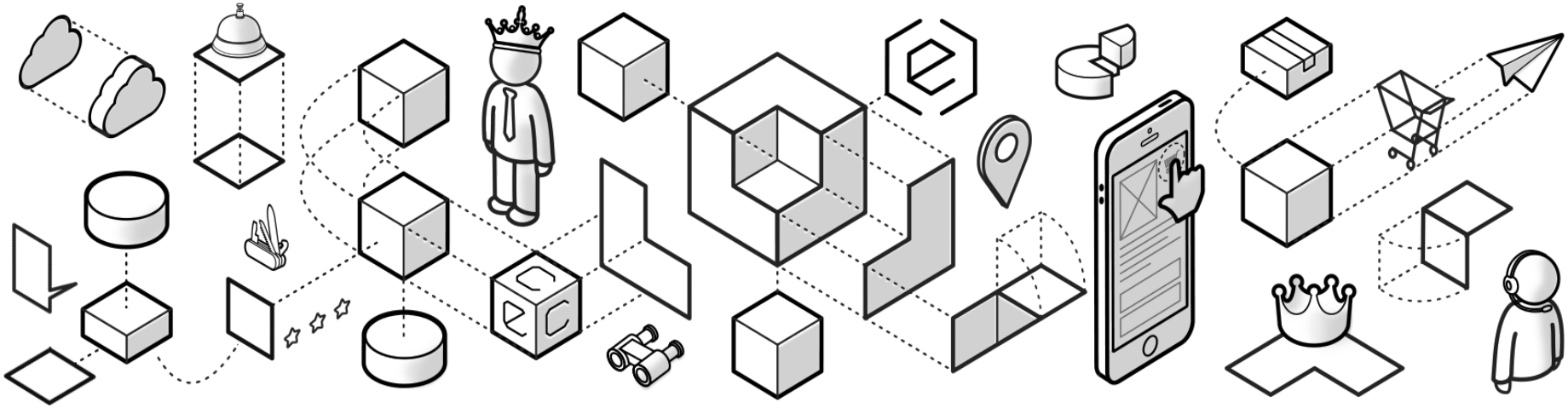
hybris software

An SAP Company

Data Modeling

hybris **Commerce Developer** Training – Part I

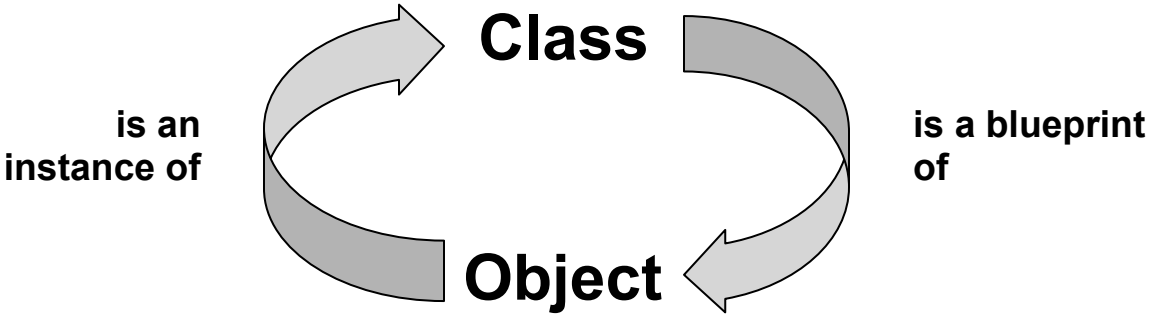


Introduction to the Type System

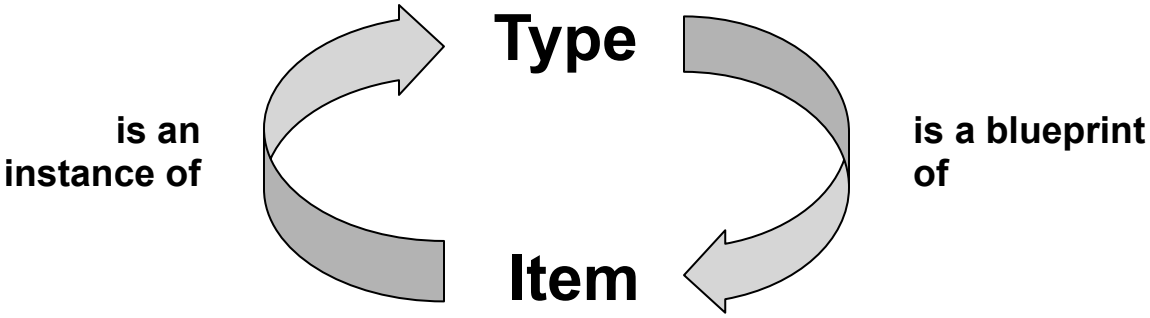
Introduction to the Type System
Collections and Relations
Deployment
Type System Localization

