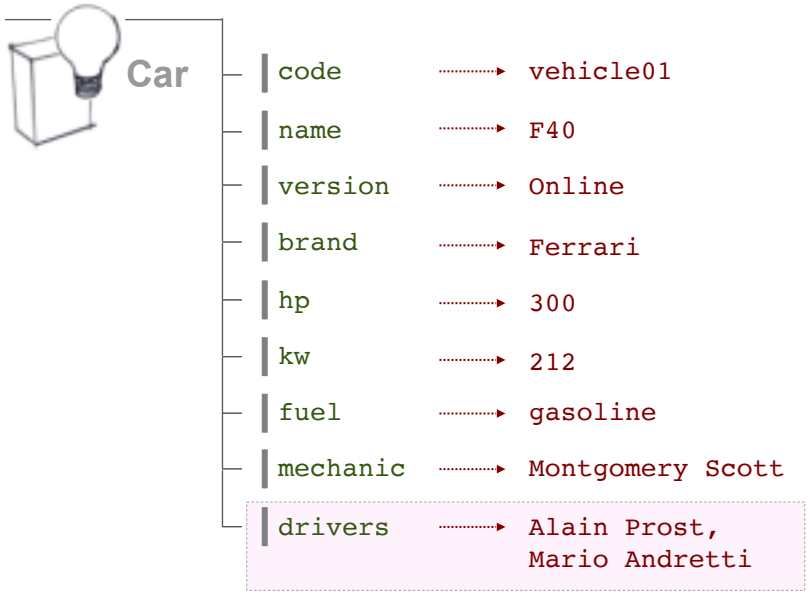
Extending the Data Model • Composed Type References

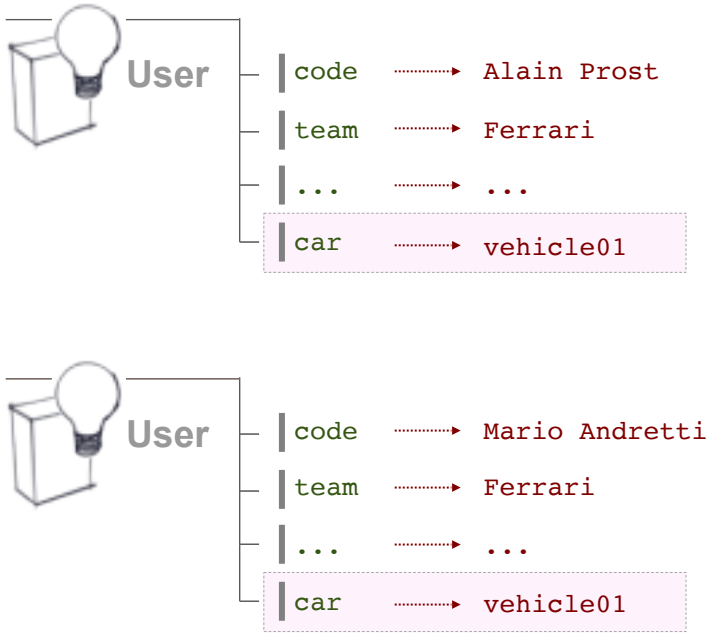


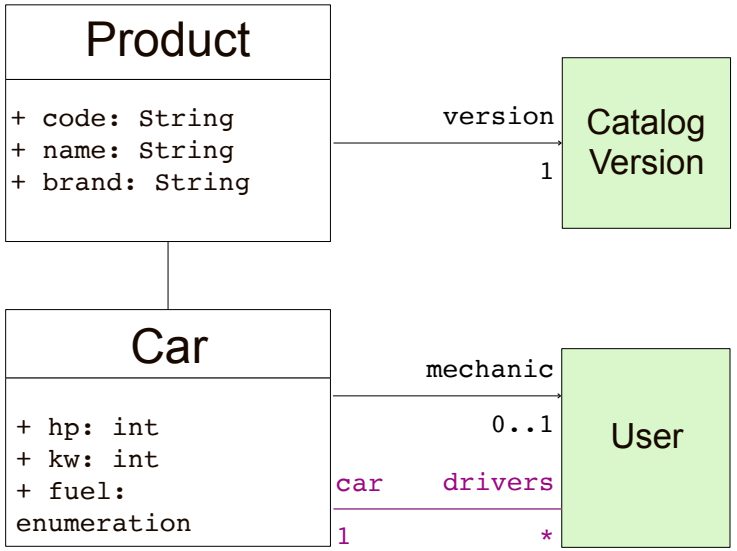



```
<items>
  <itemtypes>
    <itemtype code="Car" extends="Product" autocreate="true" generate="true">
      <attributes>
        ...
        <attribute qualifier="mechanic" type="Employee">
          <modifiers read="true" write="true" search="true" />
          <persistence type="property" />
        </attribute>
        ...
```

Extending the Data Model • Relations








```
<items>
  ...
  <relations>
    <relation code="Car2EmployeeRelation"
              localized="false" autocreate="true" generate="true">
      <sourceElement qualifier="car" type="Car" cardinality="one" />
      <targetElement qualifier="drivers" type="Employee" cardinality="many" />
    </relation>
  </relations>

 <itemtypes>
   ...
```

