2012

Mallika Sagar

HANG OFF TABLES

Contents

Hang Off Tables - Overview

Creating a hang-off entity enables us to:

.. Create a relationship between a Selling and Fulfillment Foundation standard table and a hang-off table.

.. Invoke Extensible APIs that store, modify and retrieve data from hang-off tables.

Thus, hang off is a table that has relationship with one of the standard application database table. which is also called the hang-off table's parent. A parent table can have many child hang-off tables but a hang-off table cannot have more than one parent. Currently multiple parent relationship for a hang-off table is not supported. Also, not all tables can act as parent for the hang-off tables. This can be verified in the Extending_the_Database guide for the particular version. Till the latest release(9.2), the ones which are marked as hang-off enabled are - order, order line, work order, shipment, item, and organization tables.
Sterling ERD can also be checked to verify whether a particular table can have hang-off table created or not, For the hang off enabled tables, the following would be mentioned in the Entity Definition, "**You can extend this table by adding columns and by creating hang-off table(s) ".**

Hang-offs can be created both for OOTB tables and Custom tables till n levels, and there is no functional limitation defined for "n" levels in either of the cases. However, creating a chain of hang-off on hang-off might have an adverse affect on the performance of the product as more SQL calls will be fired. Hence, sufficient testing must be done around the performance aspect of such an implementation.

Before we start off, we should familiarize our self with certain rules which have to be kept in mind while creating hang off tables:

1. Hang-off table names as well as Primary Key must not start with a Y.
2. Hang-off table must have a primary key.
3. The YIFApi interface does not extend APIs for hang-off tables. Therefore, the APIs for these tables must be configured as services.
4. Javadocs are not created for the APIs created by the infrastructure to support hang-off tables.
5. XSD generation and validation is not done for custom or hang-off tables.
6. Every hang-off table must have below fields:

CREATETS

MODIFYTS

CREATEUSERID

MODIFYUSERID

CREATEPROGID

MODIFYPROGID

LOCKID

NOTE: If not passed in the extensions xml, these fields will be created by the product automatically.

Steps to create hang-off table

1. Copy and rename the <INSTALL_DIR>/repository/entity/extensions/Extensions.xml.sample file as
<INSTALL_DIR>/extensions/global/entities/<your_filename>.xml
To understand what changes we need to make to the extensions.xml file, please view the contents of a sample extensions.xml below:

Consider a scenario where we want to create a hang off table for YFS_ORDER_HEADER called EXTN_ITEM_ACTIVATION. The extended table will have:

- primary key as ITEM_ACT_KEY.
- two other columns, ORDER_LINE_KEY and SERIAL_NUMBER.
- relationship established with the parent table through the column ORDER_HEADER_KEY.
- audits enabled for ORDER_LINE_KEY and SERIAL_NUMBER.