hybris and Java

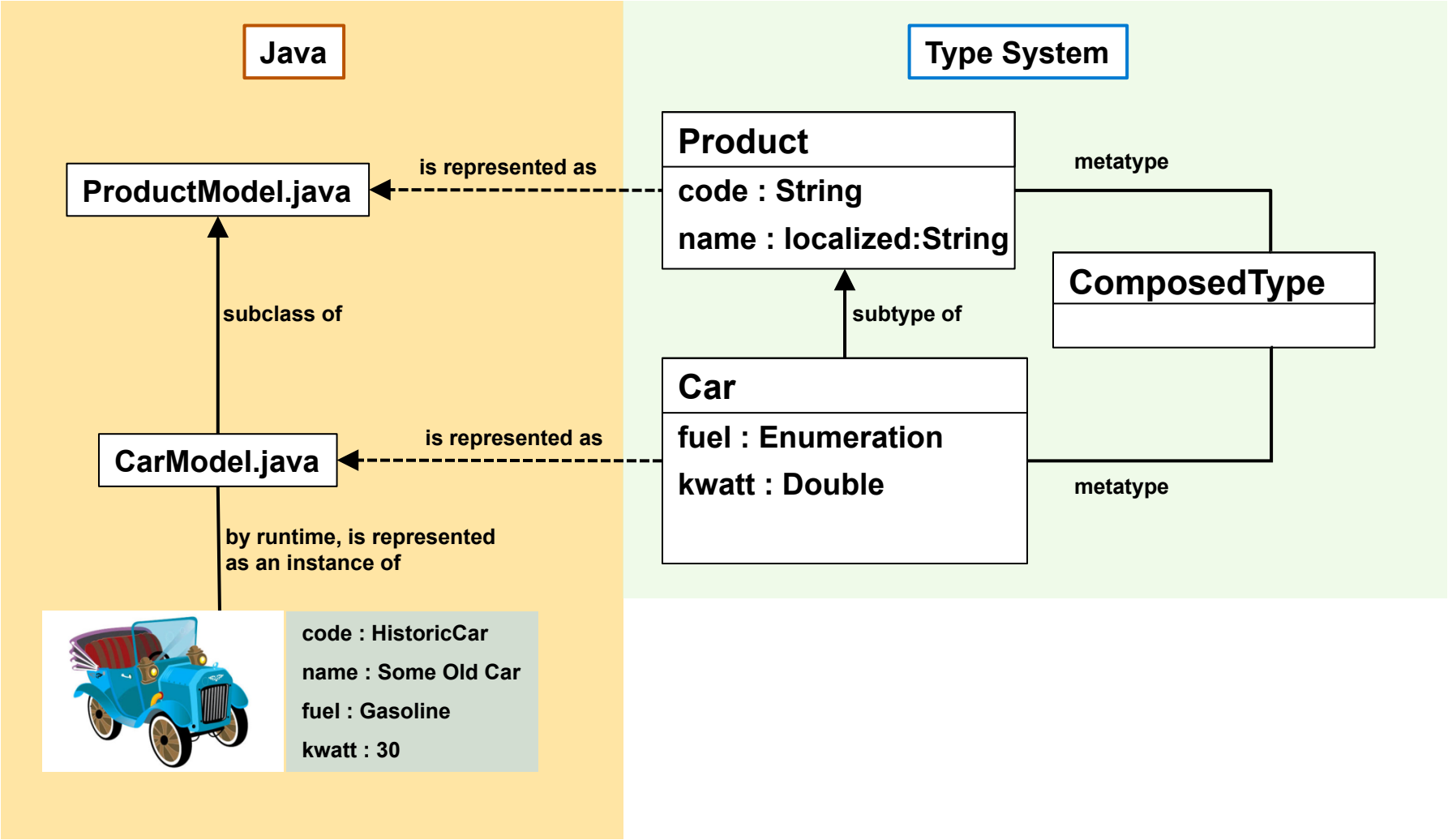
Java



hybris



Java Classes vs hybris Types



Types used in hybris

AtomicType

Represents Java value objects which are mapped to database types

- Java Primitives: `int`
- Wrapper: `Integer`
- Some Reference types: `java.util.Date`

CollectionType

Represents a typed collection

MapType

Represents a typed Map

Types used in hybris

ComposedType

Composed object of other hybris types

EnumerationMetaType

ComposedType which describes enumerations

RelationType

ComposedType which describes binary relations between 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:

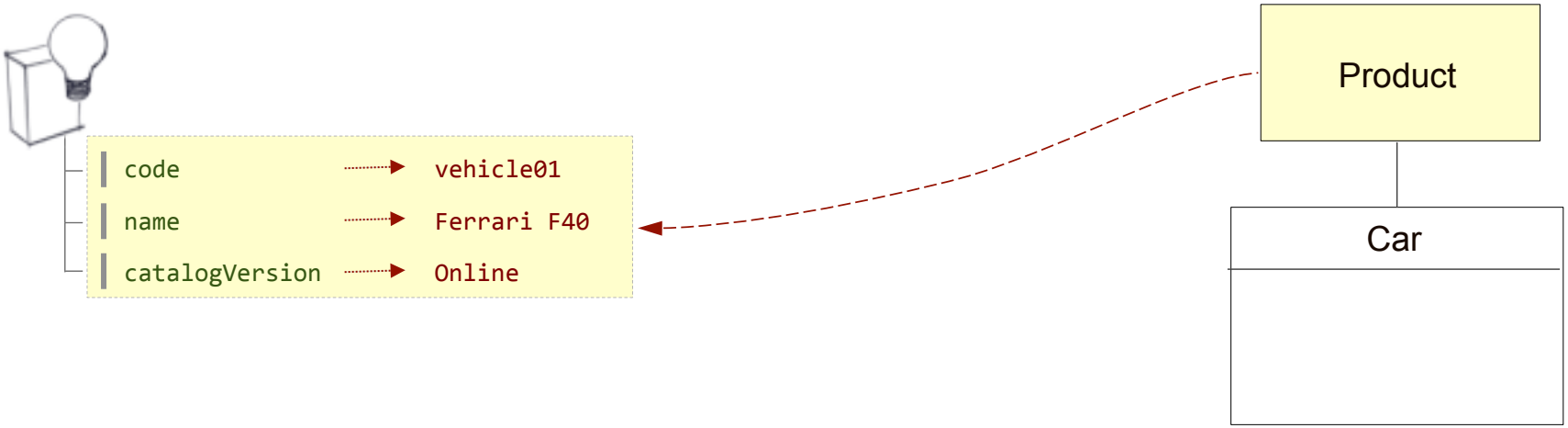
```
<itemtype code="Product" ...>
   ...
   <attributes>
      ...
      <attribute qualifier="MyAttribute">
      ...
   <attributes>
</itemtype>
```

  - Redefine inherited attribute definitions from super type

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

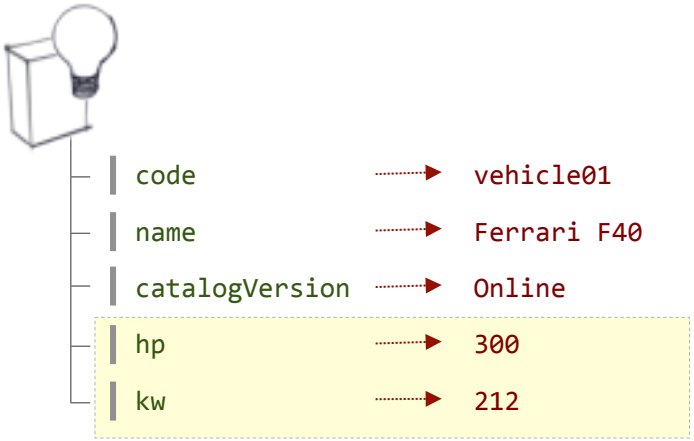
    - If you change the attribute's java type, the new type must extend the original type

