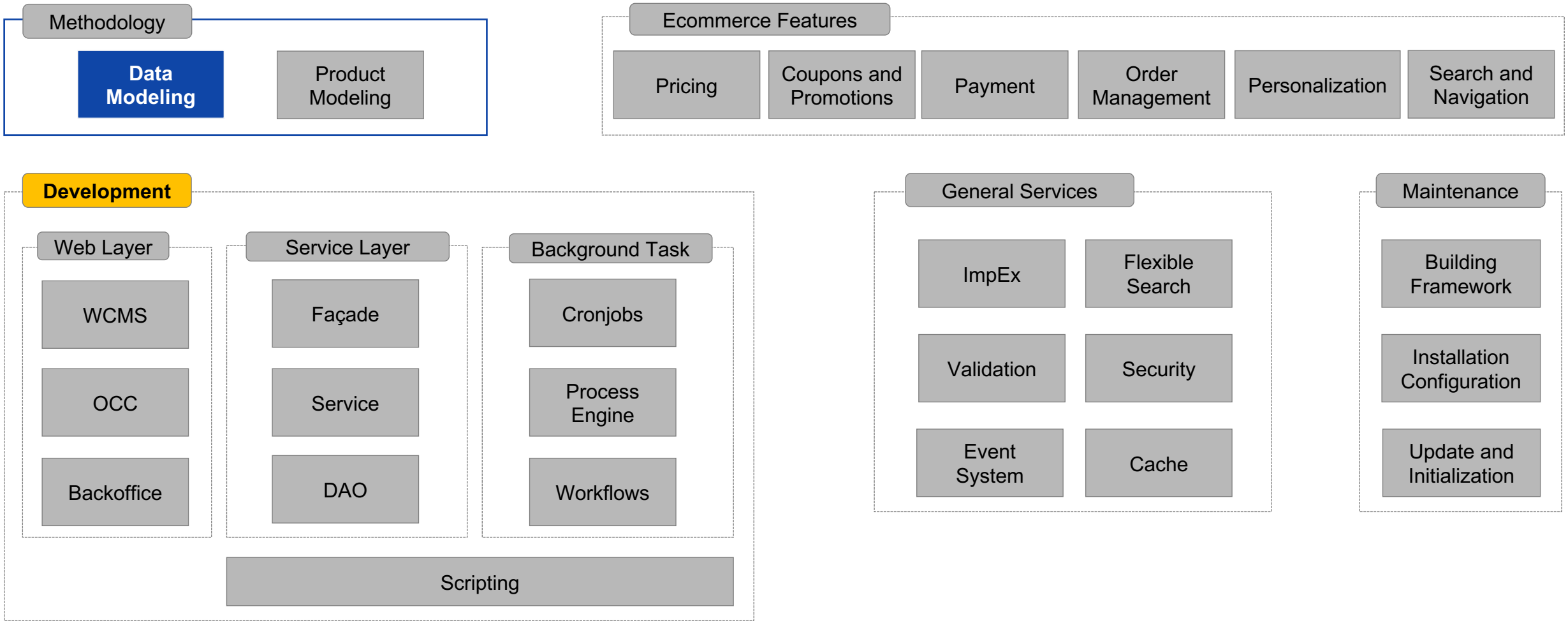
SAP Customer Experience

Data Modeling

What we will cover in this topic



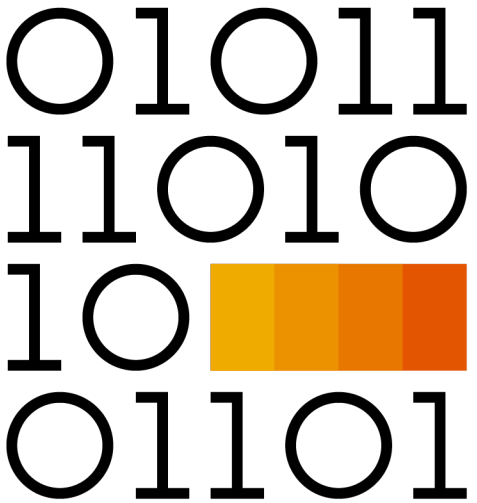
We will learn about:

- Introduction to the Type System
- Collections and Relations
- Deployment
- Type-System Localization

Preparation

Complete step P1 of the Data Modeling exercise

(The setup ant target will compile your system during the lecture)



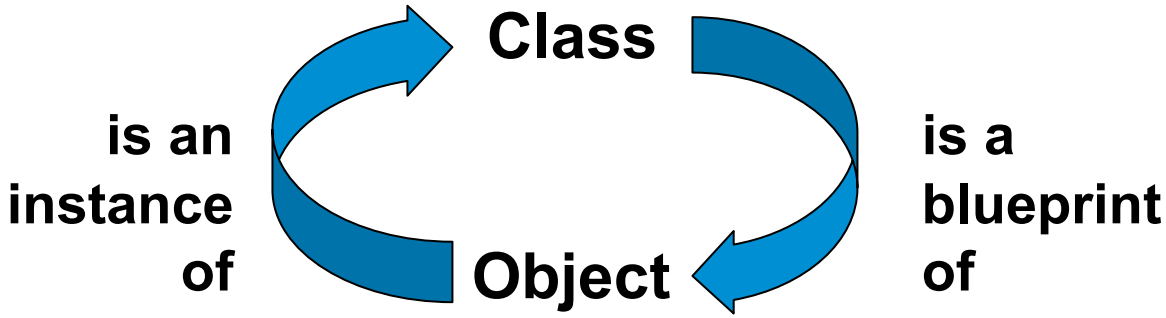
Introduction to the Type System

SAP Commerce and Java

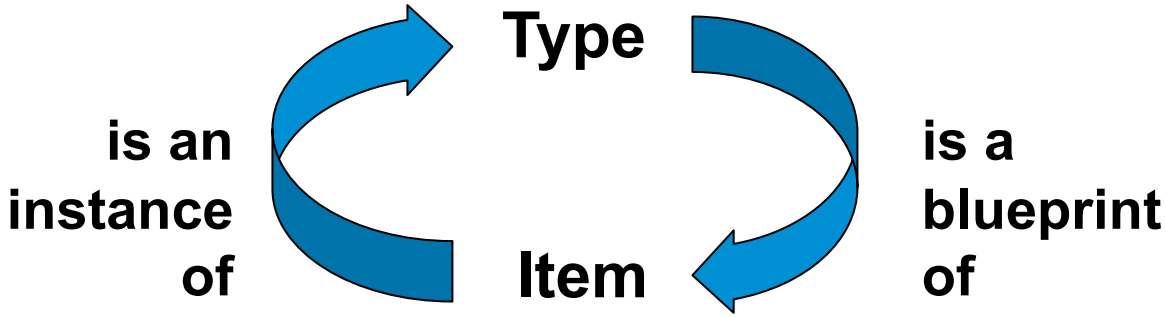
Who is responsible for converting SAP Commerce type definitions to Java classes?