Make the following modifications to the <your_filename>.xml file placed at <INSTALL_DIR>/extensions/global/entities in the first step under 'Steps to create hang-off table':

```
 <?xml version="1.0" encoding="UTF-8"?>
<DBSchema>
            <Entities>
                        <Entity ApiNeeded="Y" AuditRequired="N" Description="Contains Activation and Deactivation
Records" HasHistory="False" Prefix="EXTN_" TableName="EXTN_ITEM_ACTIVATION">
                                    <Attributes>
                                                <Attribute ColumnName="ITEM_ACT_KEY" Type="CHAR" Size="24" DefaultValue="' '" Description="System
generated primary key" Nullable="False"XMLName="ItemActKey"/>
                                                <Attribute ColumnName="ORDER_HEADER_KEY" Type="CHAR" Size="24" DefaultValue="'
'" Description="" Nullable="False" XMLName="OrderHeaderKey"/>
                                                <Attribute ColumnName="ORDER_LINE_KEY" Type="CHAR" Size="24" DefaultValue="'
'" Description="" Nullable="False" XMLName="OrderLineKey"/>
                                                <Attribute ColumnName="SERIAL_NUMBER" Type="CHAR" Size="24" DefaultValue="'
'" Description="" Nullable="False" XMLName="SerialNumber"/>
                                         <Attribute
ColumnName="LOCKID" DataType="Lockid"  DefaultValue="0" Description="" Nullable="False" XMLName="LockID" />
                                                <Attribute ColumnName="CREATETS"  DataType="TimeStamp"  DefaultValue="sysdate" Description="Create
TimeStamp" />
             <Attribute ColumnName="MODIFYTS"  DataType="TimeStamp" DefaultValue="sysdate" Description="Modify TimeStamp" />
                        <Attribute ColumnName="CREATEUSERID" DataType="Createuserid"  DefaultValue="' '" Description="Creating User ID" />
                        <Attribute ColumnName="MODIFYUSERID" DataType="Modifyuserid"  DefaultValue="' '" Description="Modifying User ID" />
                         <Attribute ColumnName="CREATEPROGID" DataType="Createprogid"  DefaultValue="' '" Description="Creating Program ID" />
                         <Attribute ColumnName="MODIFYPROGID" DataType="Modifyprogid"  DefaultValue="' '" Description="Modifying Program ID" />
            </Attributes>
                                    <PrimaryKey Name="ITEM_ACT_PK">
                                                <Attribute ColumnName="ITEM_ACT_KEY" />
                                    </PrimaryKey>

                         <Parent ParentTableName="YFS_ORDER_HEADER" XMLName="YFSOrderHeader" >
                                 <Attribute ColumnName="ORDER_HEADER_KEY" ParentColumnName="ORDER_HEADER_KEY" />
                         </Parent>


                        <AuditReferences>
                                    <Reference ColumnName="ORDER_LINE_KEY" />
                                     <Reference ColumnName="SERIAL_NUMBER" />
                         </AuditReferences>
            </Entity>
  </Entities>
</DBSchema>
```

From this we have the following understandings:

a. Any hang-off entity has to be defined under Entities element and within Entity element :

```
………..
<Entities>
  <Entity …………
```

b. Understanding attributes of Entity element:

```
 <Entity ApiNeeded="Y" AuditRequired="N" Description="Contains Activation and Deactivation
Records" HasHistory="False" Prefix="EXTN_" TableName="EXTN_ITEM_ACTIVATION">
```

<u>APINeeded</u> : Indicates whether or not APIs should be generated for the hang-off table in question. Valid values are Y or N. A default set of API's are generated if Y is passed.

For example in the EXTN_ITEM_ACTIVATION table, Selling and Fulfillment Foundation creates following API's when the database extension jar is generated:

getEXTNItemActivation

listEXTNItemActivation

createEXTNItemActivation

modifyEXTNItemActivation

deleteEXTNItemActivation

AuditRequired : Value can be Y or N depending on whether audit is required for the hang-off table or not. If the hang-off table is Order related, then irrespective of whether the value is N or Y, audits will get logged in the yfs_order_audit and yfs_order_audit_detail wrt to the Audit References specified in the Entity definition.

If the hang-off table is non – order related, then If AuditRequired is Y, audits will be logged in the yfs_audit table else not. Again columns for which audits will be logged depends on the Audit References defined.

HasHistory : This flag is automatically inherited from the parent table. For example, let us assume that EXTN_ITEM_ACTIVATION table is created as an hang-off table for YFS_ORDER_HEADER, which has an associated history table. Then EXTN_ITEM_ACTIVATION_H is automatically generated by the database framework.

Prefix : is added to the hang-off table's name. In our example the Prefix used is "EXTN_" which is added to the hang-off table's name. It should not start with "Y".

TableName : Name of the hang-off table. It has to be prefixed by the Prefix specified. In the above example, name of the hang-off table is EXTN_ITEM_ACTIVATION.

c. After entity, all the columns for the hang-off table are defined as Attribute element.

