Q: How to Hybris HSQL DB Tables? = SQuirreL SQL Client

Step 1 = Download the Squirrel SQL Jar File from below link: -

http://sourceforge.net/projects/squirrel-sql/files/1-stable/3.7.1/squirrel-sql-3.7.1-standard.jar/download

Step 2 = D-Click / RMB on Jar file & Continue the installation by clicking → Next → Next…

Note: - Change default installation path (**C:\Program Files\squirrel-sql-3.7.1**) to "**C:\tools**" & finish installation

Step 3 = Download HSQL Driver from below link: -

https://sourceforge.net/projects/hsqldb/files/hsqldb/hsqldb_2_4/hsqldb-2.4.0.zip/download

Copy this downloaded zip (**hsqldb-2.4.0.zip)** & paste in "**C:\tools**".

Extract this ZIP file using 7-zip (or) WinZip.

Step 4 = Goto the extracted folder (**hsqldb-2.4.0\hsqldb-2.4.0\hsqldb\lib**) & **Copy** the **hsqldb.jar** & **Paste** in **C:\tools\lib**

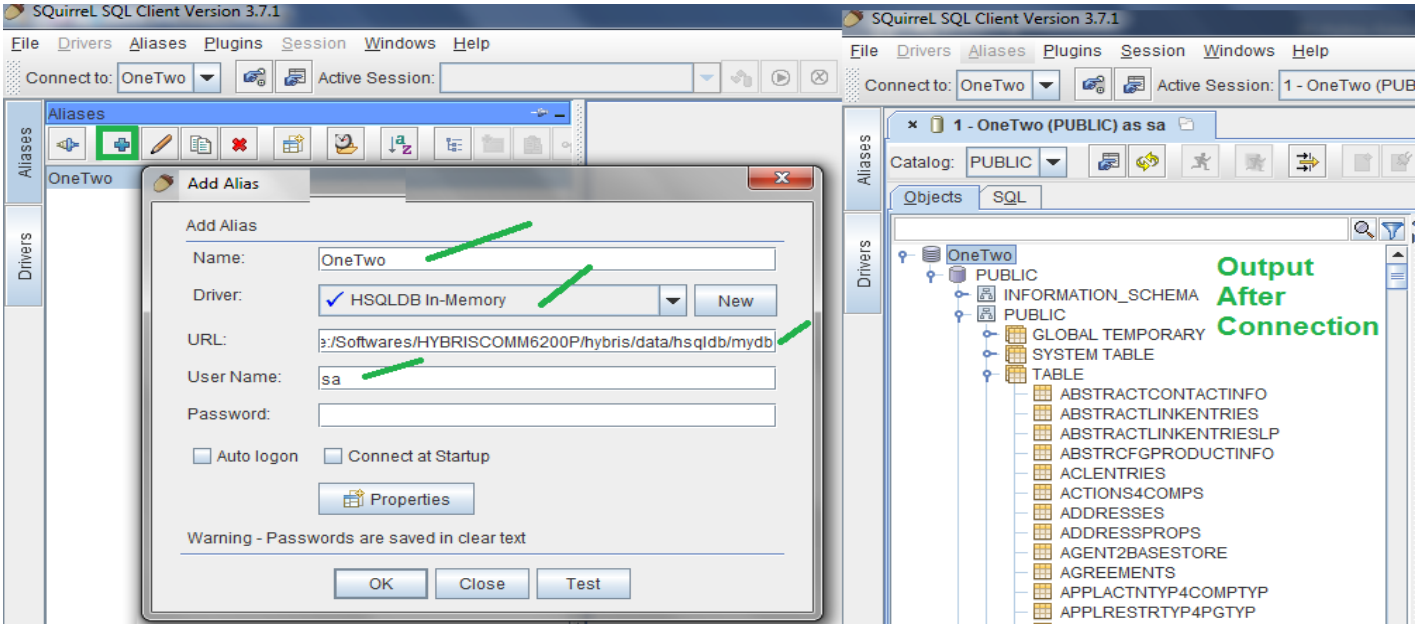
Step 5 = → **C:\tools** folder & then double click on **squirrel-sql.bat** file. This will open the UI (**Squirrel SQL Client**)

Click on **+** symbol & Enter the details on the below screen

Name = Whatever name you want (Ex -- OneTwo) & Driver = HSQLDB In-Memory

URL = jdbc:hsqldb:file:**/Softwares/HYBRISCOMM6200P/**hybris/data/hsqldb/mydb;readonly=true

User Name = sa → OK → Connect / Test



Step 6 = Next time, You can simply click on "Squirrel SQL Client" & Connect.

Expand HSQLDB (Created Connection) – PUBLIC --- PUBLIC – TABLES – Double click on any table to see the data

Explain Data model of Hybris?

- ✓ Hybris Data Modeling answer to **Questions** like: -

  (1) How Hybris is going to handle the DB. **(2)** How can we create new tables.

  (3) How can we create entire DB. **(4) …**

  Ans: - Hybris Data Modeling: -

  Helps you in developing your DB.

  Can take care of the DB Connections & DB Queries. And …

Hybris	Java	DBMS
Item Type	Java Class	Table
Attribute	Class Members (Variables)	Columns
Instance	Object	Row
Service Layer Classes	Methods	Functions (Nearly)
Model Classes	POJO Classes	---
Generic Item	Object Class	---
Extends (Keyword to inherit a table)	Extends (Keyword to inherit a class)	---
Atomic types	Primitive data types	
Deployment Table="TableName"	---	Create Table TableName
PK	---	Primary Key

Note: - Hybris also generates **POJO** classes for each defined **item type**. They are known as **model classes**. This class provides **getter** and **setter** method for attributes.

- ✓ **Data Modeling =** A process of creating DB Structure & analyzing it for meeting the project requirements.
- ✓ Hybris **service layer** provides all services for **DB handling**.
- ✓ Data Modeling in Hybris is designed (or) Created with the help of **Type System** in Hybris.

  Type System supports the following types & each one has its own significance (or) **items.xml** file contains 6 tags & those are is important (ACEMRI): -

  Note: - Not all of those tags are mandatory. But if you are having them, then order should be same.

  Q: If you change the order then build fails. Why?