The SAP Commerce build system (ant-based)

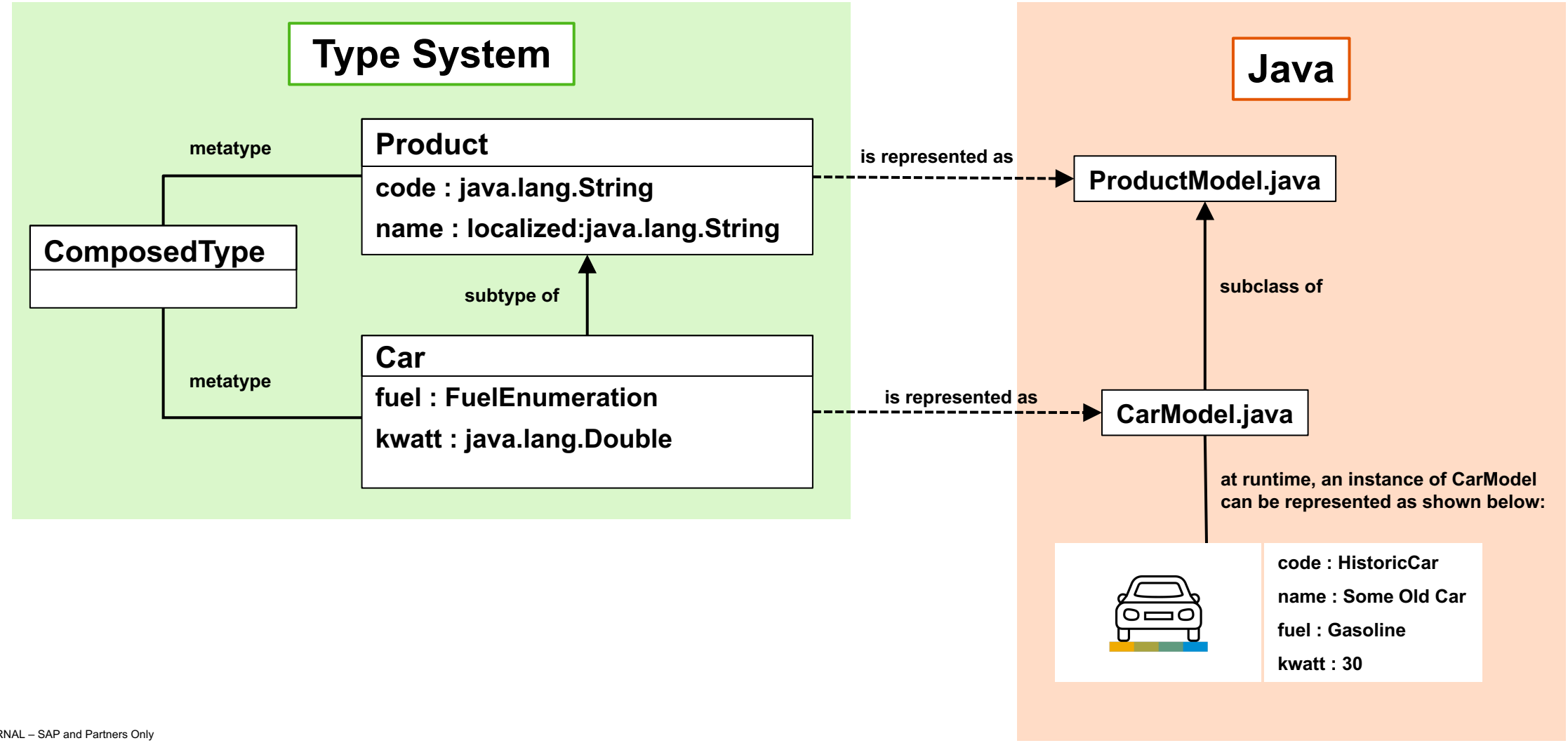
Java



SAP Commerce



SAP Commerce Types vs Java Classes



Types used in SAP Commerce Cloud

AtomicType

- Represents Java value objects which are mapped to database types
  - Java Primitive keywords: `int`
    (actually, a `java.lang.Integer` that initializes to `Integer.valueOf(0)`)
  - 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)

- Records 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 new type which is a specialization (sub-type) of an existing type

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

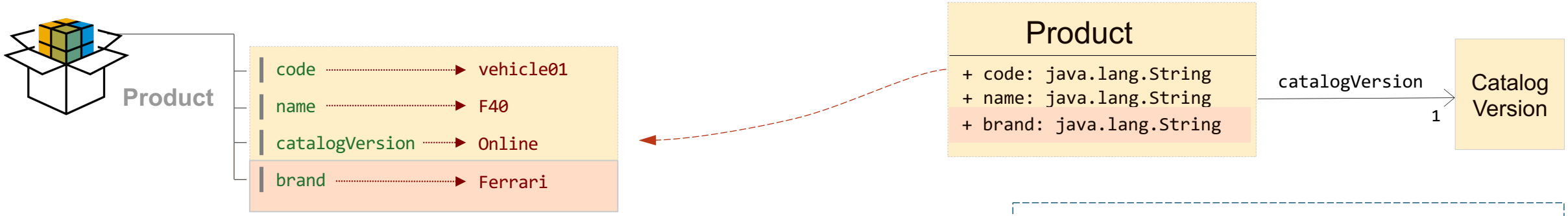
  - Define “completely new” types

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

- Expand the definition of an existing type
  - Add attribute definition(s) (a.k.a., attribute injection) to a type defined in another extension

```
<itemtype code="Product" autocreate="false" ...>
   ...
   <attributes>
      ...
      <attribute qualifier="myAttribute" type="java.lang.String" >
         <persistence type="property"/>
      </attribute>
      ...
   </attributes>
</itemtype>
```

Extending the Data Model - Adding Attributes to an Existing Type



- Add attribute definitions to existing types (attribute injection)

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

(no **extends**)

autocreate

true: this XML entry is intended to be the start of a *new* ItemType definition (i.e., another ItemType having the same **code** must not already be defined in another extension).

false: this XML entry is intended to merge with an *existing* type definition (i.e. an ItemType having this **code** must already be defined in another extension).

As *Product* has already been defined in a core extension, set to **false**.

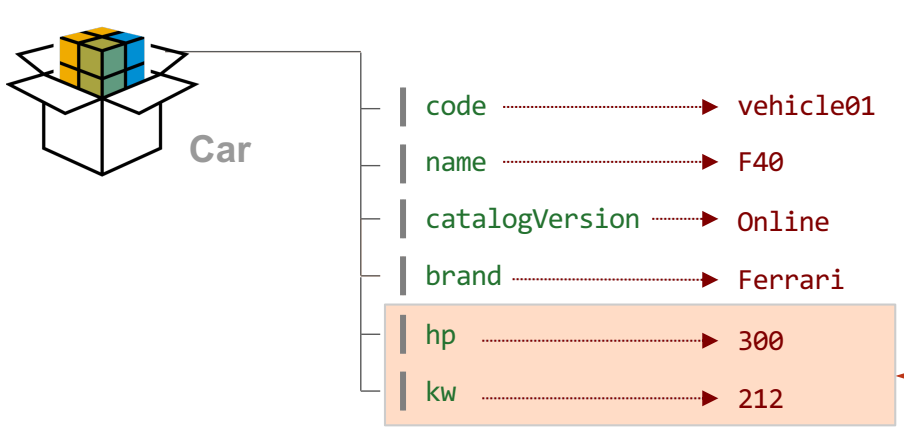
generate

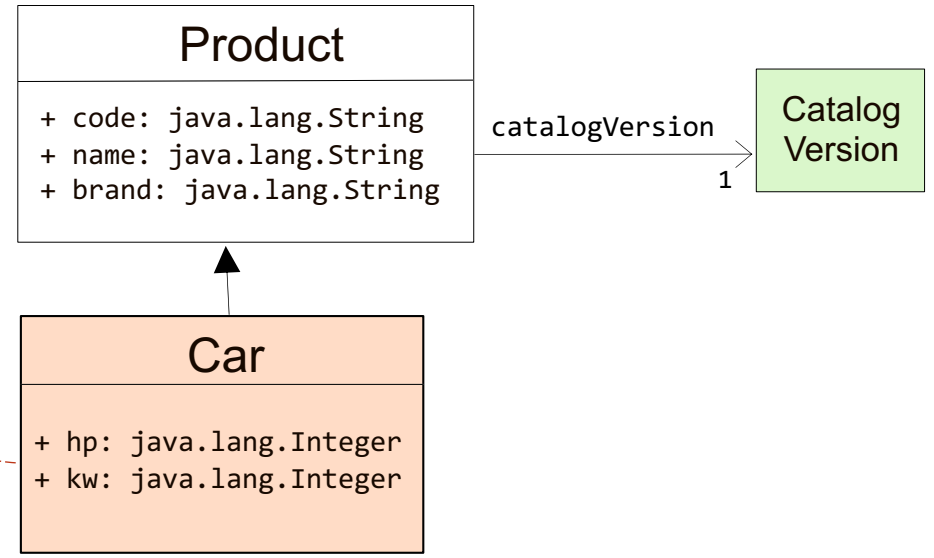
IGNORED if autocreate="**false**".

true: build process generates a corresponding model class for you (e.g. CarModel) in Java domain.

false: build process will not generate a model class.