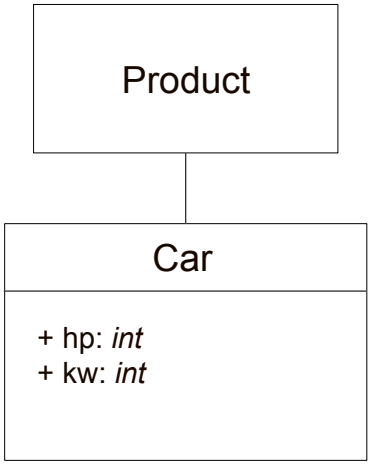
hybris Type System • Extending




```
<items>
  <itemtypes>
    <itemtype code="Car" extends="Product" autocreate="true" generate="true">
      ...
```

hybris Type System • Attributes

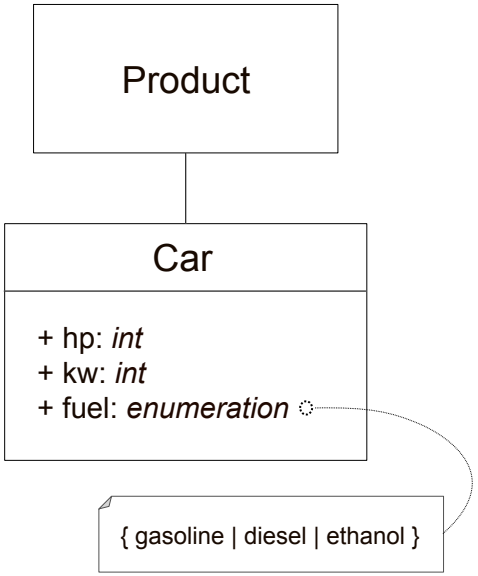





```
<items>
  <itemtypes>
    <itemtype code="Car" extends="Product" autocreate="true" generate="true">
      <attributes>
        <attribute qualifier="hp" type="java.lang.Integer">
          <description>Horse Power</description>
          <persistence type="property"/>
        </attribute>
      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

hybris Type System • Enumerated types

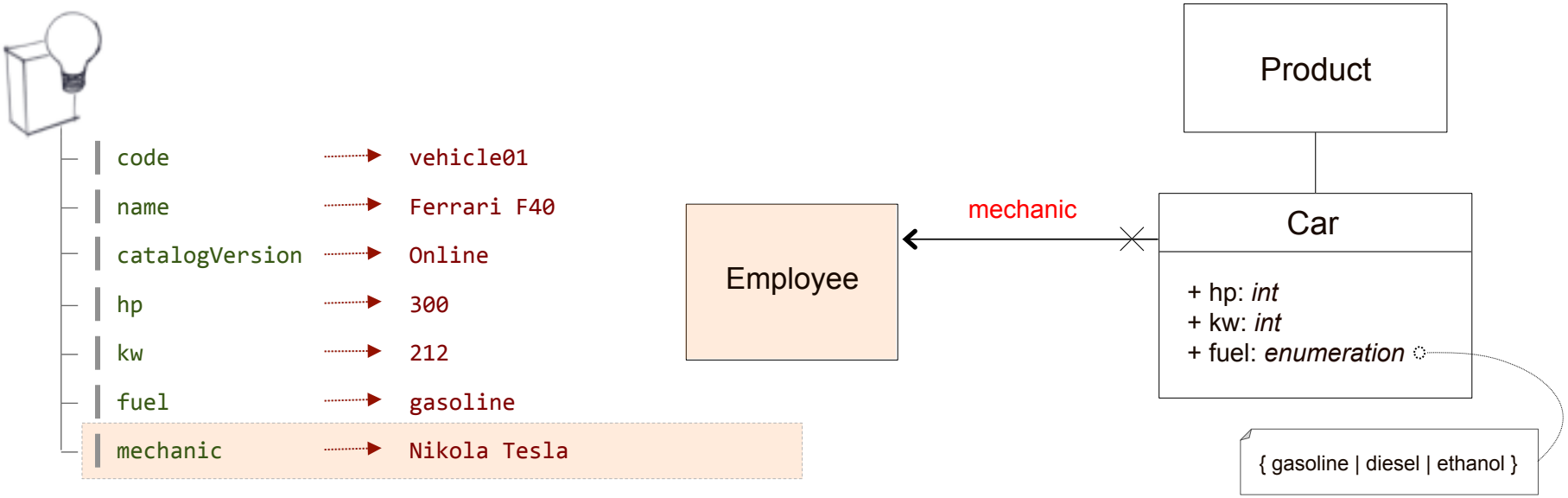
code	→	vehicle01
name	→	Ferrari F40
catalogVersion	→	Online
hp	→	300
kw	→	212
fuel	→	gasoline




```
<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">
          <description>Fuel for this car</description>
          <persistence type="property"></persistence>
        </attribute>
```

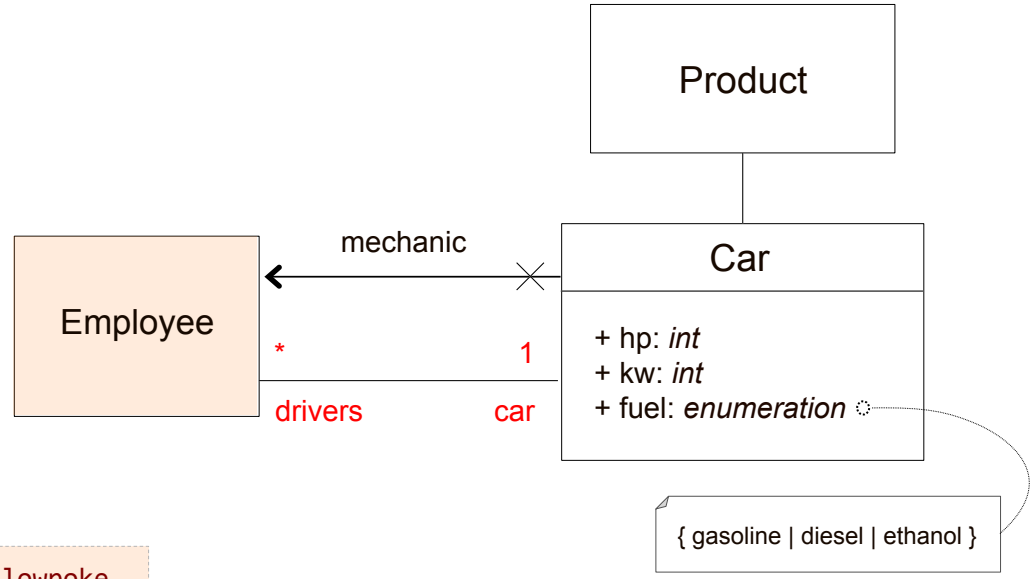
hybris Type System • Composed Type reference




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

hybris Type System • Relations

code	vehicle01
name	Ferrari F40
catalogVersion	Online
hp	300
kw	212
fuel	gasoline
mechanic	Nikola Tesla
drivers	Jacek Szybki, Drew Slowpoke