  Ans: - That is the order defined in XSD. Who defined the XSD = Framework.

  - **(1) Atomic types** = Primitive types (Object, Number, Integer, Boolean, Byte, Double, String and…)
    Number, Boolean, String, Character, Date, Map, Class ... = All these extends to "**Object**".
    Integer, Byte, Double, Float, Long, BigInteger, BigDecimal ... = All these extends to "**Number**".
  - **(2) Collection types** = Represent group of element types (Like -- Group of Products / Customers)
    Scenario = One user can have multiple orders. How to design the tables?

User Table (Without Order List)

User ID	User Name
101	One
102	Two
103	Three
104	Four

Order Table

Order ID	Order Name	User ID
O101	ABC	101
O102	BCD	101
O103	CDE	102
O104	DEF	101
O105	EFG	103
O106	FGH	102

User Table (With Order List)

User ID	User Name	Order List
101	One	O101, O102, O104
102	Two	O103, O106
103	Three	O105
104	Four	

Note: - If we don't have "User Table (With Order List) then "How to get Order Count of each user?

Ans = select count (Order.OrderID), User.UserID from Order, User where User.UserID = Order.UserID group by Order.UserID

Disadvantage with above = It will read all the records in Order table (Order table might have Lacks records). Which is performance challenging.

Q: Tell me the logic which will give Orders count of the user without scanning / reading entire Orders table?

Ans = Create User Table with Oder List & do below: -

select OList from User where UserID = 101;

Q: What will be the datatype of OList in Hybris? = collectiontype

Q: What is the default collection in Java = Collection.

Hence we need to specify this Collection as list / set.

List = Duplicates Allowed & Insertion Order is followed.

Set= Duplicates not allowed & Insertion Order depends on Implementation

We can create **set** collection / **list** collection / **media** collection / ...

Note: - Too much data normalization is not good. Bcoz joining is difficult & decrease performance.

```
<collectiontype code="ProductCollection" elementtype="Product" autocreate="true" generate="true"/>
<collectiontype code="LanguageList" elementtype="Language" autocreate="true" generate="true" type="list"/>
<collectiontype code="LanguageSet" elementtype="Language" autocreate="true" generate="true" type="list"/>
```

- **(3) Enum types** = Used for preparing particular set of values (Example: - Days in Week)

Create a **column** which will accept only fixed set of values (Gender Column which accept M / F).

Oracle = create table ABC (Gender Varchar(10) check (gender in ('F','M'));

Similar, if you want to create similar in frontend, then in hybris we have – **enumtypes.**

```
<enumtype code="CreditCardType" autocreate="true" generate="true">
    <value code="amex"/>    <value code="master"/>
    <value code="visa"/>    <value code="diners"/>        </enumtype>
```

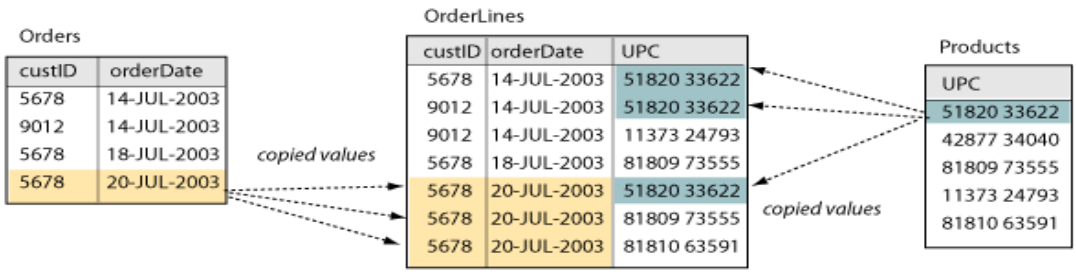
- **(4) Map types** = Used to store key value Paris. Each key will have it's own value.

If you want to create a column which accepts **key & values** pairs (Like Map) then in hybris we have maptypes. Generally, **we don't use** maptypes. Bcoz we don't have **column** which store key, values pairs. argumenttype = key & returntype = value

Example = Localized fields require separate value for each language.

```
<maptype code="localized:java.lang.String" argumenttype="Language"
returntype="java.lang.String" autocreate="true" generate="false"/>
For Example:
<attribute autocreate="true" qualifier="name" type="localized:java.lang.String">
    <modifiers read="true" write="true" search="true" optional="true"/>
</attribute>
```

- **(5) Relation Types** = There are 3 types of relations (1 – 1, 1 – N & M– N).



- Each **Order** is associated with **one or more** OrderLines
- Each **OrderLine** is associated with one and **only one Order**.

- Each **OrderLine** is associated with one and **only one Product**.
- 1 - 1 ---> Q: Min No. Tables required = 1
- 1 - N ---> Q: Min No. Tables required = 2
- M - N ---> Q: Min No. Tables required = 3
- Relationship also called as **cardinality**.
- In relations, we will have **sourceElement** & **targetElement** all the times.

Creating Relation between the tables (Country & Regions are linked together):-

```
<relation code="Country2RegionRelation" generate="true" localized="false" autocreate="true">
   <sourceElement type="Country" qualifier="country" cardinality="one">
      <modifiers read="true" write="true" search="true" optional="false" unique="true"/>
   </sourceElement>
   <targetElement type="Region" qualifier="regions" cardinality="many">
            <modifiers read="true" write="true" search="true" partof="true"/>
   </targetElement>    </relation>
```

- **(6) Item types** = Creating Tables (or) Updating exiting table

  Any application works on some kind of data (customers, products, promotions etc., or Meta data). These we defined as Item type in Hybris.

  In Hybris, Item types are basically XML declaration. They are defined in **items.xml**.

  Items.xml is located in **resources** directory of any extension.

  Item Type = Table + Java Classes (Contains Setters & Getters).

  Table structure is configured over this type with the help of attributes.

  Each attribute in "**Item type**" definition will represent a **column** in the table.

```
<itemtype code="Customer" extends="User" jaloclass="de.hybris.platform.jalo.user.Customer"
 autocreate="true" generate="true">
   <attributes>
      <attribute autocreate="true" qualifier="customerID" type="java.lang.String">
         <modifiers read="true" write="true" search="true" optional="true"/>
         <persistence type="property"/>
      </attribute>    </attributes>    </itemtype>
```

- **Items.xml =** It is a File used for data modeling.