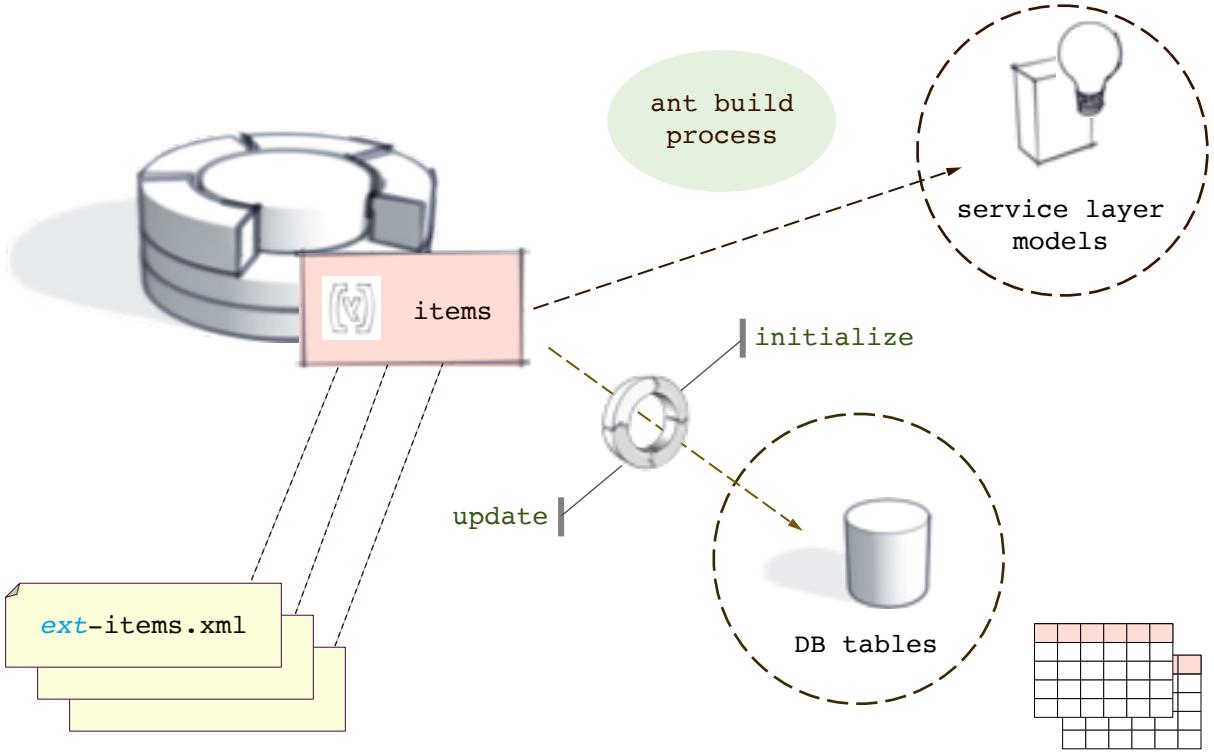
This example is **1:many**, but **many:many** relations are also supported

Commerce Type System • Automatic Generation

1. SAP Commerce item definitions are found in each extension's *extensionName*-items.xml

2. The ant process assembles type definitions and generates Models

3. Invoking initialize or update creates/modifies the required table.



Collections and Relations

Collection types

- Collection of target type
- Can be used as an attribute type of a ComposedType
- Allows you also to define AtomicType collections
- Performance considerations: Accessing, Searching
- Database integrity considerations

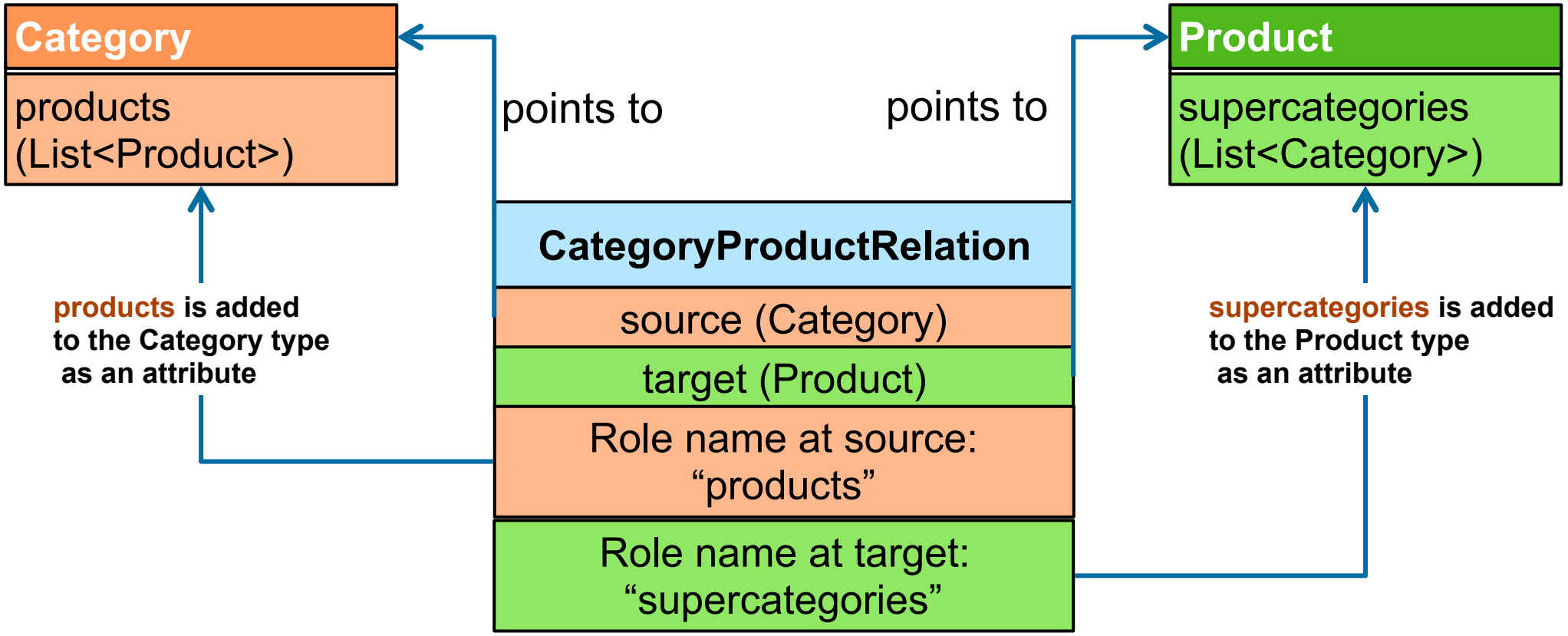
```
<collectiontype code="StringCollection"
                elementtype="java.lang.String" autocreate="true" />
<collectiontype code="LanguageCollection"
                 elementtype="Language" autocreate="true"/>

...

<itemtype code="..." ...
      <attribute qualifier="urlPatterns" type="StringCollection">
      <attribute qualifier="writeableLanguages" type="LanguageCollection">
      ...
```