```
<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>
```

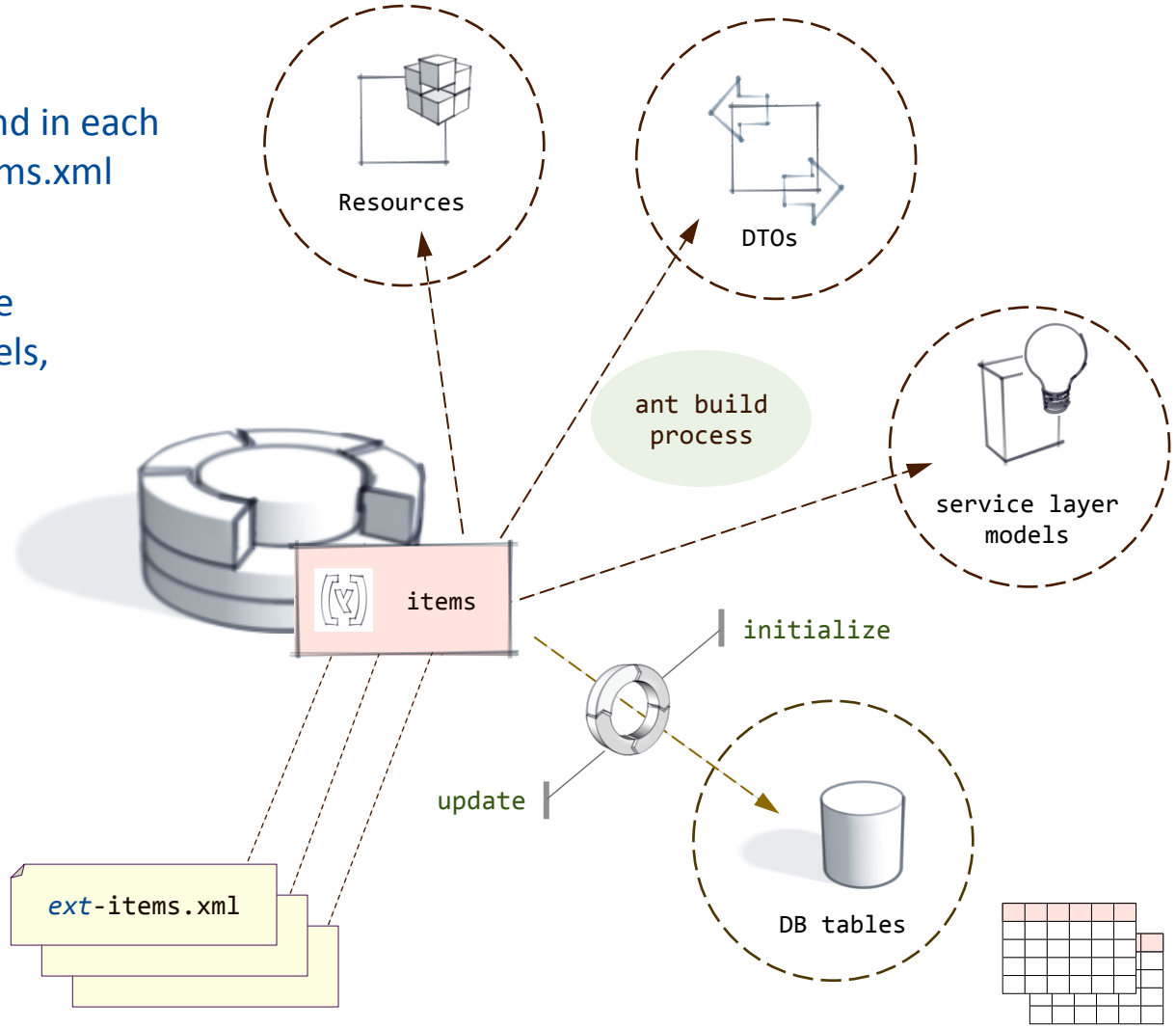
hybris Type System • Automatic Generation

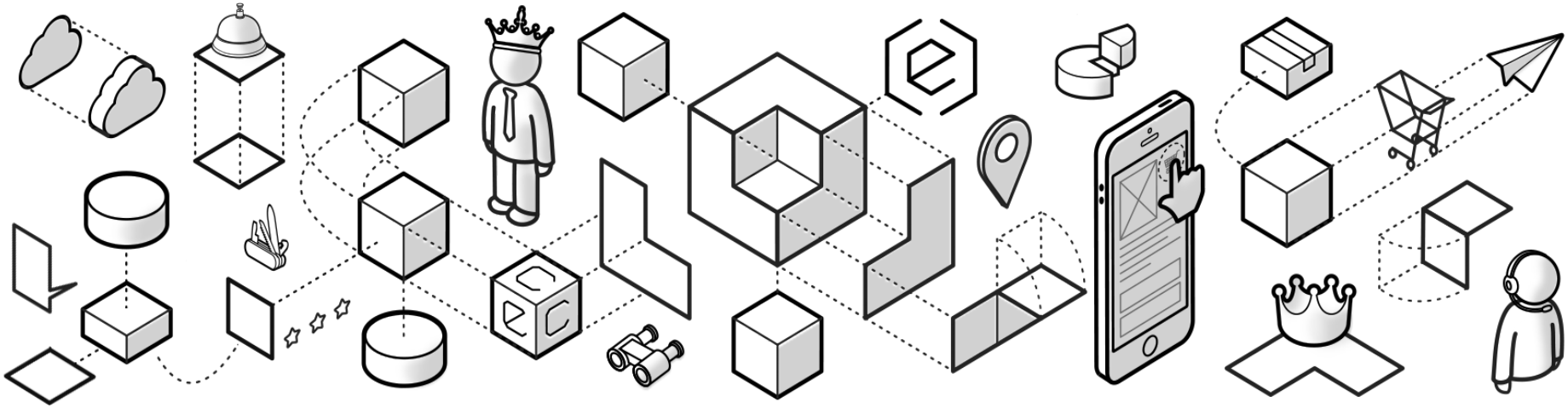
1. hybris item definitions are found in each extension's *extensionName*-items.xml

2. The ant process assembles type definitions and generates Models, DTOs, and Resources.

3. hybris also creates the required tables.

4. Invoking initialize or update creates the required table.





Collections and Relations

Collectiontypes

- 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

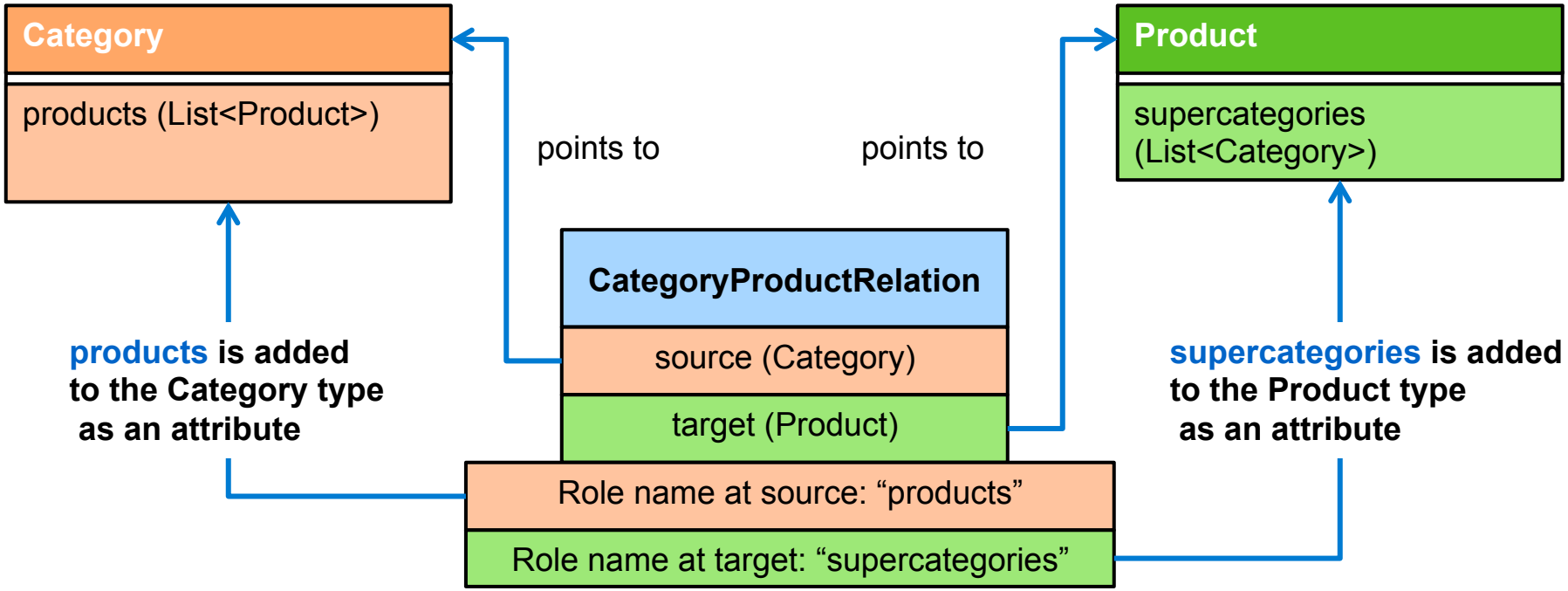
```
<collectiontype code="StringCollection"
                elementtype="java.lang.String" autocreate="true" />
<collectiontype code="LanguageCollection"
                 elementtype="Language" autocreate="true"/>