Note: - Explain what is Deployment (or) How to create separate table?

- ✓ In Hybris, table deployment in DB is defined by **deployment tag**.
- ✓ If we add a deployment tag to "**Item type definition**", then it creates **separate table**.
- ✓ If we don't add deployment tag to it, it will deploy "**Item type**" in parent table.
- ✓ Hybris supports Updating existing table & Creating own tables.

```
<deployment table="Products" typecode="1" />
```

- ✓ **What are the Hybris Conventions for Deployment Codes (Must be +Ve & Range = 0 – 32767): -**
  - (1) **commons** extension (132XX)
  - (2) **print** extension (234XX, 239XX)
  - (3) **processengine** extension (327XX)
  - (4) **Legacy** xpring extension (244XX, 245XX)
  - (5) **b2bcommerce** extension (100XX)

Note: - Typecode values between **0 and 10000** are reserved for **hybris-internal use**. So, Typecode values > 10000 are free for you to use.

Q: What happens if we don't specify deployment table?

Ans: - It will go & check what is base itemtype and so on.....then finally attributes stored in **GenericItem** (Bcoz GenericItem is the Base itemtype).

This is not best option. Hence, have separate deployment tables every time.

In Hybris -- There won't be separate DB Designers.

We specify everything in XML, Hybris internally creates required tables.

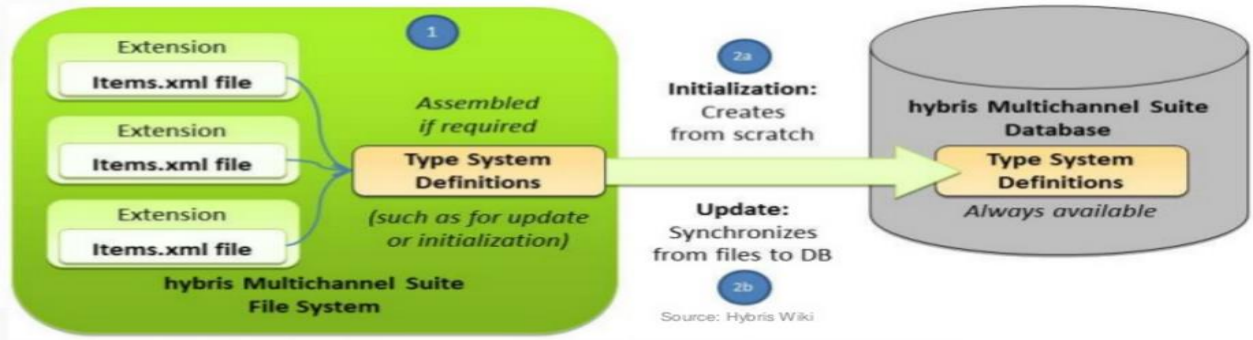
Whatever name given for **Code** is used in **Hybris** & Whatever **name** given for table is used in **DB** level.

```
<itemtype code="UserRight"  extends="GenericItem"   jaloclass="de.hybris.platform.jalo.security.UserRight"
          autocreate="true"    generate="true">
    <deployment table="UserRights" typecode="29"/>
```

Deployment table = "UserRights" --- Means, In DB **UserRights** table will be created.

Note: - itemtype code name & deployment table name can be same.

Q: How does data model works?



Q: Explain Important Features of the Hybris Data Models?

- ✓ **Multi DB Support: -**

```
<attribute qualifier="XXXvalueCenter" type="java.lang.String">
    <description>XXX Value Center</description>
    <modifiers read="true" write="true" optional="true" />
    <persistence type="property">
        <columntype database="oracle"> <value> VARCHAR2(20) </value> </columntype>
        <columntype database="mysql"> <value> VARCHAR(30) </value> </columntype>
        <columntype database="sqlserver"> <value> NVARCHAR(40) </value> </columntype>
    </persistence>
</attribute>
```

- ✓ **Dynamic Attributes**

```
<persistence type="dynamic" attributeHandler="dynamicAttributesProdCatTimeRanges"/>
implements DynamicAttributeHandler<attributeType, ItemType>
```

- ✓ **Dynamic Enums**

```
<enumtype code="CreditCardType" autocreate="true"  generate="true" dynamic="true">
    <value code="amex"/>  <value code="visa"/>  <value code="master"/>
    <value code="diners"/>  </enumtype>
```

- ✓ **Multi-Tenant Support** = Tenants allow customizing: - currency, currency format, data format.