Relations

- One2Many and Many2Many
- Both sides are (can be) aware of the other



What's so Important About Relations?

If in doubt: Use `relations`, not `CollectionTypes`, because:

- Opposite side is not "aware" of the `CollectionType`
- `CollectionTypes` are stored in a database field as a comma-separated list of references (PKs) or atomic values
- Can cause overflow
- More difficult to search and generally lower performance

Deployment

Questions

? What is the main incompatibility between object-oriented programming and relational databases?

Representing inheritance relationships

? How can we represent inheritance in a relational database

Usually, using one of two strategies:

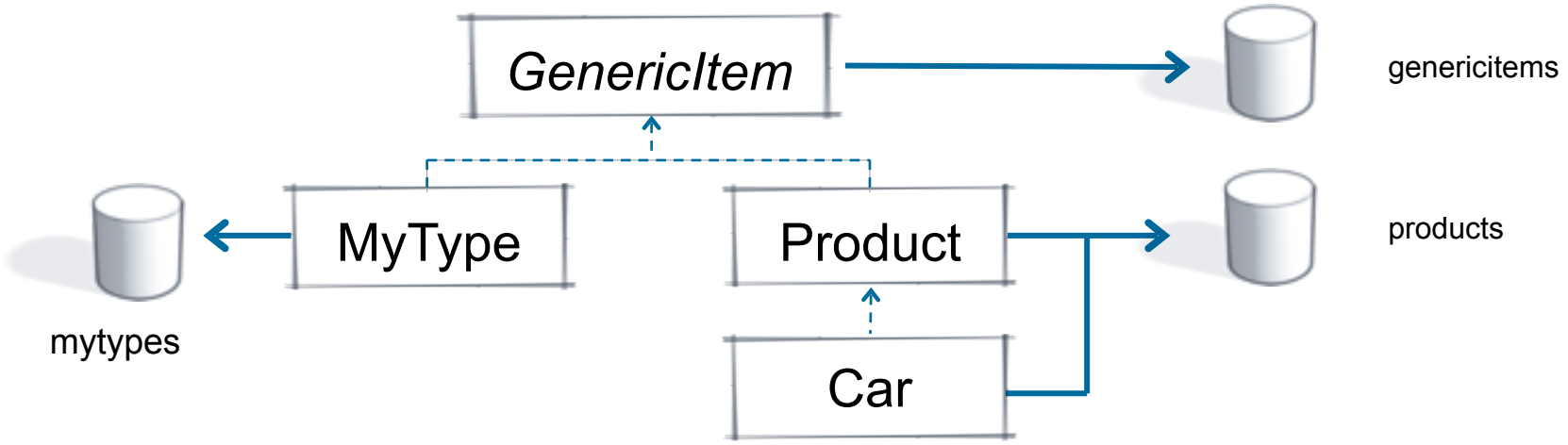
1. Subtype shares the supertype's table
2. Use different tables for supertype and subtype

? What are the advantages and drawbacks of these two strategies?