...

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

Relations

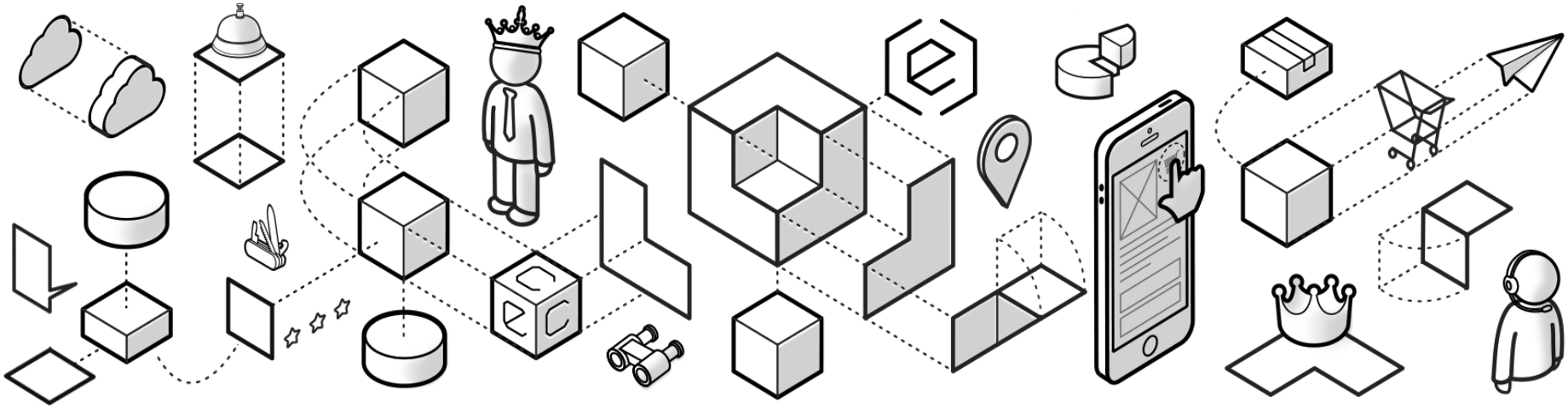
- One2Many and Many2Many
- Both sides are (can be) aware of the other



What's so Important About Relations?