Q: Explain *-i**tems.xml?**

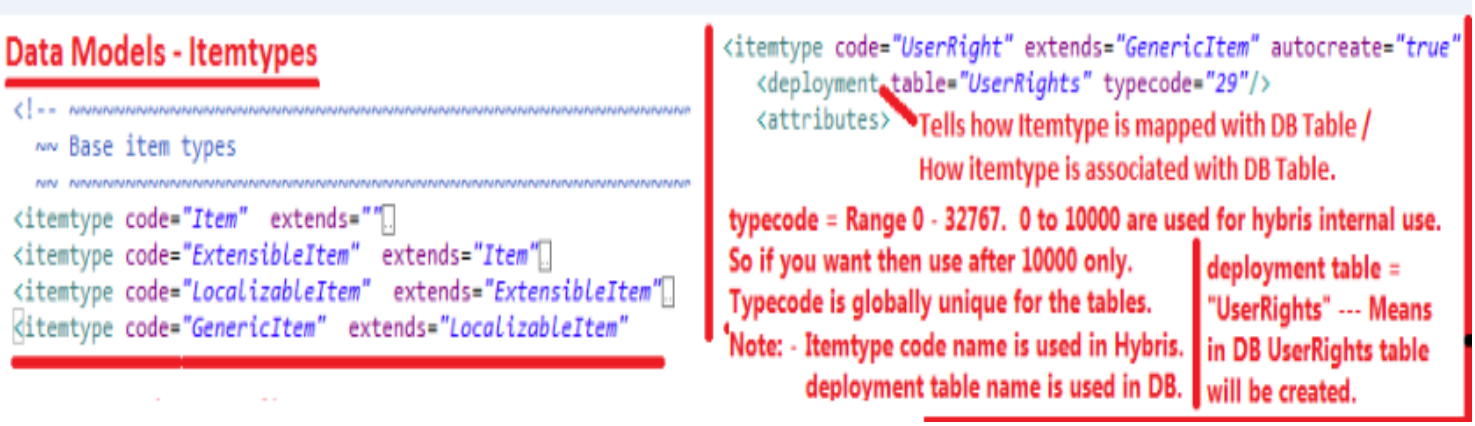
- ✓ **Q: How to create table in Oracle?**
  CREATE TABLE Persons (PID char(4), LNme varchar(255), FName varchar(255), Address varchar(255));
  Note: - MySQL Data Types = BIT, DECIMAL, ENUM, INT, LongText and ...
  Oracle Data Types = RAW, FLOAT(24), VARCHAR2, Number(10,0), Clob, Blob and ..
  So, data types are very much important. Bcoz data types are changed from 1 DB to another DB.
  Q: How do you manage if data types are different from 1 DB to another DB?
  Ans: - Write separate script for each DB. This is not good practice.
  Q: What is the Solution then?
  Ans: - Write "**Items.xml**" -- It Contains the logic to create table & structure of table.
  Hybris internally will create tables for you based on the DB from **items.xml** file.
  General Syntax = **extensionname-items.xml** (Eg., \chennatrainingpayment-items.xml)
  This file is not mandatory in all extensions.
  Only some extensions will have Items.xml
  Eg., → Eclipse, Press CTRL + SHIFT + R, Search = *training*items, You can see only 3 xml.
  Note: - We have **2 important** xml files for Learning: -
  1) **core-items.xml** = We can't modify this as it's available in **platform**. To learn use this.
  2) **catalog-items.xml** = We can't modify this as it's available in **platform**. To learn use this.
  Note: - These are available in ...\hybris\bin\platform\ext\core\resources**core-items.xml**
  Q: To write XML, what we need to have / What is base? = **XSD** (XML Schema Definition).
  Hence to write **items.xml**, we need to have **items.xsd.**

- ✓ Data model of Hybris are defined **declaratively** in **items.xml** (Used define data structure in project).
- ✓ Creates new data model from scratch as defined in "**items.xml**".
- ✓ The data model of extension is defined in **<extension-name>-items.xml** file
- ✓ Below are few important points about **items.xml**: -
  - ✓ The item type in itself is of type Item.
  - ✓ To define an **items.xml** file there must be an **xsd** file which has some rules.
  - ✓ **Instance** of item type stored is **item type table**, but the **item type definition** is stored in **composedType**. (Eg: - Student item type, instances of Student will be stored in Student table, but the definition of Student will be stored in composedType table).
  - ✓ **GenericItem** is the default parent of each **item type**.
    In **Java** super class for all classes is **Object** class.
    Q: Why Object class is the super class for all the classes.
    Ans: - To reduce the code duplication. Basic methods that every class should contain and they are very basic features or functionalities that. Every class should have and support (like getting Class name, String representation of Object, finalize, notify, equals etc...).
- ✓ We can **extend** one item type using keyword (Simpler to Java inheritance using extends).

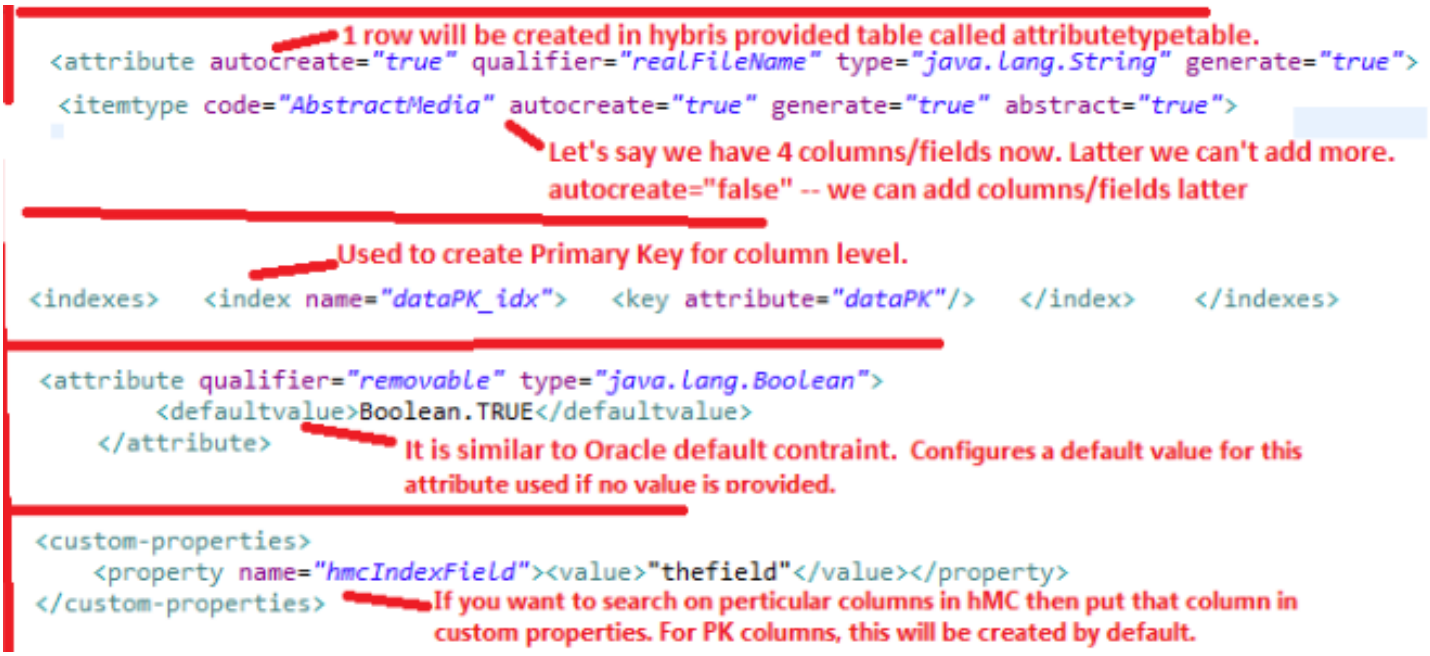
- ✓ **Note: -** In Hybris for us, "**GenericItem**" is the **super item** for all the items. But Internally for hybris always **Item** is super item for everything.
  If we create **new item type** & **not** mentioned the **extends**, then by default it will extends "**GenericItem**".
  Note: - Base item types: -
  Item = <itemtype code="Item" extends="">
  ExtensibleItem = <itemtype code="ExtensibleItem" extends="Item">
  LocalizableItem = <itemtype code="LocalizableItem" extends="ExtensibleItem">
  GenericItem = <itemtype code="GenericItem" extends="LocalizableItem"