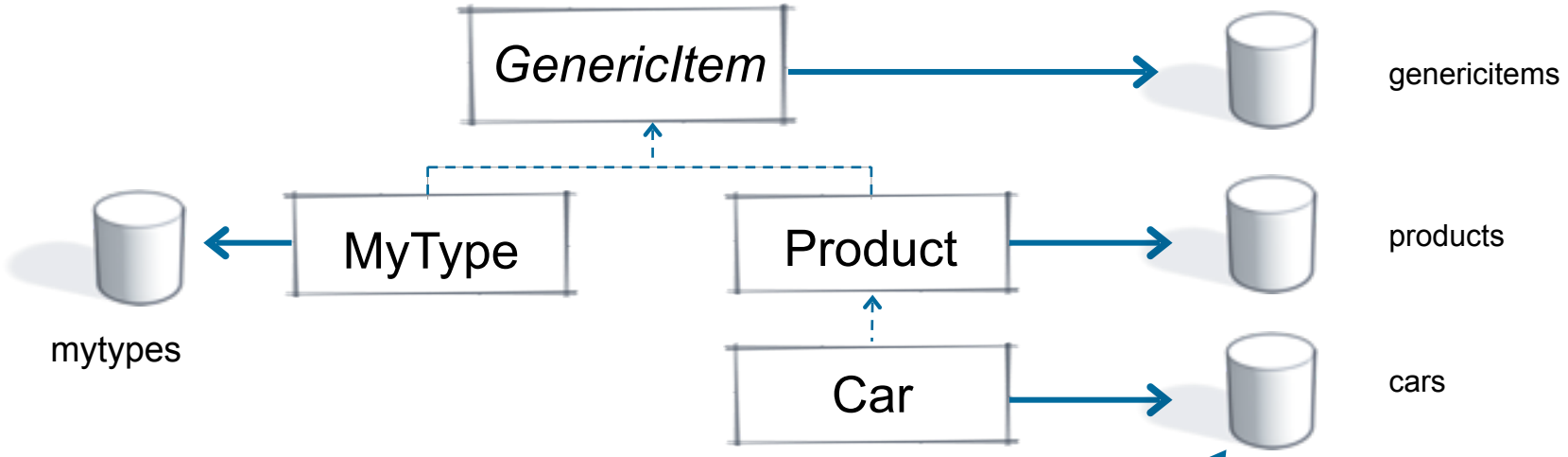
- Single-table strategy: better query performance but low storage efficiency
- Multiple-table strategy: worse query performance but high storage efficiency

Object Relational Mapping • Storing objects in the DB

- By default, items for a given type are stored in the same database tables as its supertype
- Specify any item type's *deployment* to store its items in its own db tables.
- SAP Commerce recommends that deployment be specified for the first layer of `GenericItem` subtypes
  - Consider carefully the performance implications of specifying deployment for other item types
  - Set `build.development.mode = true` in `local.properties` to mandate that all direct children of GenericItem have deployment specified.