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

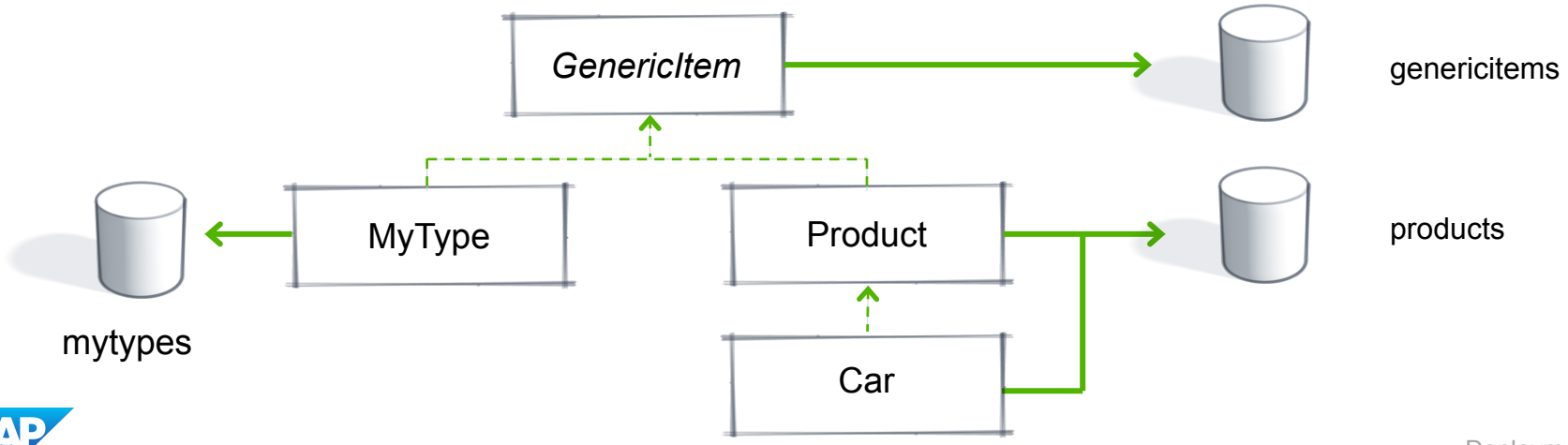
- Opposite side is not "aware" of the `CollectionType`
- `CollectionTypes` are stored in a database field as a comma-separated list of references (PKs) or atomic values
- Can cause overflow
- More difficult to search and generally lower performance