- ✓ We can control the access to an attribute using **modifiers (or)** What are **default Modifiers**: -
  - ✓ **Example: -** <modifiers read= "true" write= "false" search= "true" initial= "true" optional= "false" unique="true" private = "false" private = "false"/>
    - **read** – If false, we cannot access it from our java program.
    - **write** – If false, we cannot modify the value.
    - **optional** – If false, it is mandatory to initialize this attribute. Since it is not optional.
    - **unique** – If true, attribute must hold a unique value. Similar to unique constraints in DBMS.
    - **search** – If true, the attribute is searchable through queries.

- ✓ We can **control**, how the tables are created for an item type. **Default** table for each item type is **GenericItem**. The table is called as deployment in Hybris. So GenericItem is the default deployment.
- ✓ <deployment table="UserRights" typecode="29"/>
  - ✓ Table name is UserRights. Type code (+Ve Integer) internally used by Hybris to generate primary keys.
  - ✓ If we **don't give deployment** tag for item type, instances of that item type will store in **GenericItem** table. This will cause GenericItem table to expand drastically, which is **not good for DB performance**.
  - ✓ **Persistence** is one of the property of item type attribute. If this property is **set to property**, it means that attribute **will be stored in database**. If it is **set to dynamic**, the attribute will not be part of database table. It is just **calculated** at run time. This is similar to calculated column in DBMS.



Note: - generate="true"/"false" will be there for all types. But only "enumtype & itemtype" will have meaning. So, It will create *.java file for enum & itemtype
<atomictype class="java.lang.Object" generate="false"/> -- For atomictype / collectiontype / ... does not matter even we specify generate = "true / false".



Q: Explain Dynamic Attributes? =

We need a column in Java level, but don't need in DB level. When we go with this scenario?

Ans: - Sometimes, you may have fields that are derived from other fields, and should only be done programmatically, rather than having the state be persisted via **serialization**.

The **transient** keyword in java is used on class attributes / variables to indicate that serialization process of such class should ignore such variables. A **transient variable** is a variable that **cannot be serialized**.

Hybris supports **transient** variables. Whenever you create attribute, we will have persistence option: -

```
<attribute autocreate="true" qualifier="taxValues" type="TaxValueCollection" generate="true">
    <persistence type="dynamic"/>
    <modifiers read="true" write="true" search="false" optional="true"/>
</attribute>
```

Note: - Persistence tells whether you want to save the data / not.

If you put **persistence = property** → Means you can save the data into that column.

If you put **persistence = dynamic** → Means you will not able save data into that column.

Advantage with Dynamic Attributes = (1) Data **will not** be saved in **DB** as its dynamic. (2) The **custom logic** is written once and **used** all the time wherever that attribute is required.

Q: Explain generate = "true"?

Generally, we will be having 3 layers = Presentation Layer (PL) , Business Layer (BL), Database Layer (DL).

PL = We have registration form. **Q:** In Java -- How we save data in DB whatever entered in PL registration form?

Ans: - From BL to DL → How will you store data = ORM / Hibernate.

Q: Who write hibernate class / Entity class (Which is equal to DB table)? = Developer

But in Hybris You no need to write **Entry** class. **DB table** & **Entity class** will be created based "**items.xml**" by Hybris automatically.

Entity class in hybris = Model class / Model.java

For **Product** table, we will have having **ProductModel.java**

```
<itemtype code="Product"   extends="GenericItem"      jaloclass="de.hybris.platform.jalo.product.Product"
          autocreate="true"    generate="true">
    <deployment table="Products" typecode="1" propertytable="ProductProps"/>
    <attributes>
        <attribute autocreate="true" qualifier="code" type="java.lang.String" generate="true">
```

a) Abstract Jalo layer classes, starting with Generated (GeneratedXXX.java)

b) Non abstract Jalo layer classes, with the same name as the item type (XXX.java)

generate ="false" means it won't generate the above 2 classes.

XXXModel.java will generate automatically. It doesn't matter whether you specify generate=true/false.. In attribute level... If you specify **generate = "true"** then it generates setter & getter methods.

Q: When can we have **only setter** method (No getter method)?

Ans: - Example PWD filed.

```
<modifiers read="true" write="true" search="false" optional="true" initial ="true" unique="true"/>
    <custom-properties>    <property name="hiddenForUI">
                               <value>Boolean.TRUE</value>  </property>   </custom-properties>
```

In modifiers level... If you give **read = "true"**, Means it will only create getter method (no setter method).

If you give **write = "true"**, Means it will only create setter method.

Q: In what cases, we will have **only getter** methods (No setter methods).

Ans: - Connection properties. (OR) Creation Time Column (It's only once). Latter it will not be modified.

Q: What is the alternate if we don't have the setter method? = CTOR Injection.

Q: How to make a field as mandatory? = optional = "false"

Note: - If you want to return your column as part of Flexible Search then **search = "true"**.

Q: How to give **value while creation** of the row itself. It can be changed only once. That is 1st time itself. Attribute is writable only at initialization time? **Ans: -** initial="true"

Note: - Generally, it is used with the combination of **write = "false"**. Because It can change only once.

Q: How to make particular column / attribute as primary key? → **unique = "true"**

Q: How to create composite Primary Key?

Ans: - Put **unique = "true"** for whatever columns you want. Internally it creates the Composite key.

Note: - <custom-properties> If you want to search on **particular columns in hMC** then put that column in custom properties. For PK columns, this will be created by default.