Extending the Data Model - Defining a New Subtype





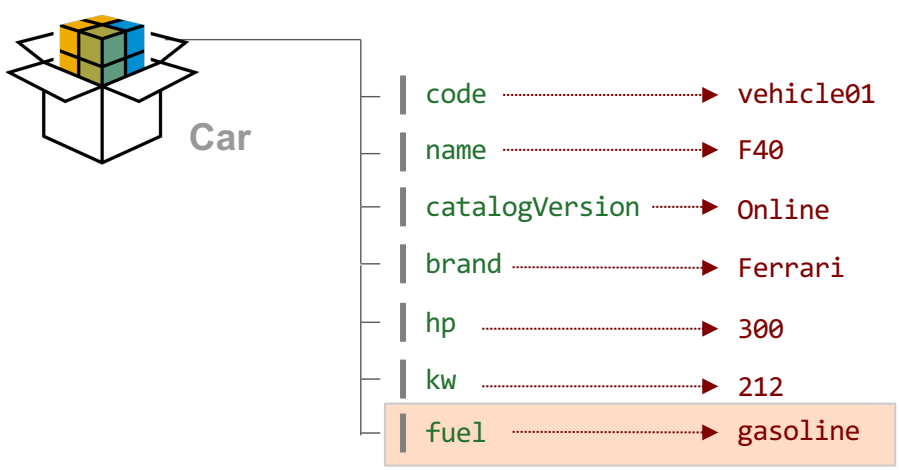

```
<items>
  <itemtypes>
    <itemtype code="Car" extends="Product" autocreate="true" generate="true">
      <attributes>
        <attribute qualifier="hp" type="java.lang.Integer">
          <description>Horsepower</description>
          <persistence type="property" />
        </attribute>
      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

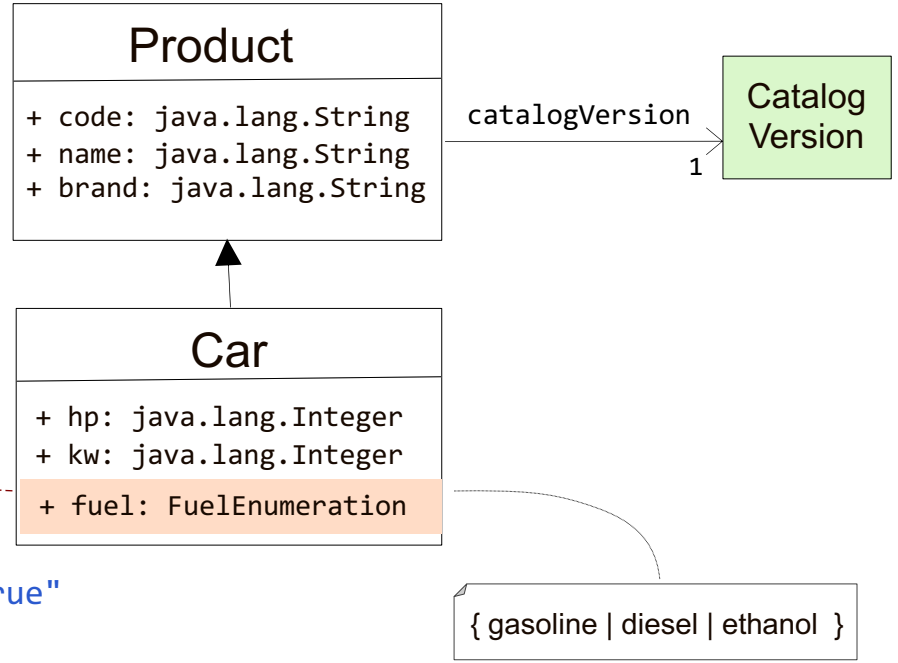
? What is the difference between *property* and *dynamic*?

? In what conditions will *dynamic* be used?

More details about dynamic attribute are covered in the live session series: SAP Commerce Cloud – Additional Technical Essentials

Extending the Data Model - Enumerated Types

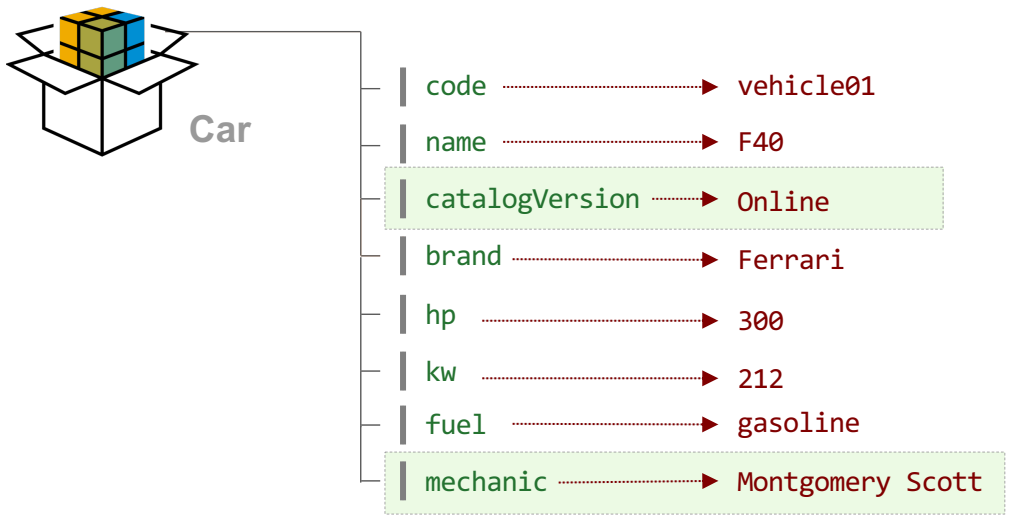


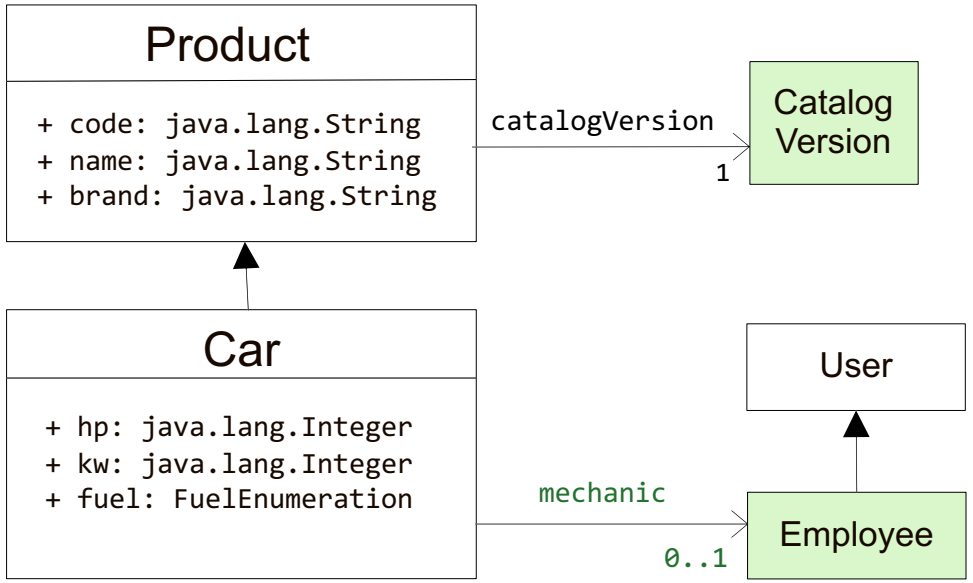