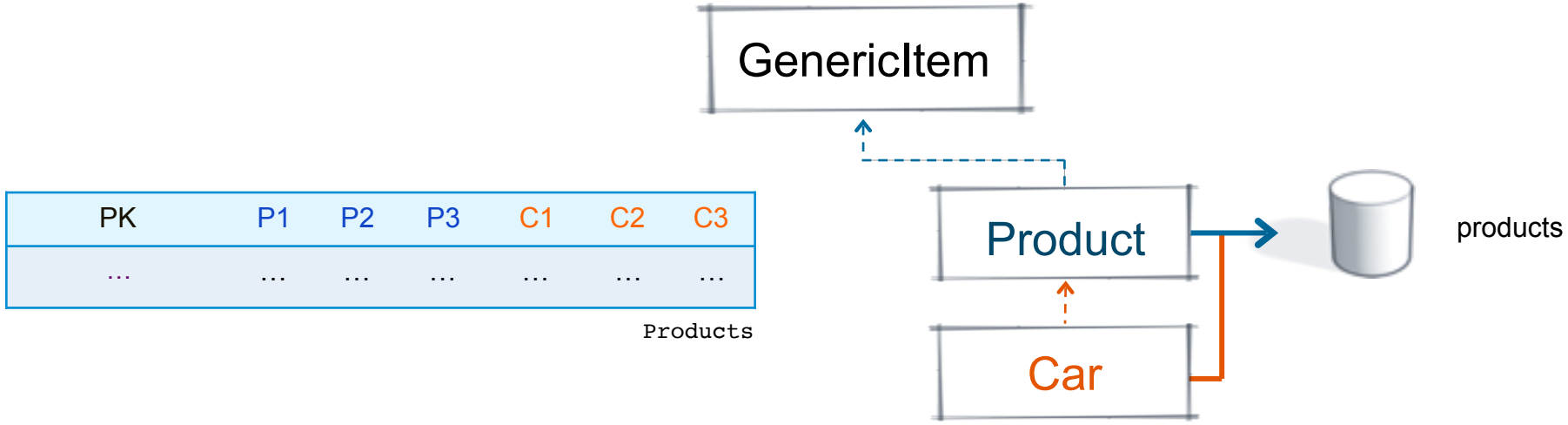
O-R Mapping • Deployment Example

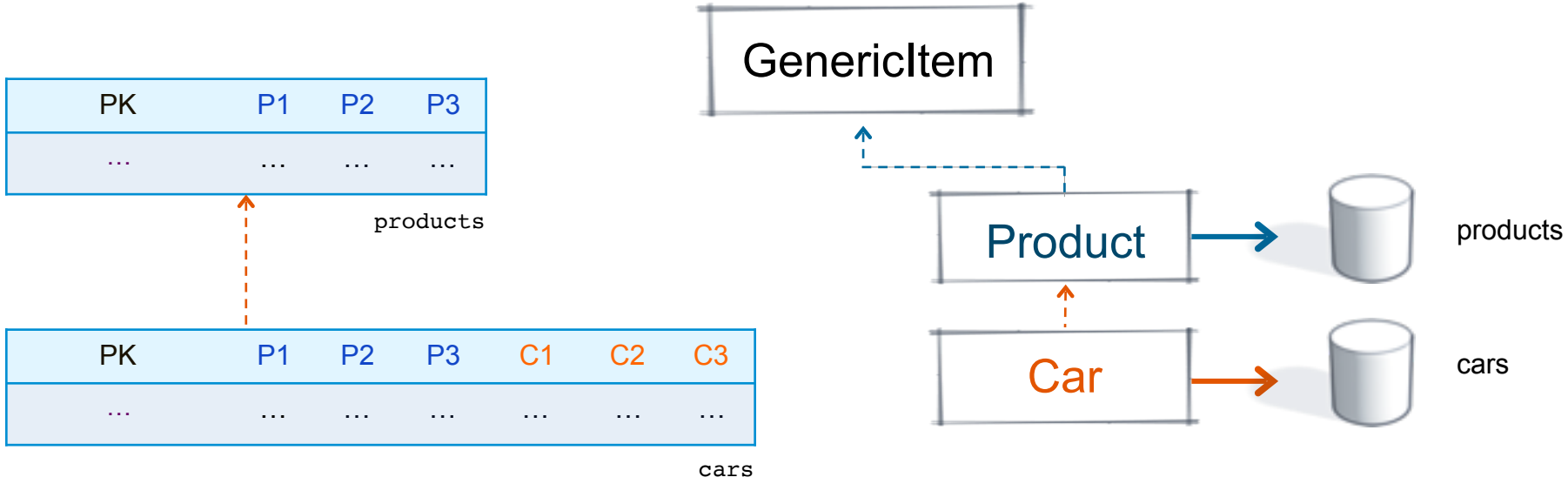



```
<itemtype code="Car" ...
    <deployment table="cars" typecode="20100" ...
```

0 .. 10000	are reserved by SAP Commerce
10000 .. 32767	are free for use, with *some exceptions*

O-R Mapping • Table Structure





O-R Mapping • Attributes of a (Composed) Type

PK	code	name	…	mechanic
8796256894977	vehicle01	Ferrari F40	…	8796128083972

products

Atomic type property – value stored directly in db. For instance, property of type String is stored as VARCHAR

Composed type property – reference to another item. The DB column stores a PrimaryKey (PK) value.

code	vehicle01
name	Ferrari F40
mechanic	Nikola Tesla

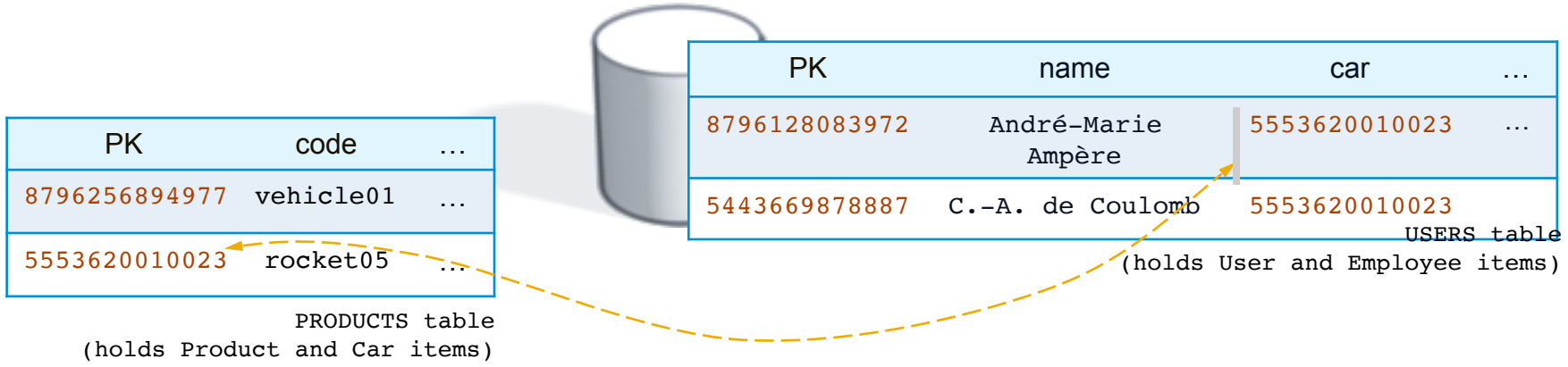
PK	name	…
8796128083972	Nikola Tesla	…

users

O-R Mapping • Deployment of Relations • 1

- **One-to-Many**
  - Additional column at the many side which holds the PK of the One side
  - Users table from the example below would have an additional column car
    - As with Car, Employee type does not have its own deployment, so its items live in the parent type's table *Users*

```
<relation code="Car2DriversRelation" generate="true" autocreate="true"
          localized="false" >
    <sourceElement qualifier="car" type="Car" cardinality="one" />
    <targetElement qualifier="drivers" type="Employee" cardinality="many"/>
</relation>
```



O-R Mapping • Deployment of Relations • 2

A deployment **must** be specified for many-to-many relationships

- **Many-to-Many**
  - New database table which holds the source and target PKs

```
<relation code="Product2ReviewerRelation" autocreate="true" generate="true"
          localized="false">
    <deployment table="Prod2ReviewerRel" typecode="20123"/>
    <sourceElement qualifier="reviewers" type="Employee" cardinality="many" >
          <modifiers read="true" write="true" search="true" optional="true" /
>
    </sourceElement>
    <targetElement qualifier="products" type="Product" cardinality="many" >
          <modifiers read="true" write="true" search="true" optional="true" /
>
    </targetElement>
</relation>
```

Relation "reviewers"

PK	uid	...
3776876789221	ajfoyt	...
6152677365115	mandretti	...