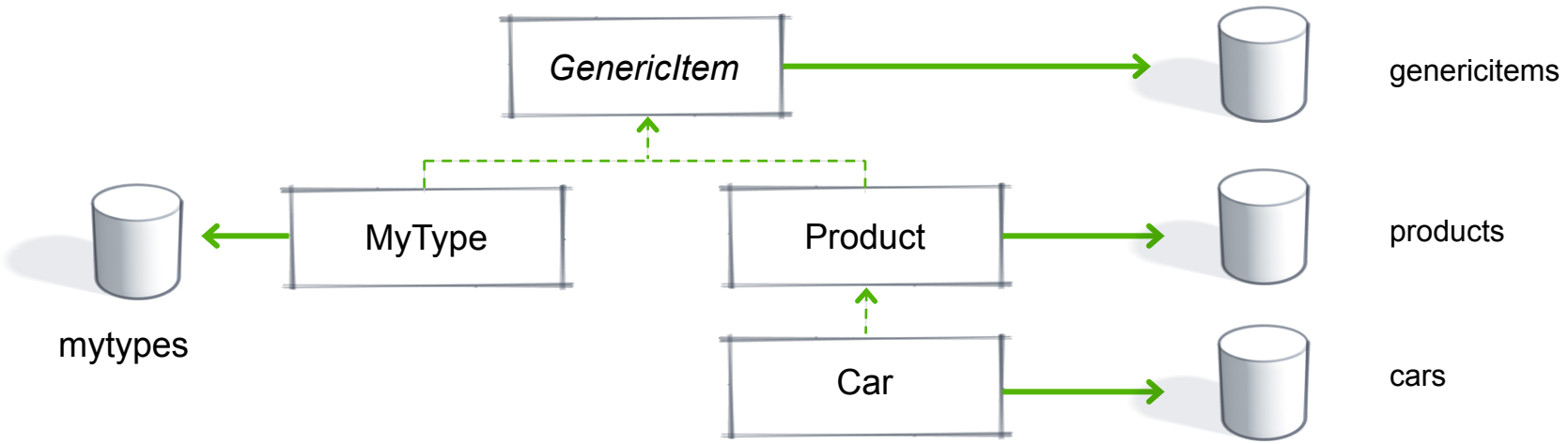
Deployment

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.
- hybris 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" …
```

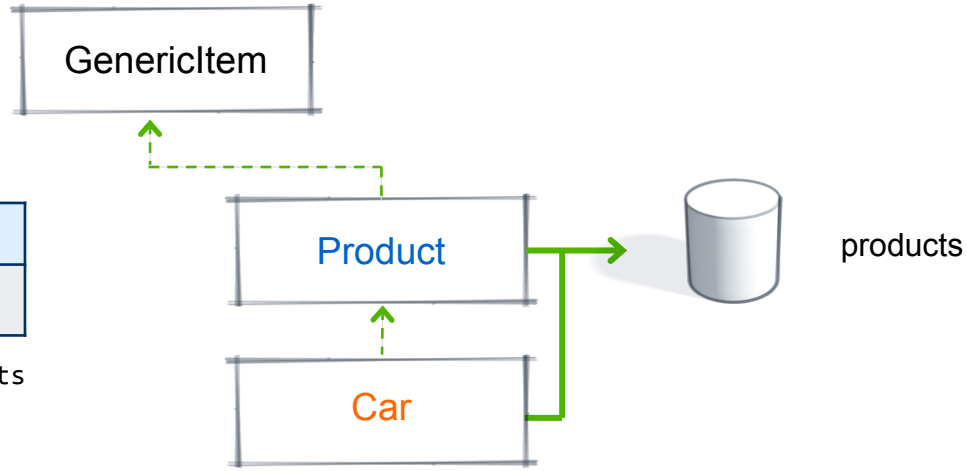
0 .. 10000 are reserved by hybris

10000 .. 32767 are free for use,
with *some exceptions*

O-R Mapping – Table Structure

PK	P1	P2	P3	C1	C2	C3
...	...	...	...	...	...	...

Products

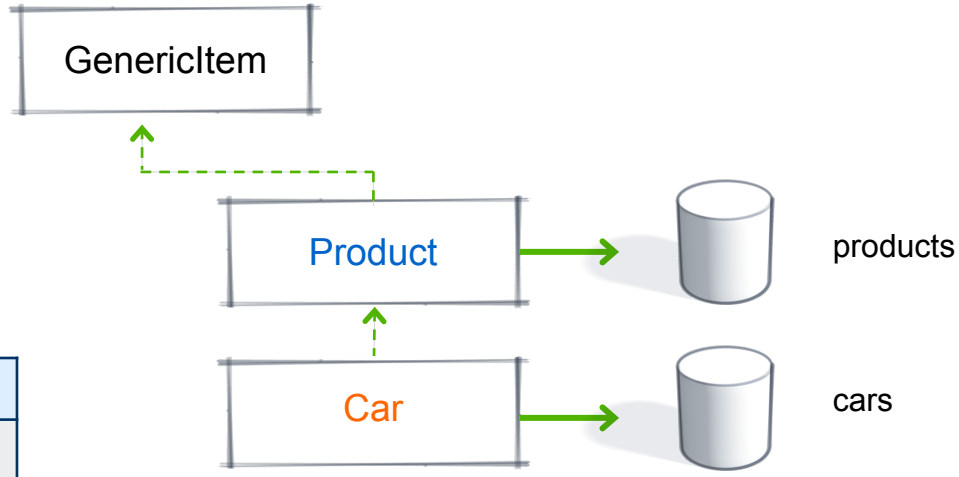


PK	P1	P2	P3
...	...	...	...

products

PK	P1	P2	P3	C1	C2	C3
...	...	...	...	...	...	...

cars



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	…

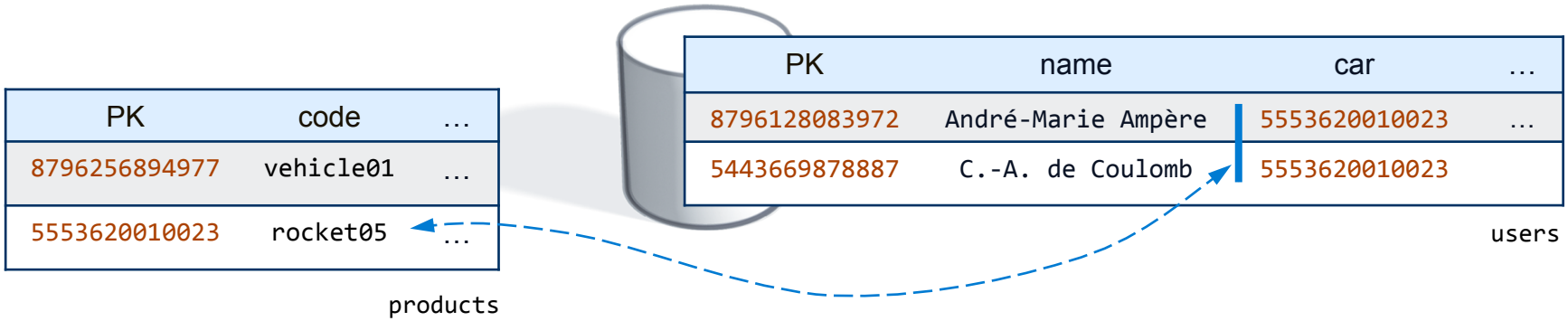
users

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

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

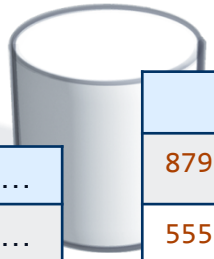


O-R Mapping – Deployment of Relations • 2

Many-to-Many

- New database table which holds the source and target PKs

```
<relation code="CategoryProductRelation" autocreate="true" generate="true"
          localized="false">
    <deployment table="Cat2ProdRel" typecode="143"/>
    <sourceElement qualifier="supercategories" type="Category" cardinality="many"
                   ordered="false">
        <modifiers read="true" write="true" search="true" optional="true"/>
    </sourceElement>
    <targetElement qualifier="products" type="Product" cardinality="many"
                   collectiontype="list" ordered="true">
        <modifiers read="true" write="true" search="true" optional="true"/>
    </targetElement>
</relation>
```



PK	code	…
8796256894977	vehicle01	…
5553620010023	rocket05	…

products

Source	Target
8796256894977	3776876789221
5553620010023	3776876789221
5553620010023	6152677365115

Cat2ProdRel

PK	code	…
3776876789221	Commuting	…
6152677365115	Exploration	…

categories

O-R Mapping – Deployment of Collections

Collections

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

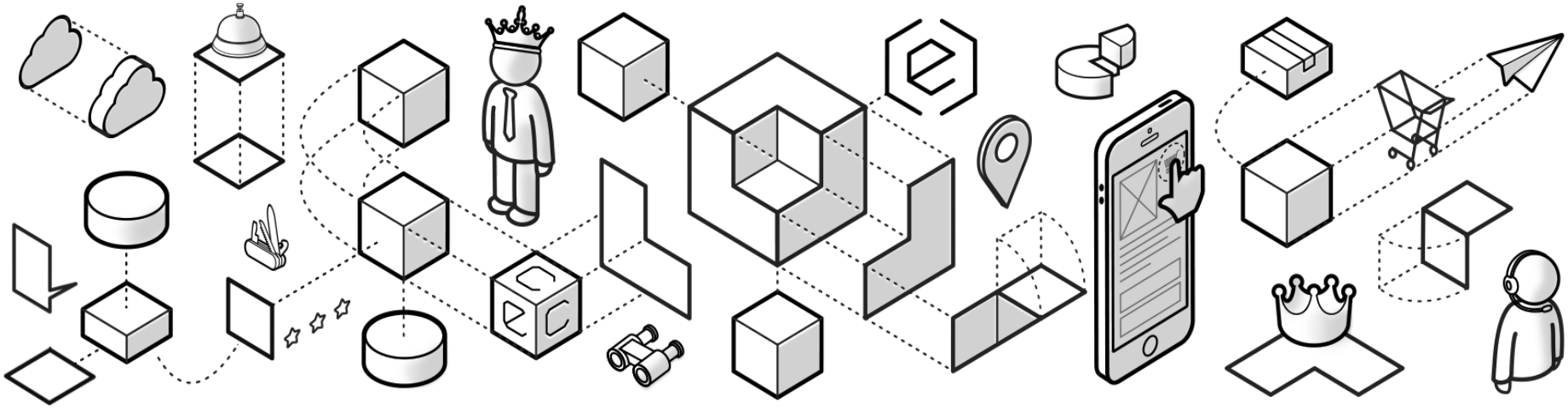
```
<collectiontype code="StringCollection"
                elementtype="java.lang.String" autocreate="true" />
<collectiontype code="LanguageCollection"
                elementtype="Language" autocreate="true" />

...