```
<items>
  <enumtypes>
    <enumtype code="FuelEnumeration" generate="true" autocreate="true"
              dynamic="true">
       <value code="diesel" />
       <value code="gasoline" />
       <value code="ethanol" />
    </enumtype>
  </enumtypes>

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

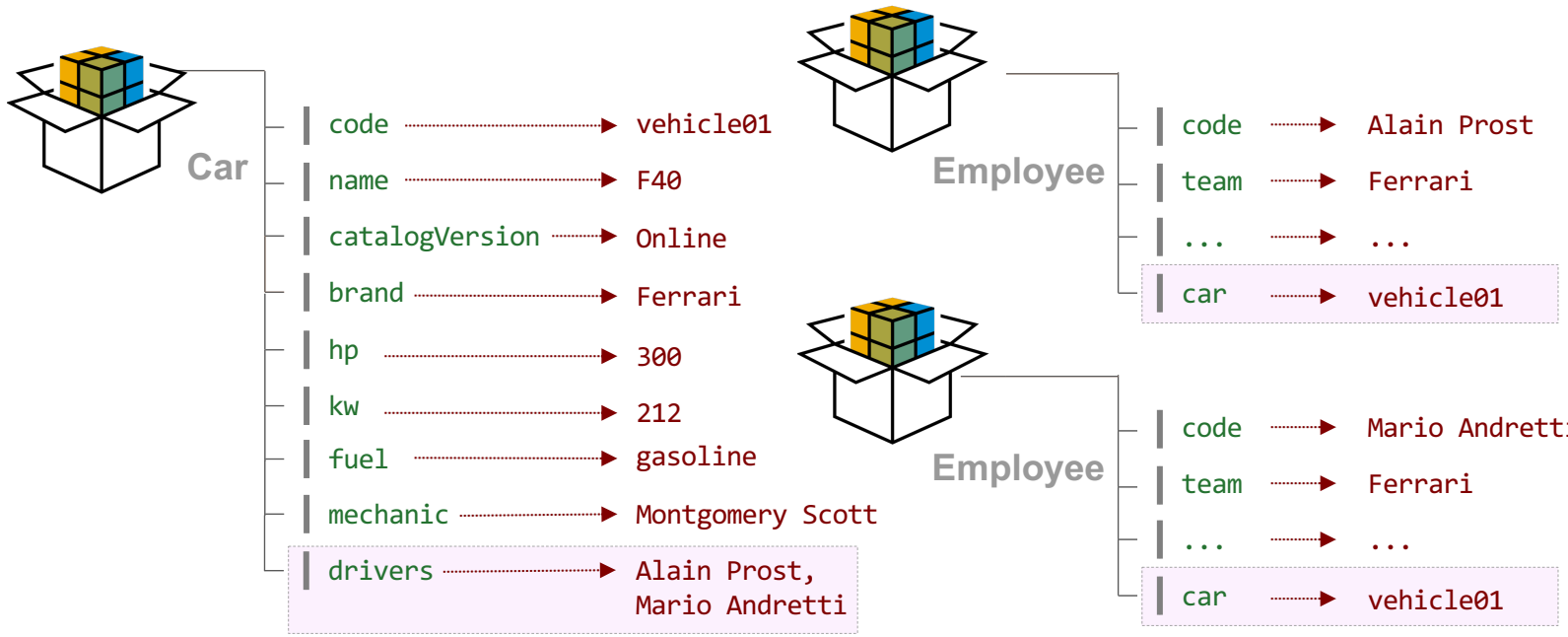
Extending the Data Model - Composed-Type References

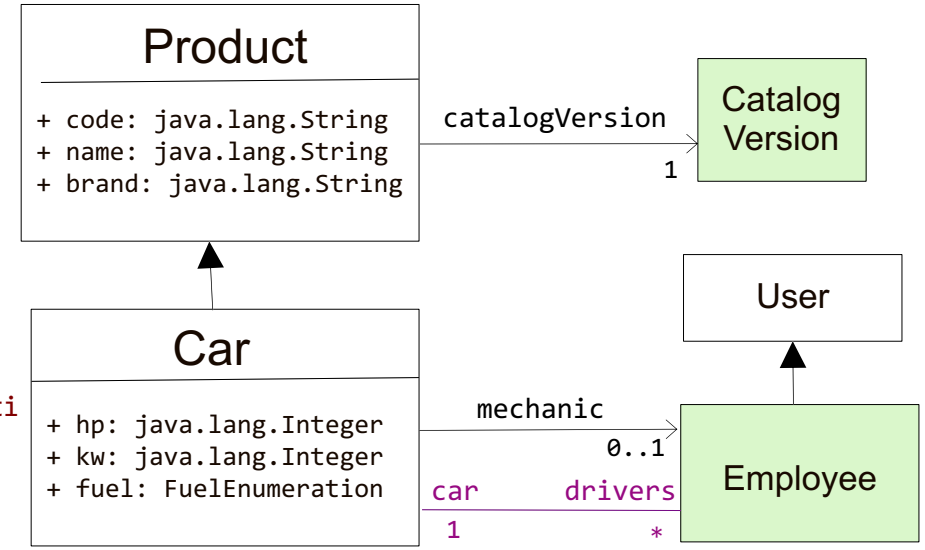





```
<items>
  <itemtypes>
    <itemtype code="Car" extends="Product" autocreate="true" generate="true">
      <attributes>
        ...
        <attribute qualifier="mechanic" type="Employee">
          <modifiers read="true" write="true" search="true" optional="true"/>
          <persistence type="property"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

Extending the Data Model - Relations






```
<items>
  ...
  <relations>
    <relation code="Car2DriverRelation"
              localized="false" autocreate="true" generate="true">
      <sourceElement qualifier="car" type="Car" cardinality="one" />
      <targetElement qualifier="drivers" type="Employee" cardinality="many" />
    </relation>
  </relations>