USERS table
(holds User and Employee items)

source	target
3776876789221	8796256894977
3776876789221	5553620010023
6152677365115	5553620010023

Prod2ReviewerRel

Relation "products"

PK	code	...
8796256894977	vehicle01	...
5553620010023	rocket05	...

PRODUCTS table
(holds Product and Car items)

O-R Mapping • Deployment of Collections

- **Collections**
  - Stored in one database column
  - Comma separated list of PKs or Atomic Values

```
<collectiontype code="StringCollection"
                elementtype="java.lang.String" autocreate="true" />
<collectiontype code="LanguageCollection"
                elementtype="Language" autocreate="true" />

...

<itemtype code="..." ...
      <attribute qualifier="urlPatterns" type="StringCollection" />
      <attribute qualifier="writeableLanguages"
type="LanguageCollection" />
      ...
```

PK	code	urlpatterns	writeableLanguages	…
8796256894977	Example1	http://ex1.com/a,http://ex2.com/b,http://ex3.com/c	93938293,93029304,01920394	…

products

Type System Localization

Two Shades of Localization

- Service layer provides support for i18n
- Leverage that to localize:
  - Types *names* and their attribute *names* ← Backoffice employees in different countries
  - Your data *values* (Itemtype properties) ← Front-end customers around the globe

? So, who needs what?

Type System Localization • Type-Name and Attribute-Name Display Labels

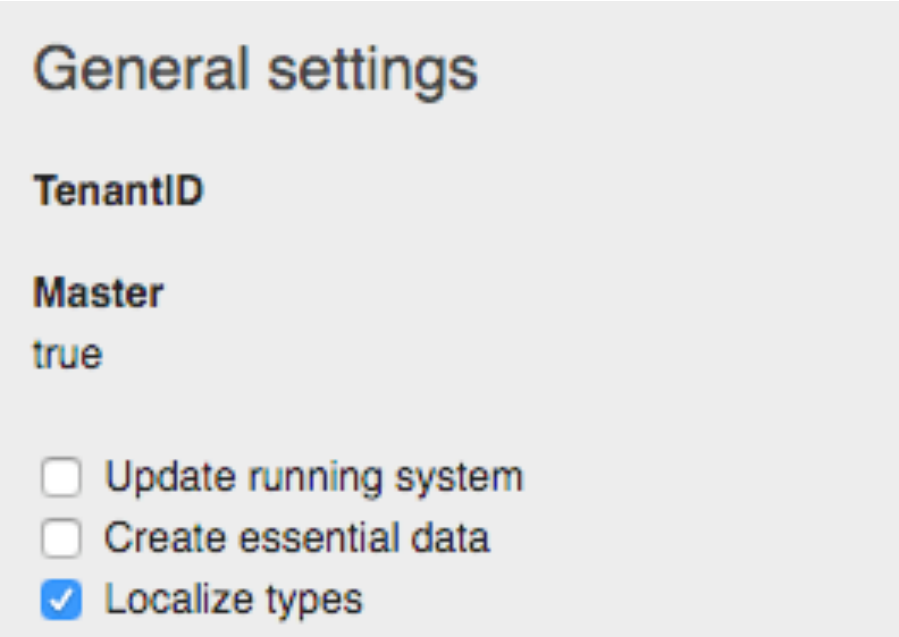
- Language-specific labels are used for displaying type-names and attribute-names
  - Used in Backoffice, based on session language setting
  - Specified in files named `extension-locales_XY.properties`, where:
    - `extension` is the name of the extension
    - `XY` is the ISO code of the language / locale
    - Properties convention (i.e., entries within this .properties file):

```
type.{typename}.name=value
type.{typename}.description=value
type.{typename}.{attributename}.name=value
type.{typename}.{attributename}.description=value
type.{enumcode}.{valuecode}.name=value
```

For Backoffice employees in different countries

Localize Types during System Initialization or Update

- To read Type-definition localizations from .properties files and store them in the database metadata tables:



- Overrides type localizations in the database with the ones from the `locales_XY.properties` files