Q: Let's say, you have a table in which we are storing some intermediate values. In this case POJO class is not required. If you **don't want POJO** class to be generated for the attributes then make **generate = "false"**.

```
<itemtype code="TestEmployee"  autocreate="true"  extends="Employee"  generate="false"/>
```

Note: - The concept of generate = "true" will be there throughout all the types.

Note: - <atomictype class="java.lang.Object" autocreate="true" generate="false"/> → For **atomictype**, even we have generate = "true" / "false" does not matter.

<collectiontype code="ExampleCollection" generate="false"/> → For **collectiontype**, even we have generate = "true" / "false" does not matter. No Meaning.

Only "**enumtype & itemtype**" only will affect with generate = "true" / "false". It will create *.java files. Rest all the types it won't create *.java file.

autocreate = "true / false" =

1) itemtype level = If **autocreate = true,** means item will be created during **initialization**. Default is **'true'**.

Let's say, we have 4 columns now in itemtype & latter you want to add some more columns in itemtype, then make autocreate="false".

2) attribute level = If **autocreate = true**, attribute descriptor created during initialization. Default is **'true'**.

Note: - By default, there are some tables with in hybris named as "atomictype, collectiontype, itemtype...". When you make **autocreate = "true"** then in this tables there will be row created.

Q: Why to create Index? = Indexes are created for faster search.

Oracle Syntax = CREATE INDEX **PIndex** ON Persons (LName);

Hybris Syntax =

```
<indexes>  <index name="qualifierIDX" unique="true">
               <key attribute="qualifier"/>
           </index>    </indexes>
```

Note: - At the end of all the attributes, you can add this.

Q: attribute **defaultValue** =

```
<attribute autocreate="true" qualifier="autocreate" type="java.lang.Boolean">
    <defaultvalue>      java.lang.Boolean.TRUE    </defaultvalue>
```

It is similar to default **constraint** values in **Oracle**. = CREATE TABLE Persons(DEFAULT 'Chenna');

At the time creation of table, you can mention the default value.

Example: - When you create customer, add him to customergroup by default.

Q: itemtype: -

```
<itemtype code="Item"  extends=""  jaloclass="de.hybris.platform.jalo.Item"
   deployment="de.hybris.platform.persistence.Item" autocreate="true"  generate="false"  abstract="true">
```

1) code = "Item" = Item table is will be created. This is used in Hybris level.

2) extends = " / "ABC" / ... → Defines the class, which will be extended. Default is **'GenericItem'**.

3) jaloclass = "XXX" = Specifies the name of the associated jalo class. Default is [extension-root-package].jalo.[type-code] which will be generated if not existent.

4) deployment = "XXX" = It is Deprecated. **Use separate** deployment sub tag. All instances of this type will be stored in a separated database table (<deployment table="MediaFolders" typecode="54"/>).

5) autocreate="true" = If **'true'**, the item will be created during initialization. Default is **'true'**.

Let's say, we have 4 columns now in itemtype & latter you want to add some more columns in itemtype, then make autocreate="false".

6) generate="false" = If **'true'**, the source code for this item will be created. Default is **'true'**.

Let's say, U have a table in which we are storing some intermediate values. In this case POJO class is not required.

If you don't want POJO class to be generated for the attributes then make "generate = "**false**""

7) abstract="true" = Marks type & jalo class as abstract. If **'true'**, the type can't be instantiated. Default is **'false'**.

Q: attribute: -

```
<attribute autocreate="true" qualifier="totalDiscounts" type="java.lang.Double" generate="true">
    <persistence type="property">
        <columntype>  <value>  java.math.BigDecimal  </value>  </columntype>  </persistence>
    <modifiers read="true" write="true" search="true" optional="true" unique="true" initial="true"/>
    <defaultvalue>  Double.valueOf(0.0d)  </defaultvalue>  </attribute>
```

1) autocreate = If **'true'**, the attribute descriptor will be created during initialization. Default is **'true'**.

Note: - By default, there are some tables with in hybris named as "atomictype, collectiontype, itemtype...".

When you make autocreate = "**true**" then in this tables there will be row created.

2) qualifier = In attribute, qualifier is the Column / **Attribute name**. Means, column name = totalDiscounts.

Qualifier of this attribute. Attribute qualifiers must be unique across a single type.

3) read = true & write = "true"

If you specify read = "true", Means it will only create getter method (no setter method).

If you specify write = "true", Means it will only create setter method.

4) search = "true" → If you want to return your column as part of Flexible Search then search = "true".

Defines if this attribute is searchable by a FlexibleSearch. Default is **'true'**.

Attributes with persistence type set to **'jalo'** can't be searchable.

5) optional = "false" → It makes a field as mandatory.

Defines if this attribute is mandatory or optional. Default is **'true'** for optional. Set to **'false'** for mandatory.

6) <persistence> = Persistence tells whether you want to save the data / not.

If you put persistence = property → Means you can save the data into that column.

If you put persistence = dynamic → Means you will not able save data into that column.

Defines how the values of the attribute will be stored.

Possible values: 'cmp' (deprecated), 'jalo' (not persistent, deprecated)

'property' (persistent), 'dynamic' (not persisted).

7) generate = "true" → If **'true'**, getter and setter methods for this attribute will be generated during a hybris Suite build. Default is 'true'.

Note: - The concept of generate = "true" will be there throughout all the types.

<atomictype class="java.lang.Object" autocreate="true" generate="false"/> → For atomictype, even we have generate = "true" / "false" does not matter.

<collectiontype code="ExampleCollection" generate="false"/> → For collectiontype, even we have generate = "true" / "false" does not matter. No Meaning.

Only "enumtype & itemtype" only will affect with generate = "true" / "false". It will create *.java files. Rest all the types it won't create *.java file.

8) unique = "true" = It creates primary key column if unique = "true".

If 'true', the value of this attribute has to be unique within all instances of this type.

If there are multiple attributes marked as unique, then their combined values must be unique.

Q: How to create composite Primary Key? = Put **unique = "true"** for whatever columns you want. Internally it creates the Composite key.

9) initial = "true" → We can give value while creation of the row itself. It can be **changed only once**. That is 1st time itself. Attribute is writable only at initialization time?

If **'true'**, the attribute will only be writable during the item creation.

