1. **Why and What use of typecode in Items in Hybris ?**

Before going sleep, I want to discuss some usefull topic with you about related to typecode in hybris

have you ever think of below code

modelService.get(PK) -it will gives result of model of that primary key

So from above code it seems that in hybris, Primary Key(PK) all items of whole types is unique

so again my question is that when above code is execute.  is hybris search all items in all table in hybris ?

surely answer is NO then how modelService search that items ?

Concept -  So here typecode of items comes in picture. Basically in hybris "have you ever think how PK generate in hybris". Primary Key of item of type is generate from using of below things  
  a. typeCode of type  
  b. getClusterID();  
  c. CreationTime();  
    
So by above combination of logic , primary key of item created in hybris

So when you write modelService.get(PK) then from PK it extract typecode of ITEM then it directly apply search in particular table . so i think you got understand about type

What is Promotion evaluation and partially activated promotion in Hybris ?

So Guys,

In every of my **Hybris interview** I face question related to **Promotion**, so today let’s start with some of them.

1. **What is Promotion evaluation and partially activated promotion ?**

Let’s you have a promotion "Buy 4 get 1 free" means when you buy 4 quantity then you will get 1 free item of same

So there will start **evaluate promotion** against a **shopping cart** and **order** then question will what will evaluate, it could be below

1. **Evaluate** which of the **available promotions have been activated or "fired".**

2. **Evaluate** which of the **available promotions have been partially activated or "could fire".**

So above both mode of evaluation is performing by **Promotion extension**.

So from above if User has added 4 quantities then there promotion action will fire i.e. means **promotion activated**

But suppose user has added 3 products not 4 then it means he **very short from to get promotion** means if he buy 1 more then he can get promotion. Then this **condition called “could fire promotion”** i.e. **partially activated promotion**

So In above case **Hybris do associate a single item to this** and communicated to customers on a website. For instance, a **message** such as **"Add another item to your basket and 1 free in your order"** could be displayed.

**Technical:**

If you look in **Promotion logic (In jalo class)** where you will find that below method

**public** List<PromotionResult> evaluate(**final** SessionContext ctx, **final** PromotionEvaluationContext promoContext)

       {

}

**Activated Promotion:**

       // Create a view of the order containing only the allowed products

**final** PromotionOrderView orderView = promoContext.createView(ctx, **this**, restrictResult.getAllowedProducts());

**if** (orderView.getTotalQuantity(ctx) >= qualifyingCount)

                     {

**final** **long** eligibleCountForPromotion = orderView.getTotalQuantity(ctx) / qualifyingCount;

**final** **long** nonEligibleCount = orderView.getTotalQuantity(ctx) % qualifyingCount;

**final** **long** totalFreeCount = freeCount \* eligibleCountForPromotion;

                           // Begin logging of promotions consuming order entries

                           promoContext.startLoggingConsumed(**this**);

                           // Get a price comparator

**final** Comparator<PromotionOrderEntry> comparator = PromotionEvaluationContext.*createPriceComparator*(ctx);

                           // Consume high priced items as these are the ones that will be paid for

                           orderView.consumeFromTail(ctx, comparator, (orderView.getTotalQuantity(ctx) - (totalFreeCount + nonEligibleCount)));

                           // Consume the free items from the cheap end of the list, as these result in the lowest discount

**final** List<PromotionOrderEntryConsumed> freeItems = orderView.consumeFromHead(ctx, comparator, totalFreeCount);

                           // Create the actions to take for this promotion to fire.  In this case an entry level discount is created

                            // for each of the free items.

**final** List<AbstractPromotionAction> actions = **new** ArrayList<AbstractPromotionAction>();

**for** (**final** PromotionOrderEntryConsumed poec : freeItems)

                           {

                                  // Set the adjusted unit price to zero, these are free items

                                  poec.setAdjustedUnitPrice(ctx, 0D);

**final** **double** adjustment = poec.getEntryPrice(ctx) \* -1.0D;

                                  // This action creates an order entry discount.

                                  actions.add(PromotionsManager.*getInstance*().createPromotionOrderEntryAdjustAction(ctx, poec.getOrderEntry(ctx),

                                                adjustment));

                           }

                           // Put together a the result for this iteration.