 <itemtypes>
   ...
```

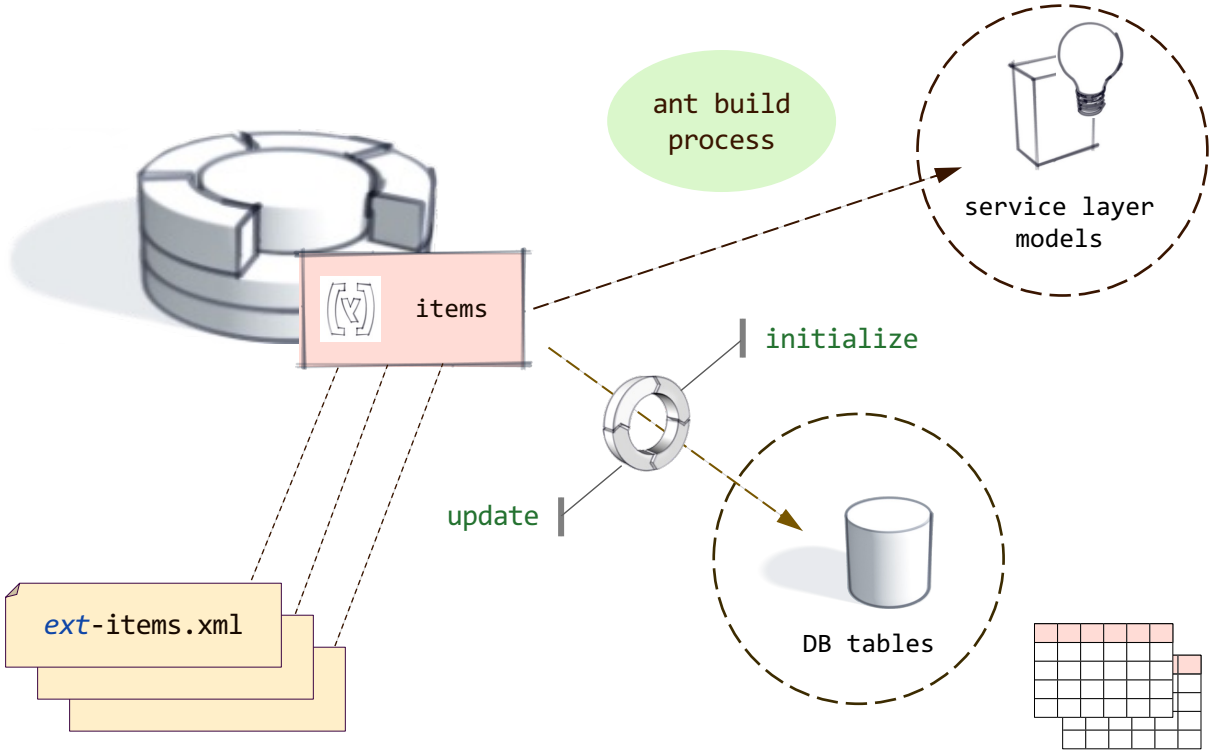
This example is 1:many, but many:many relations are also supported

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

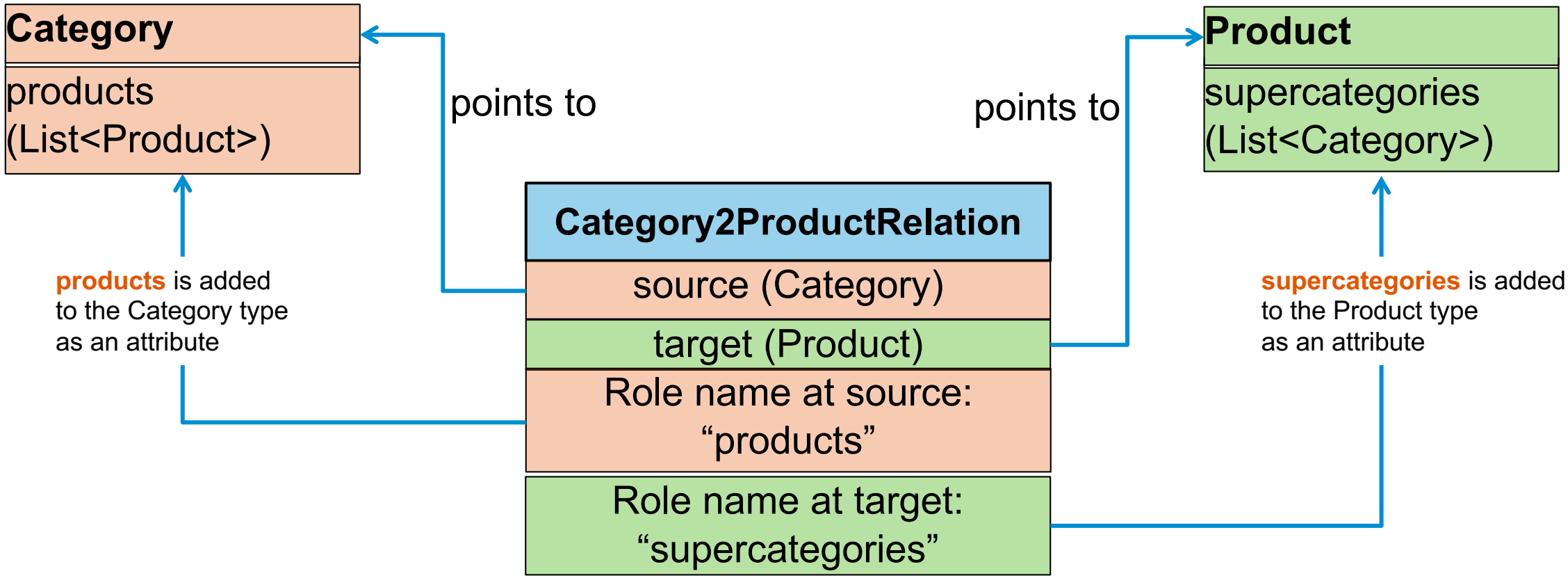
```
<collectiontype code="StringCollection"
                elementtype="java.lang.String" autocreate="true" />
<collectiontype code="LanguageCollection"
                 elementtype="Language" autocreate="true"/>

...

<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
- Riskier
  e.g., Can cause "silent" value truncation if db "default column width" is too small: `varchar(80)`
- More difficult to search
- Generally lower-performance for *large* collections
  (but Generally higher-performance for *small* collections, as it avoids a join)

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:

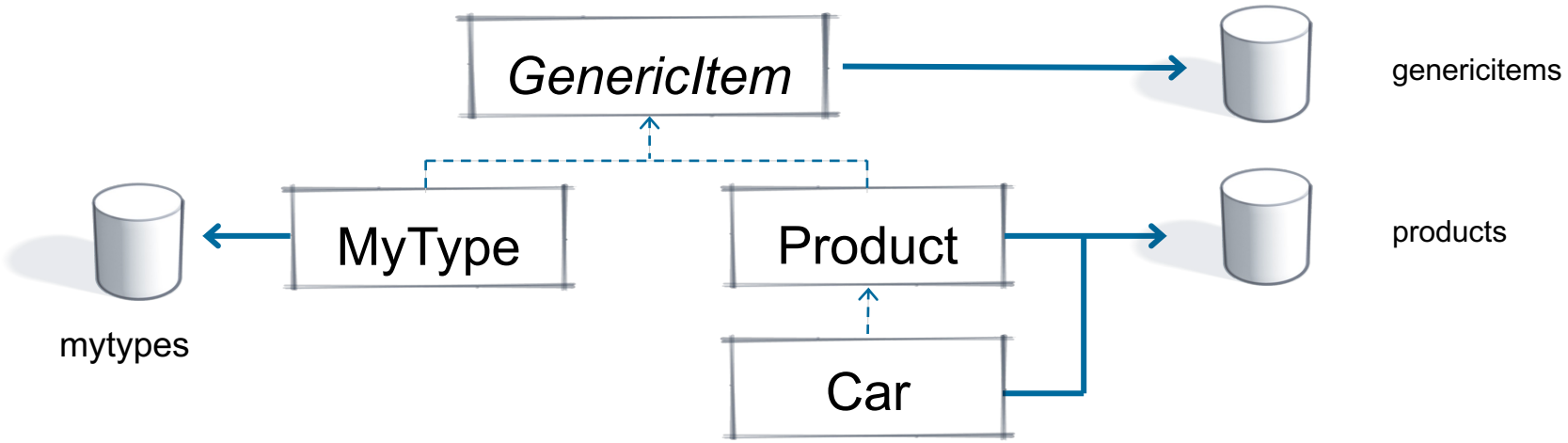
1. **Subtype shares the supertype's table**
2. **Use different tables for supertype and subtype**

What are the advantages and drawbacks of these two strategies?