Localizing Attribute *Values*

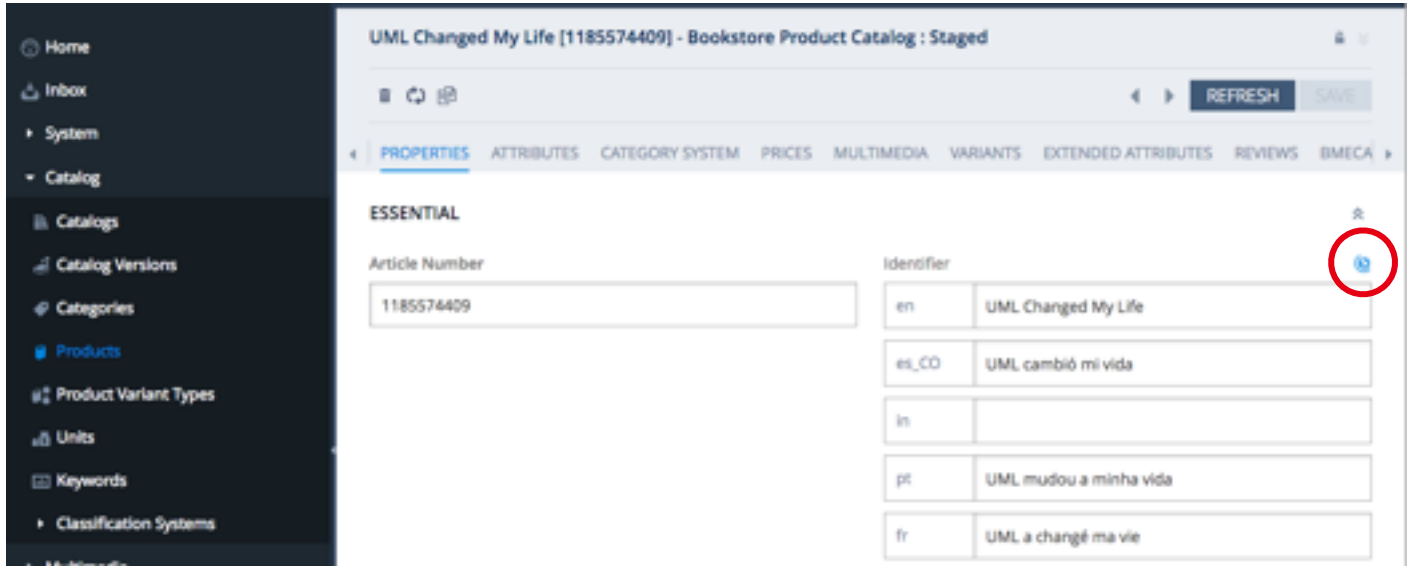
- Any itemtype property may be localized

```
<itemtype code="Product">
   <attribute qualifier="barcode" type="java.lang.String" />
   <attribute qualifier="quote" type="localized:java.lang.String" />
   <attribute qualifier="mugshot" type="localized:Image" />
   . . .
```

- Backoffice apps and ImpEx will allow input in multiple languages

For front-end customers around the globe

1. **Define in *-items.xml**
2. **Populate in Backoffice or ImpEx**
3. **Displayed in storefront based on customer's locale**



Enabling Localized Data Types

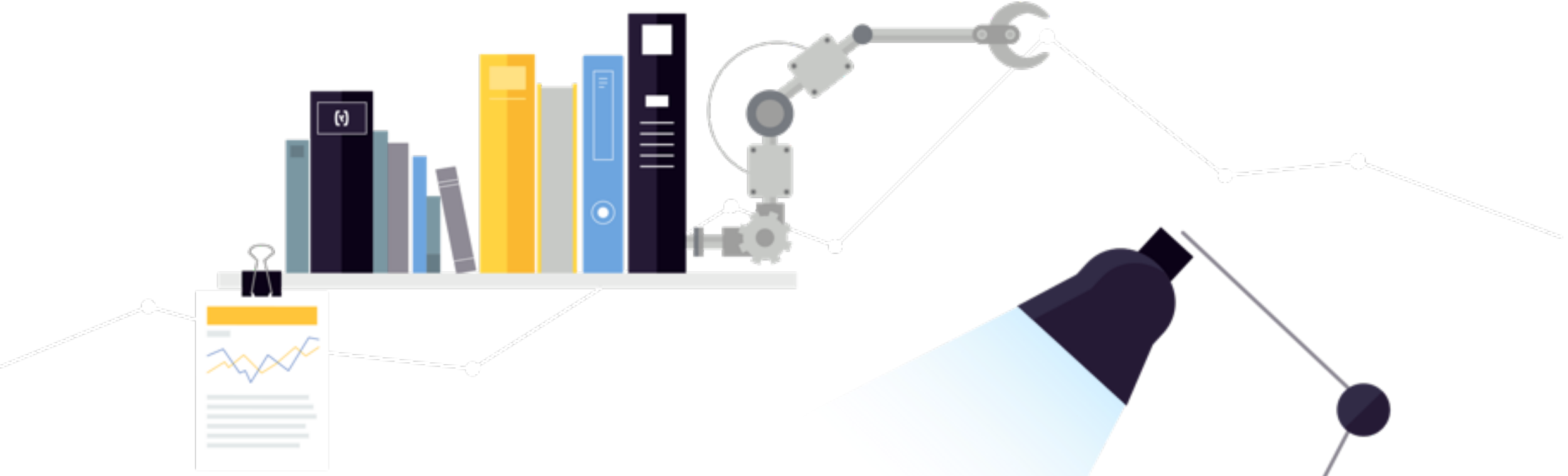
- The `localized:` prefix is not a keyword; localized types must be defined in `*-items.xml`
  - A <maptype> entry exists for each OOTB localized type, defined in the core extension's `core-items.xml`
- For example

```
<itemtypes>
   <itemtype code="Car" extends="Product">
      <attribute qualifier="ownerManual" type="localized:Booklet" />
      ...
   </itemtype>
   <itemtype code="Booklet" extends="Product">
      ...
   </itemtype>
   ...
<itemtypes>
```