**final** PromotionResult result = PromotionsManager.*getInstance*().createPromotionResult(ctx, **this**,

                                         promoContext.getOrder(), 1.0F);

                           // Get a list of all the order entries that were consumed during this run

**final** List<PromotionOrderEntryConsumed> consumed = promoContext.finishLoggingAndGetConsumed(**this**,**true**);

                           result.setConsumedEntries(ctx, consumed);

                           // Add the actions that this promotion has produced

                           result.setActions(ctx, actions);

                           // Add the result object to the list of results

                           results.add(result);

                     }

**Partially activated promotion:**

// Check to see if there are still some qualifying products in the basket

**final** **long** remainingCount = orderView.getTotalQuantity(ctx);

**if** (orderView.getTotalQuantity(ctx) > 0)

                     {

                           // Start logging the products we could take

                           promoContext.startLoggingConsumed(**this**);

                           // Consume the products passing false the removeFromOrder.  This means that we noted which products

                           // \*could\* cause us to fire, but are not actually removing them from the context.

                           orderView.consume(ctx, remainingCount);

                           // The certainty for this is calculated as a percentage based on the qualifying items available versus

                           // the number needed to make this promotion fire.

**final** **float** certainty = (**float**) remainingCount / (**float**) qualifyingCount;

                           // Create the promotion result

**final** PromotionResult result = PromotionsManager.*getInstance*().createPromotionResult(ctx, **this**,

                                        promoContext.getOrder(), certainty);

                           // Fill in the entries we could have consumed

                           result.setConsumedEntries(promoContext.finishLoggingAndGetConsumed(**this**, **false**));

                           // Add the result to the list of results

                           results.add(result);

                     }

So in partially activated promotion, still Promotion Results (promotion action) return with message “”.

**What is Node in Bussiness Process and How many type of Node in Bussiness Process in Hybris**

In my last Hybris interview i got question about **Bussiness Process** in Hybris and question was "**What is Node in Bussiness Process and How many type of Node in Bussiness Process**"

Related to what is Bussiness process which i allready discuss so lets discuss about **type of Nodes** in bussiness process

**So what is Node?**

After define process in **hybris** (through XML ) as below

<?xml version="1.0"encoding="utf-8"?>

<process xmlns="<http://www.hybris.de/xsd/processdefinition>" name="everyReplyProcess"

start="everyReplyStart" onError="onError" processClass="de.hybris.platform.fulfilment.model.ConsignmentProcessModel">

... ...</process>

Then there is must to add content in process and that content is steps of your process and each step is called Node of process and each Node has tell below

**1**. Which **node** has to be invoked next

**2**. Each **Node** has **ID**

In above **process definition** there is one declare Node that is **start node** of your process in above case**everyReplyStart** is **declare node**

In hybris there is below **type of Node** and there use

**1. Action Node**  - Actions are the most important part of the **processengine** functionality. Normally, they have to**implement a logic or call specialized services** to execute tasks that are necessary in a process

**example**

<action id="isEveryReplyProcessCompleted" bean="bprocessesCompleted">

  <transition name="OK"to="sendNotification"/>

  <transition name="NOK"to="waitForComplete"/>

</action>

**2. Wait Node** - Lets during process you need to **communicate to external environment** or need to wait for complete of **subprocess** (inside process you can also define **subprocess**) then we need to define wait node as below

<wait id="waitForEverReplySubprocessEnd" then="isProcessCompleted"> <event>EverReplySubprocessEnd</event>

</wait>

so after wait complete which sense by your event given above call Node ID - isProcessCompleted

**3. Split Node** -  Suppose during in process you want to execute Actions in parallel then you can do by Split node as below

<split id="splitNode">

  <targetNode name="node1"/> <targetNode name="node2"/>

</split>

**4. Notify Node** - Lets during process execution you want to notify about status of process to someone then you define**Notify Node** in process as below

<notify id="notifySomeOne"then="nextNodeID">

  <userGroup name="everyReplyGroup"message="Perform action"/>

</notify>

**5. End Node** - it is ends the **process** and **stores state** and message in a process

<end id="error" state="ERROR">All went wrong.</end>

<end id="success" state="SUCCEEDED">Everything was fine</end>