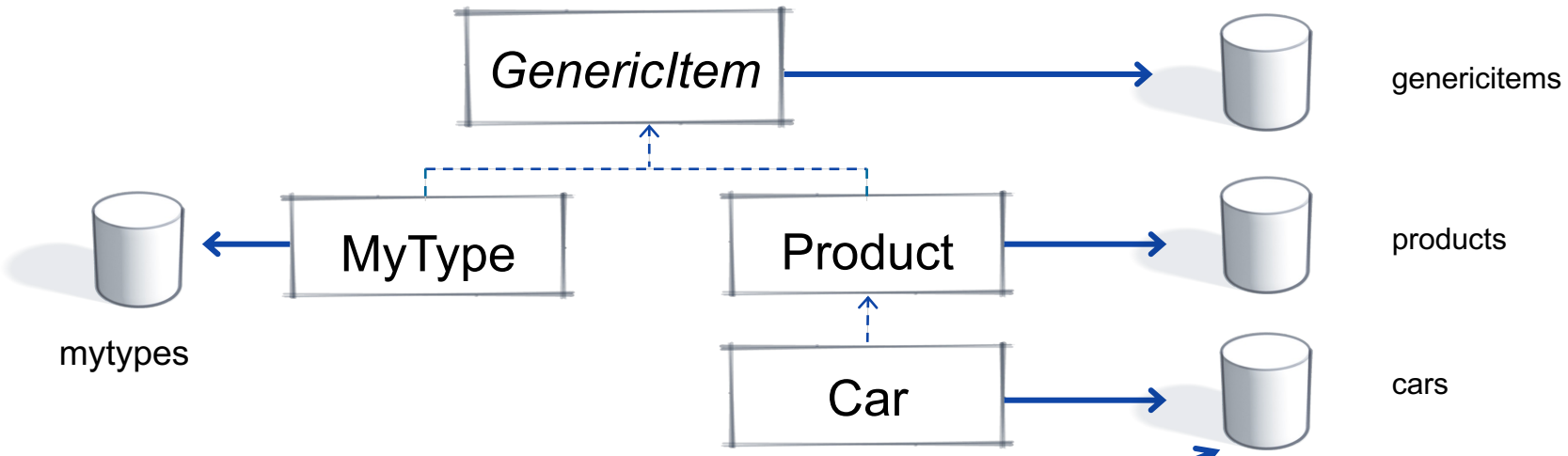
- **Single-table strategy: better query performance but low storage efficiency**
- **Multiple-table strategy: worse query performance but high storage efficiency**

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
  - Required by default (in development mode) by setting
    `build.development.mode = true`
    ...which mandates that all direct children of GenericItem have *deployment* specified.



O-R Mapping - Deployment Example




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

O-R Mapping - TypeCode of Deployment table

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

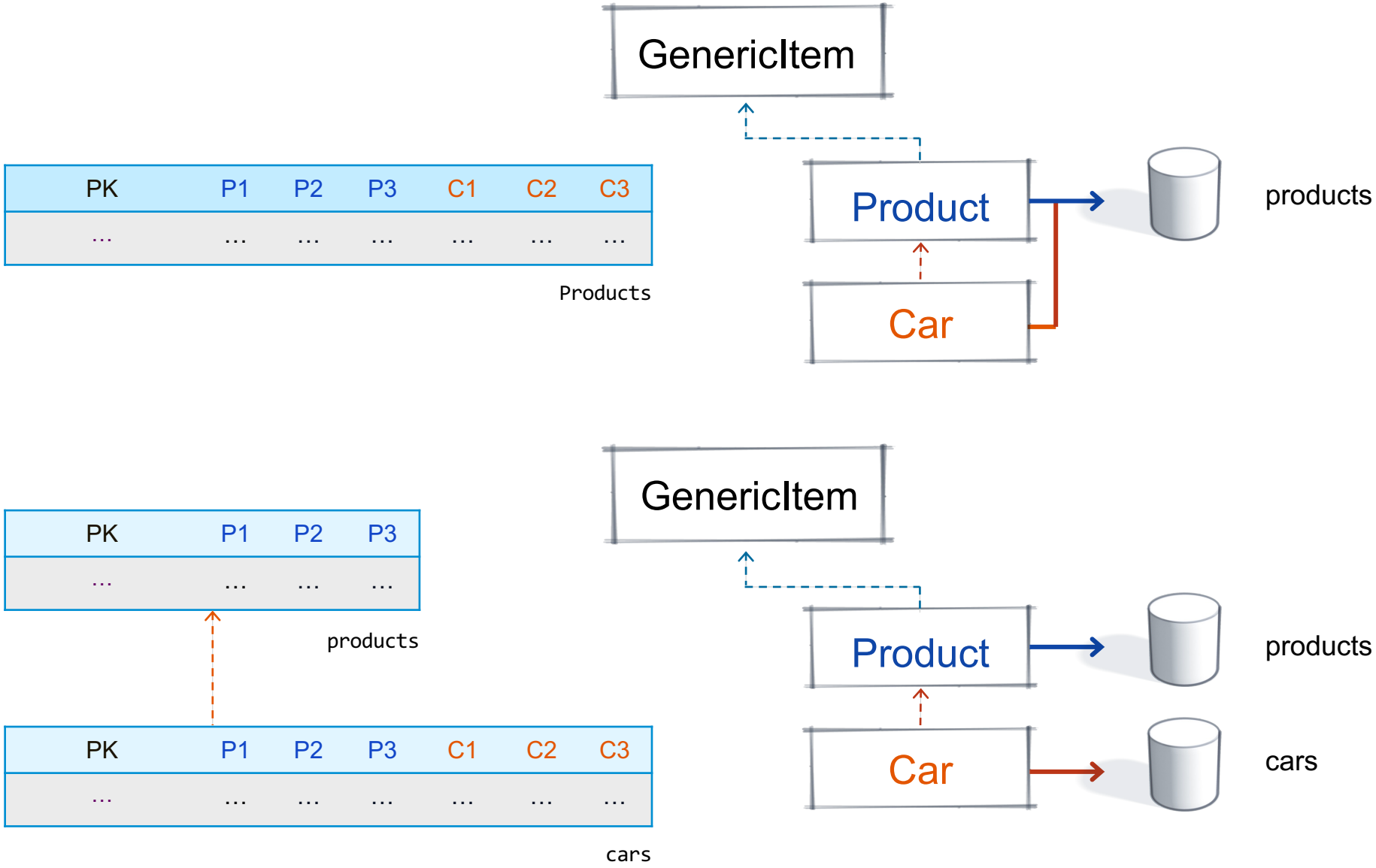
- Always maintain a registry of the typecodes used by the deployment tables created in a project
- Always check SAP Commerce Help pages to look for possible exceptions

`0 .. 10000` are reserved by SAP Commerce
`132xx` are reserved by *commons* extension
`327xx` are reserved by *processing* extension
`244xx` and `245xx` are reserved by legacy *xprint* extension
`100xx` are reserved by *b2bcommerce* extension

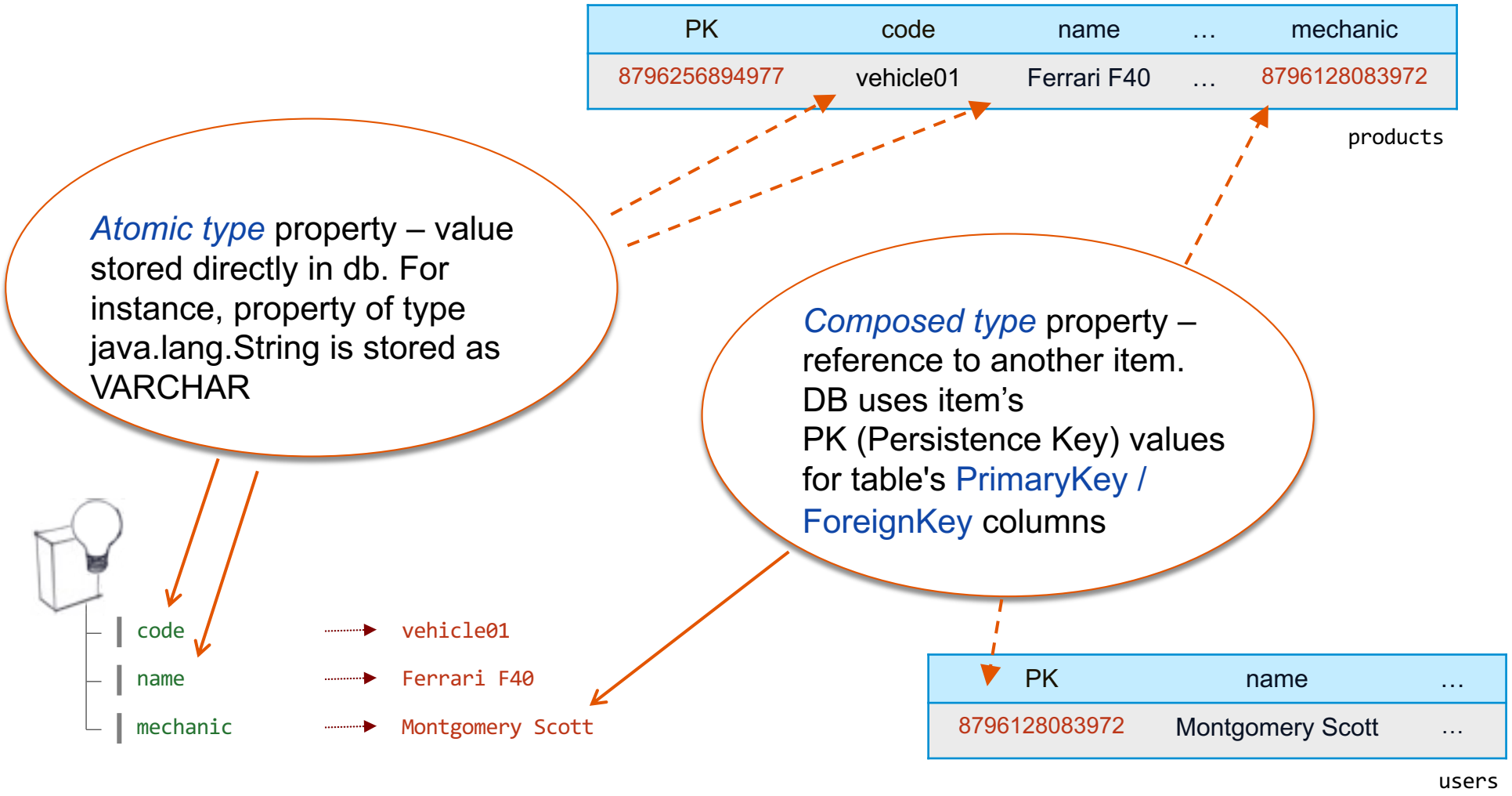
- For a full list of exceptions, see the file