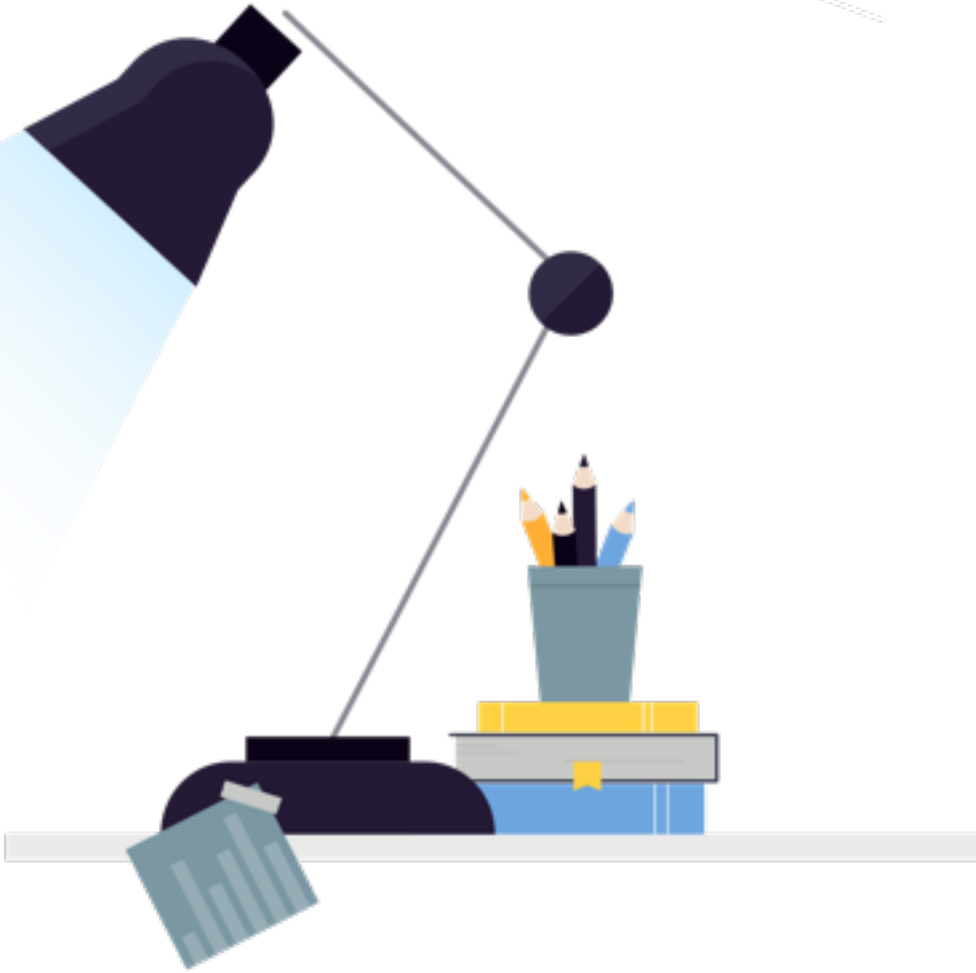
If we wanted to define a localized user's manual for our car…

```
<maptypes>
   <maptype code="localized:Booklet" argumenttype="Language"
            returntype="Booklet" autocreate="true"
generate="false">
   ...
<maptypes>
```

We must also define the `localized:Booklet` maptype



Demo



References

Type System Documentation:

- https://help.sap.com/viewer/d0224eca81e249cb821f2cdf45a82ace/1905/en-US/8c755da8866910149c27ec908fc577ef.html

Type System Definition Items.xml:

- https://help.sap.com/viewer/d0224eca81e249cb821f2cdf45a82ace/1905/en-US/8bffa9cc86691014bb70ac2d012708bc.html

Specifying a Deployment for SAP Commerce Cloud Platform Types:

- https://help.sap.com/viewer/d0224eca81e249cb821f2cdf45a82ace/1905/en-US/8c6254f086691014b095a08a61d1efed.html

Data Model Design Resources and Performance Implications

- https://www.sap.com/cxworks/article/433893244/Data_Model_Design_with_the_SAP_Commerce_Cloud_Type_System

Data Modeling Guidelines

- https://help.sap.com/viewer/3fb5dcdfe37f40edbac7098ed40442c0/1905/en-US/8ecae959b9bd46b8b426fa8dbde5cac4.html

The SAP Commerce type system is used to model system in an **abstract** way.

All XML type definitions are converted during the system build into corresponding java classes, which will be used at runtime.

Item types convert to models – the foundation/entities of the Commerce Suite's type system.
Each model is comprised of attribute and relation metadata, an ID, a DB table and a supporting Java class.

Always **update/initialize** SAP Commerce to apply any type-related changes to the database.

If possible, try to use **relation types** rather than collections

The deployment tag deploys the current type to its own table instead of using the table of its super type.

- Understand the performance impact

Localization in SAP Commerce is two-fold:

- Types and attributes description localization – useful for Backoffice users
- Localized attributes – seen by storefront customers based on their locale

Exercise

DataModeling

Thank you.