Setting this to **'true'** is only useful in combination with **write='false'**. Default is **'false'**.

10) defaultvalue = Configures a default value for this attribute used if no value is provided.

The default value is calculated by **initialization** and will not be **re-calculated** by **runtime**.

It is similar to oracle default constraint.

Configure a default value if you don't change / override latter.

Q: What are the ways to define "Itemtypes" in Hybris?

- ✓ Define the **new item** type **without extending** any existing item type
- ✓ Define the **new item** type **by extending** it with existing item type
- ✓ Define the **existing item** type again with **new attributes**

Q: When we should define deployment mandatorily?

```
<itemtype code="Unit"                          1) Defining new item type by
        extends="GenericItem"                  extending GenericItem
        jaloclass="de.hybris.platform.jalo.product.Unit"
        autocreate="true"
        generate="true">
    <deployment table="Units" typecode="10"/>
```

```
<itemtype code="AbstractOrder"  extends="GenericItem"
        jaloclass="de.hybris.platform.jalo.order.AbstractOrder"
        autocreate="true"   (2) Defining new item type by
        generate="true"     extending existing item type for
        abstract="true">    which there is no deployment
```

```
<itemtype code="Cart"   extends="AbstractOrder"
        jaloclass="de.hybris.platform.jalo.order.Cart"
        autocreate="true"
        generate="true">
    <deployment table="Carts" typecode="43"/>
```

Q: How collection will be stored in DB?

- ✓ **Collection of Atomic type =** If **collection** contains **atomic types** then values stored as **binary fields** in DB.
  <attribute qualifier="nickNames" type="StringCollection">
- ✓ **Collection of Non atomic/composed type =** If **collection** contains **complex types** then those items PKs are stored as **one single String** separated by comma for each PK in the **database (Example: -** Languages**)**.
  <attribute qualifier="languages" type="LanguageList">

Q: Explain Dynamic in Enum?

- ✓ It is different from **Dynamic attributes**. If an **Enum type is dynamic** then values **added at the run time**.
- ✓ We can make Enum type as Dynamic by specifying **dynamic=true** in the Enum type definition.
- ✓ If an Enum type is non-dynamic (by **default**, dynamic="**false**") we can't add new values **at runtime**.
- ✓ If we add any **non-dynamic** Enum type **without values**, **build will fail** as it does not have any effect.
- ✓ So if you want to add new values at run time we have to make dynamic="**true**" for an Enum type.
- ✓ We can change the **flag anytime** but enforces a system update.
- ✓ If dynamic="**false**" the service layer generates real java enums (having a fixed set of values).
- ✓ If dynamic="**true**" it generates **enums** which can be used without fixed values (can add run time values).

Q: How Relation (Many – Many) works in backend?

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

- ✓ 1 table will be created for **Products** & 1 table will be created for **Categories** & 1 extra table will be created for **LinkItem** (elements on both sides of the relation are linked together via instances of a LinkItem table)
- ✓ Each **LinkItem** instance stores the **PKs** of related items for each row. So every **row of product** with associated category will have 1 row in LinkItem table with PK of **Category** and associated PK of **Product**.
- ✓ **LinkItem** instances are used internally by hybris.
- ✓ We just need to call **getters** and **setters** at the API level to set and get the values.
- ✓ **Note: - Extra table** will be created only for **many to many** relations

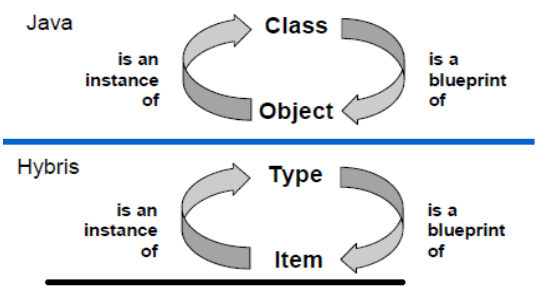
Note: - We know that **Collection** can be used as an **alternative** for **1 – M relation**.

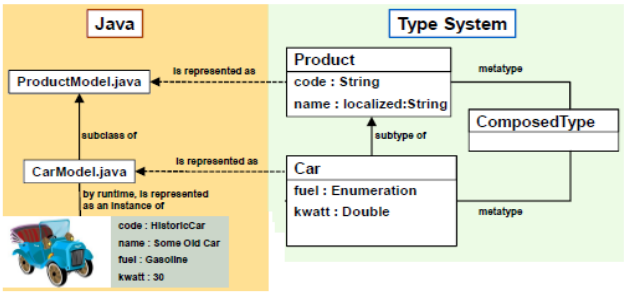
	Collection	Relation
Advantage	Fast retrieval as it does not need any Join to get the results.	There is no chance of data truncation (no matter how big the size). It's **bidirectional**.
Disadvantage	If collection **size grows** there is a possibility of data truncation. It's **unidirectional** It's **not searchable** as there will be a list of PKs only present in the column.	**Slow** in retrieval due to join between the tables required
When to Use	When we are sure that in **our current** & **future** requirements, we will not have many rows mapped for one side. It means **collection size** is small, Bcoz it helps to achieve **faster retrieval.**	Whenever **collection size is bigger** or there is a chance that it **can grow** bigger then **go with Relation,** bcoz there will be no data truncation. For **M - N**, we should go for Relation always.
When not to Use	Whenever the collection size is very big as it can lead to data truncation.	When **collection size is smaller** to compensate slow retrieval of Relation but in that case, we need to negotiate with **Bidirectional** mapping.

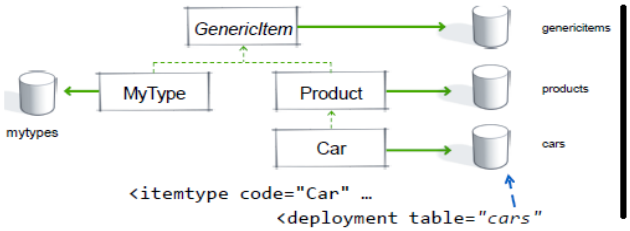
Q: Explain what is **Redeclare in items.xml? =** In Java, we can a **variable** with same in in **Parent** & **Child** classes (In this case, Parent variable will be inherited, but will be hidden as Child also having same variable). We can also **change** variable **data type** in child with same variable name.