`<HYBRIS_BIN_DIR>/platform/ext/core/resources/core/unittest/reservedTypecodes.txt`

O-R Mapping - Table Structure



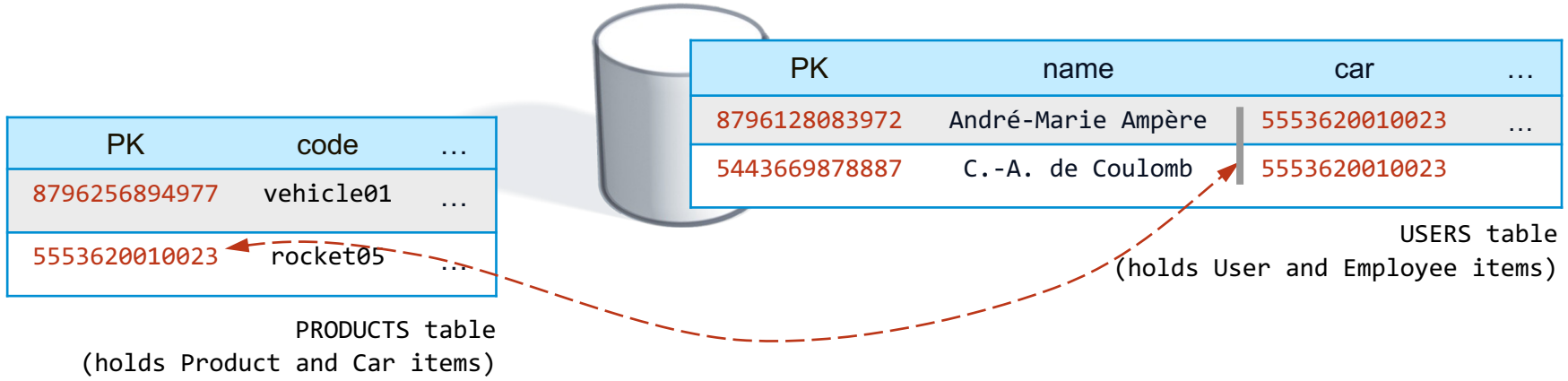
O-R Mapping - Attributes of a (Composed) Type



O-R Mapping - Deployment of Relations : 1

- **One-to-Many**
  - Additional column in the *many* side which contains the PK of the *one* side
  - Users table from the example below would have an additional column car
    - Like Car, the 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>
```

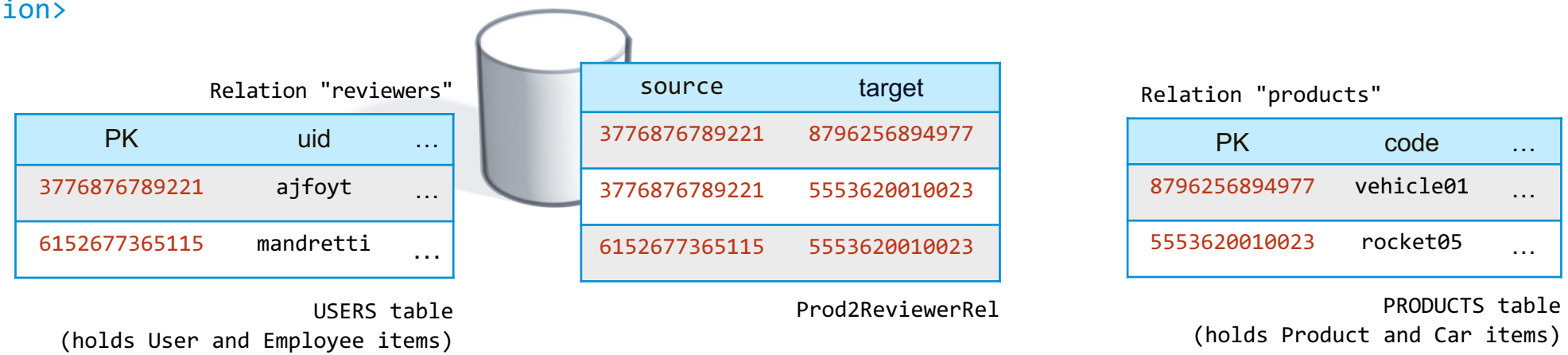


O-R Mapping - Deployment of Relations : 2

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>
```



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 Areas (and Mechanisms) for Localization

? So, who needs what?

1. **Localizing *names* in *type definitions*:**

   You can configure language-specific **display labels** for:

   - Type *names*
   - Attribute *names*
   - Enumeration-value *names*

Backoffice employees in different countries

Done using key/value entries within `.properties` files (i.e. using Java I18n resource bundles)

Front-end customers around the globe

2. **Localizing attribute *values*:**

   You can allow language-specific **attribute values**

   - Declare attribute's ***type*** as "`localized:java.lang.String`"

`localized:<SAP_Commerce_type>` is a special kind of Map declaration whose values are keyed by language ISO_code

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

- Language-specific labels are used (in place of declared **code** and **qualifier** from *`*-items.xml`*) when displaying type names and attribute names
  - Used within Backoffice, based on session-language setting
  - Specified in files named *`extensionName`*`-locales_`***`xy`**`.properties`, where:
    - *`extensionName`* is the name of the extension
    - ***`xy`*** is the ISO code of the language / locale
    - Properties convention (i.e., format of key-value pairs 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
```

(Detailed examples on next slide)

For Backoffice employees in different countries

Type-System Localization - Example

.../myextension/resources/localization/**myextension-locales_en.properties**

```
type.Car.name=Car
type.Car.description=A car in the product catalog
type.Car.hp.name=Horsepower
type.Car.hp.description=Horsepower of stock engine
type.FuelEnumeration.gasoline.name=gasoline
type.FuelEnumeration.ethanol.name=ethanol
```

.../myextension/resources/localization/**myextension-locales_en_UK.properties**