<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

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

Two Shades of Localization

- **Service layer provides support for i18n**
- **Leverage that to localize:**
  1. Types and their attributes
  2. Your data (Itemtype properties)

Type System Localization • Localizing type and attribute names

- The localization strings for types and attributes are stored in files named `extension-locales_XY.properties`
  - `extension`: the name of the extension
  - `XY`: the ISO code of the language
  - Properties convention:

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

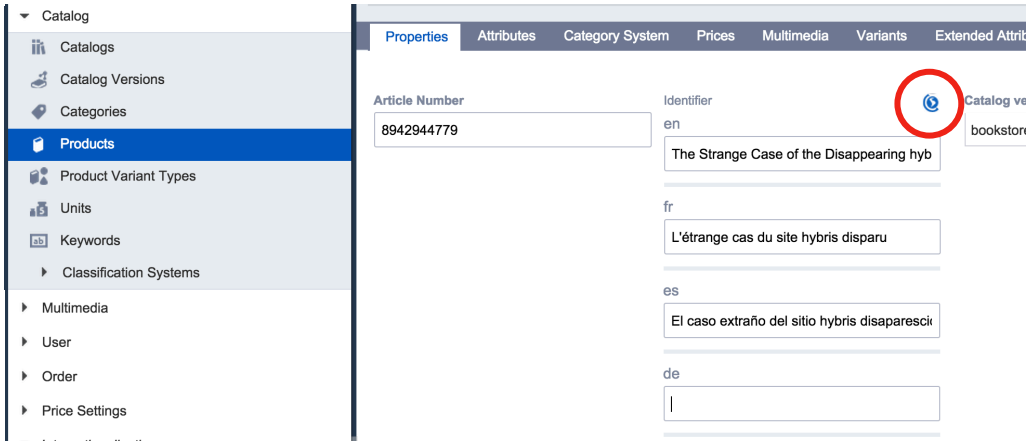
- Use generator from hMC to get all properties keys generated for selected extensions

Itemtype property localization • Localizing your data

- Any itemtype property may be localized

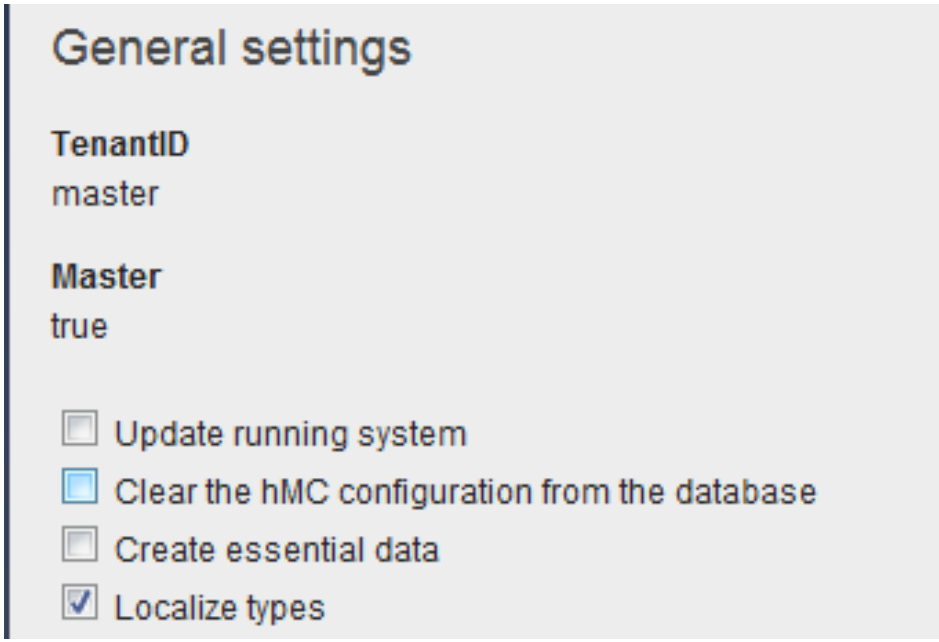
```
<itemtype code="Product">
   <attribute qualifier="barcode" type="java.lang.String" />
   <attribute qualifier="quote" type="localized:java.lang.String" />
   <attribute qualifier="mugshot" type="localized:Image" />
   ...
```

- Back office apps (hMC, cockpits, Backoffice) and ImpEx will allow input in multiple languages



Localize Types during System Initialization or Update

- Type localizations are stored in the database



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

Demo

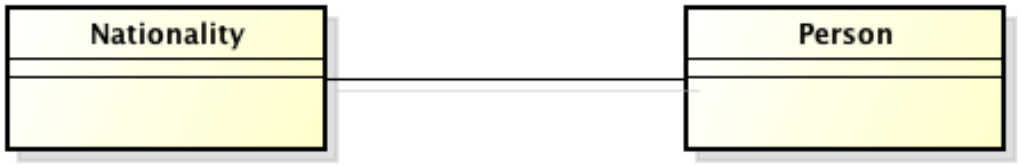
A UML Refreshing (1)

Visibility

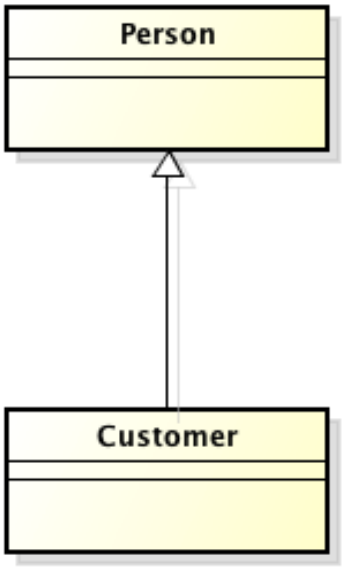
- **+** Public
- **-** Private

Relationships

- Association



- Generalization

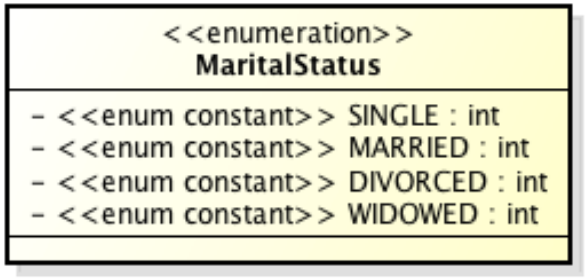


A UML Refreshing (2)

Multiplicity

- **0..1** No instances, or one instance
- **1** Exactly one instance
- **0..*** Zero or more instances
- **1..*** One or more instances

Stereotypes



Exercise 2

References

wiki.hybris.com/display/release5/Type+System+Documentation

wiki.hybris.com/display/release5/items.xml

wiki.hybris.com/display/release5/
Specifying+a+Deployment+for+hybris+Platform+Types