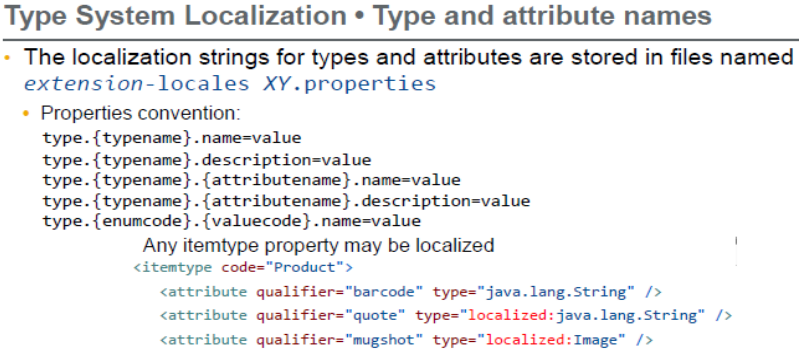
Similar to **Variable Hiding in Java**, In **Hybris we have "Re Declaring" the attributes**.

- ✓ Sometimes it is required to **re declare** the attribute in the **child item type** for various reasons.
- ✓ It could be to **change the Data type** of an attribute or make an attribute as **read only** etc.
- ✓ We can **re declare** the **same attributes** in the Child item type to change such **behavior**
- ✓ Change the **modifier** from **read=true** to **read=false**. We can make **attribute** as **unique**.
- ✓ We can add **write=false**. We can also change the **type of the attribute** but only to **subtypes**.









Q: Using a **standard Hybris configuration**, is this a valid item type definition?

<itemtype code="MyItem" abstract="false" extends="GenericItem"> </itemtype>

1. **No, this item needs a deployment table**

   Q: When we should define deployment mandatorily?

   1) Defining new item type by extending GenericItem (Above is the example).

   2) Defining new item type by extending existing item type for which there is no deployment

```
<itemtype code="AbstractOrder"  extends="GenericItem"  autocreate="true"  generate="true"  abstract="true">
<itemtype code="OneTwo" extends="AbstractOrder"  autocreate="true"  generate="true">
        <deployment table="Carts" typecode="43"/>
```

2. No, this item needs at least one attribute
3. No, only abstract items can extends GenericItem
4. No, GenericItem is a not a valid type
5. Yes, this is a valid item type definition

Q: What is the recommended way to **create a new extension** within Hybris V6?

1. Use the installer script with the extgen recipe
2. Extensions are automatically created by the build framework based on your dependencies
3. **Use the build framework with the extgen ant target to create a new extension from a template**
4. Use the build framework with the extgen maven goal to create a new extension from a template

Q: All CMS item types extend?

1. GenericItem 2. CMS Item

3. CMSItem or CMSRelation 4. CatalogAware

Q: What is the notification framework used for ?

1. notify cockpit users **2. notify customers**

3. notify administrators 4. facade for the event service

Q: Is it possible to add new enumeration values by runtime?

1. no 2. yes if you are using JRebel **3. yes if the enumeration is dynamic**

4. yes if the enumeration is a subtype of the HybrisEnumValue class

Q: What statements are wrong about extensions?

1. they have to be inside the bin folder 2. they need to have a dependency to yempty

3. they can written using Groovy 4. they are always automatically loaded

Q: When creating a new item how do you generate a new primary key?

1. using java.util.UUID.randomUUID() 2. using the primaryKeyService

3. primary keys creation is automatically handled by the database

4. primary keys creation is automatically handled by the service layer

Hybris DB Internals - Creating tables?

- Hybris creates tables during INIT / Update. It loads old-style **([-.]advanced-deployments.xml)** and new style data model definitions **(-items.xml)**.
- Hybris generates DB schema **using HybrisSchemaGenerator.initialize()**. Schema is created as a set of SQL files in temporary directory using Apache DDLUtils SQLBuilder. Hybris creates three files: -
    - **DropDDL** = With Drop Table statements
    - **DDL** = With CREATE statements (tables and indexes)
    - **DML** = With INSERT / UPDATE statements
- The generated files are executed as a batch using Spring Core JDBC.

Table types: -

- **Item Type Tables** =
    - Major part of Hybris DB tables are item type tables. These created from ***-items.xml**
    - Their names are specified in '**deployment'** tag. DB tables can have any name given by developer.
    - Actually, in this process some other tables are created or updated too.
      Example, Localized Values Tables are created if you have at least one localized attribute.
      Generic Audit Tables are created for all item type tables automatically (New feature from **6.6+**).
    - Each item type table has at least the following columns: -
      hjmpTS createdTS modifiedTS TypePkString OwnerPkString PK aCLTS
- **Localized Values Tables** =
    - If the table contains localized properties (having values in different languages), these values are stored separately from the main item type table. Table with such data has a **suffix "lp"**.
      Example, productslp has the following attributes:
      p_name p_description p_segment p_articlestatus p_summary p_style
- **Generic Audit Tables** =
    - Their names are using the item **table name, code & a suffix "sn"**.
      Example, for Products, the name of the change history table will be "**products1sn**".
      Where 1 = Type code of Product & "sn" is a fixed suffix. If tables having a long name, it will be truncated. **Example**, for table "products4restriction", history table is products4restri1081sn".
    - Not all changes are saved. You need to activate this feature for a particular type.
      It has been activated **OOB** for types = Address, B2bunit, BaseSite, Btgsegment, Cart, CartEntry, City, Consent, Country, CsTicket, Currency, Employee, Language, Order, Orderentry, Partneraddress, Paymentinfo, Product, Quote, Region, Title, Unit, User, Usergroup and ….
    - The structure of the change history tables:
      ID ITEMPK ITEMTYPEPK timestamp currenttimestamp changinguser …..
- **Property Tables** =
    - Specified for item types in ***-items.xml** as part of '**deployment'** tag ('propertytable' attribute).
      Addressprops quoteentryprops orderentryprops quoteprops
- **System Tables** =
      Aclentries Types = atomictypes, collectiontypes, composedtypes, maptypes
      Configitems enumerationvalues genericitems links metainformations numberseries