```
type.FuelEnumeration.gasoline.name=petrol
```

.../myextension/resources/localization/**myextension-locales_es.properties**

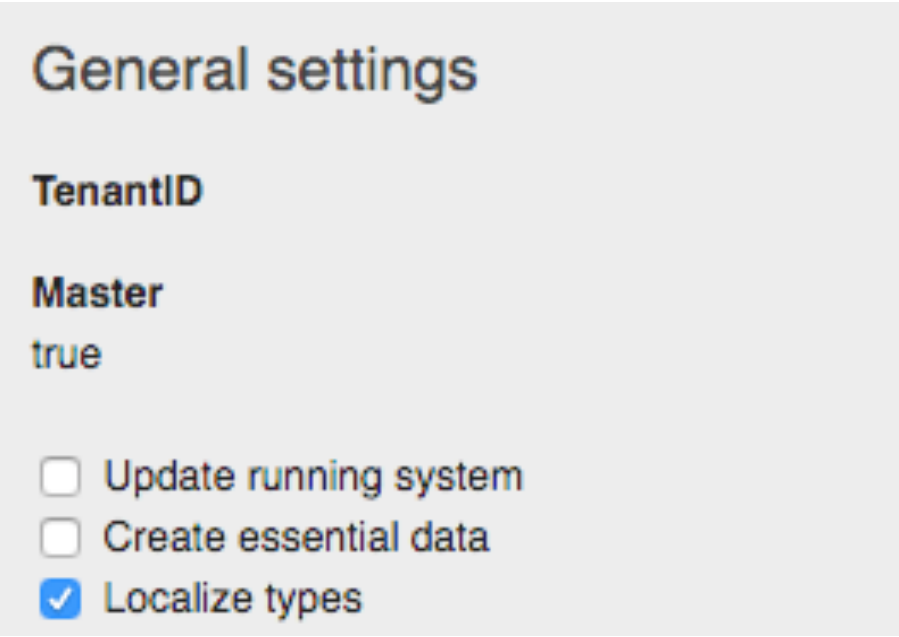
```
type.Car.name=Coche
type.Car.description=Un auto en mi catalogo de productos
type.Car.hp.name=Caballos de fuerza
type.Car.hp.description=Caballos de fuerza del motor de serie
type.FuelEnumeration.gasoline.name=gasolina
type.FuelEnumeration.ethanol.name=etanol
```

.../myextension/resources/localization/**myextension-locales_es_MX.properties**

```
type.Car.name=Carro
```

Localize Types during System Initialization or Update

- To (re)read type-definition localizations from .properties files and persist them in the database metadata tables:



- Replaces existing type-localization values in the database with the ones from the `locales_xy.properties` files

Allowing Localized Attribute Values

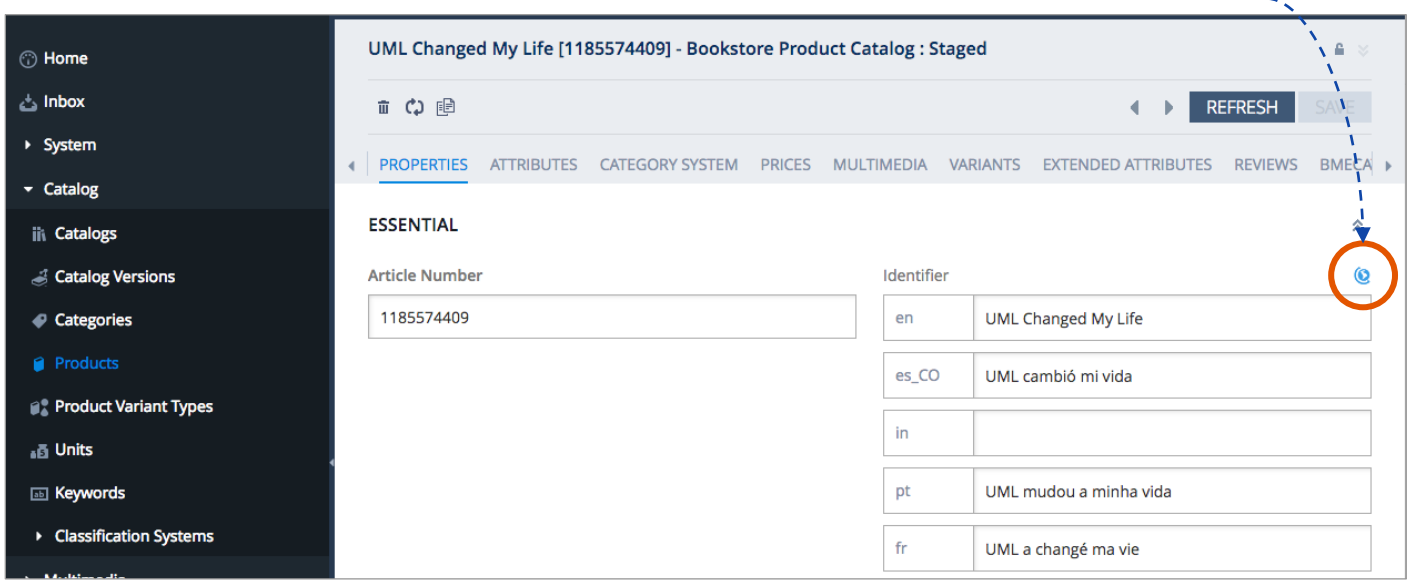
- Any itemtype property may be localized

```
<itemtype code="Product">
   <attribute qualifier=”code" type="java.lang.String" />
   <attribute qualifier=”name" 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 explicitly 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>
```

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

If we wanted each Car instance to be able to reference a distinct "owner manual" per Language…

We must also define the `localized:Booklet` maptype

Demo


R100 BIKE
849.40
R300 Bike
373.69
R200 Bike
318.97
R200 BIKe 4s
596.34
M500 BIKE
209.16
eBike E180
75.44
M550 BIKE
74.40
Y200 Bike
62.31
eBike E181
54.06
C900 BIKE
42.47
C950 BIKE
29.90
M525 BIKE
135.61
Y120 Bike
28.38
Horns
19.07
Y240 Bike
43.84
in Thousand USD, %
2,987.26 (+30.59%)
Product Revenue Won Current
in Thousand USD
444.48
Revenue New Products
0.1
0.05
0
-0.05
-0.1
Lost due to price
348
Won due to product
Won due to price
Won due to service
1,657
Top Customers
Dimension
Industry ID
Territory
Sales Unit
Country
Competitor
Measure
in Thousand USD
NA
EMEA
APJ

References

Type System Documentation:

- https://help.sap.com/docs/SAP_COMMERCE_CLOUD_PUBLIC_CLOUD/aa417173fe4a4ba5a473c93eb730a417/8c755da8866910149c27ec908fc577ef.html?q=type%20system

Type System Definition Items.xml:

- https://help.sap.com/docs/SAP_COMMERCE_CLOUD_PUBLIC_CLOUD/aa417173fe4a4ba5a473c93eb730a417/8bffa9cc86691014bb70ac2d012708bc.html?q=items.xml

Specifying a Deployment for SAP Commerce Cloud Platform Types:

- https://help.sap.com/docs/SAP_COMMERCE_CLOUD_PUBLIC_CLOUD/aa417173fe4a4ba5a473c93eb730a417/8c6254f086691014b095a08a61d1efed.html?advAll=Specifying%20a%20Deployment

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/docs/SAP_COMMERCE_CLOUD_PUBLIC_CLOUD/51e73d14aedc487384e4518b60a1f5fd/8ecae959b9bd46b8b426fa8dbde5cac4.html

Key Points

1. The SAP Commerce type system is used to model system in an **abstract** way.
2. All XML type definitions are converted during the system build into corresponding java classes, which will be used at runtime.
3. 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.
4. Always **update/initialize** SAP Commerce to apply any type-related changes to the database.
5. If possible, try to use **relation types** rather than collections
6. The deployment tag deploys the current type to its own table instead of using the table of its super type.
   - Understand the performance impact
7. 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

Data Modeling Exercise

Thank you.