# Valid data types are given in <INSTALL_DIR>/repository/datatypes/datatypes.xml file.

d. After all the columns are defined, primary key of the hang-off table is specified:

```
 <PrimaryKey Name="ITEM_ACT_PK">
                <Attribute ColumnName="ITEM_ACT_KEY" />
</PrimaryKey>
```

The ColumnName specified here refers to the hang-off table column.

# The name of the primary key in the extension XML should end with _PK.

e. After this, parent for the hang-off table is specified :

```
 <Parent ParentTableName="YFS_ORDER_HEADER" XMLName="YFSOrderHeader" >
            <Attribute ColumnName="ORDER_HEADER_KEY" ParentColumnName="ORDER_HEADER_KEY" />
 </Parent>
```

Here, ColumnName refers to the column name of hang-off table and ParentColumnName refers to the name of the column in the parent table from which the column in the hang-off table would derive the values.

c. If we want audits for the hang-off table, we need to specify the columns for which audit needs to be recorded.

```
<AuditReferences>
                <Reference ColumnName="ORDER_LINE_KEY" />
                <Reference ColumnName="SERIAL_NUMBER" />
</AuditReferences>
```

So, if there are any changes in the above columns, audits will be recorded in the corresponding audit tables.

# Audit References should be defined below the Parent element to work properly.

2. Invoke the dbverify tool from <INSTALL_DIR>/bin/dbverify.cmd(or dbverify.sh depending on the environment)to generate the SQL scripts required for your hang-off table.

After dbverify.cmd script is run successfully, the corresponding scripts are generated at : <INSTALL_DIR>/bin.

Execute the scripts generated. This will create the hang-off table and build its relationship with the parent table.
OR
Run deployer.sh (or deployer.cmd on Windows) utility to re-build the entities.jar, from the <INSTALL_DIR>/bin directory. For example:

```
deployer.cmd -t entitydeployer
```

Note: Before building the database extensions, make sure that all the extension files are stored in the INSTALL_DIR/extensions/global/entities directory.
As a part of this utility, dbverify will be run implicitly to generate the scripts in <INSTALL_DIR>/bin and the generated scripts would get executed. Hence, we do not need to explicitly execute the generated scripts.

3. Build and deploy the extensions
To **build** the extensions to our database(such as creating hang off tables, adding columns to existing tables etc), re-build the entities.jar by running the deployer.sh (or deployer.cmd on Windows) utility from the <INSTALL_DIR>/bin directory. For example:

```
deployer.cmd -t entitydeployer
```

Again, before building the database extensions, make sure that all the extension files are stored in the INSTALL_DIR/extensions/global/entities directory.
NOTE: In case deployer.cmd has already been run in Step2 then, we do not need to run it again.

After we have built the required extensions, we must **deploy** them to make Sterling Selling and Fulfillment Foundation available for use. For this, we have to re-build and deploy the application EAR file.

1. Set up the application server appropriately for deploying the application.

2. Create the EAR package for the application server. To create the application EAR file for a single WAR deployment, run the following command from the <INSTALL_DIR>/bin directory:

```
buildear.cmd -Dappserver=<application server> -Dwarfiles=<war file> -Dearfile=<ear file>
```

Running this command creates the EAR file in the <INSTALL_DIR> /external_deployments directory.

3. Deploy the EAR file on the application server.

For more information about creating and deploying the EAR file, we can also refer to the installation documentation.

After following the above steps, the hang-off table will be created.
# For more details on building and deploying the extensions you can also refer the Customization Basics guide.

Some related tasks that can be performed on hang off tables

Modify the existing API templates to incorporate the hang-off attribute changes

The standard APIs can be extended to retrieve information from the hang-off table.

For generating the template XML, run the template XML generation tool from your <INSTALL_DIR>\bin directory by using the following command:

sci_ant.cmd -Dtable=<TABLE_NAME> -f templateXmlGen.xml

Run the above script for new table as well as parent table.

eg. EXTN_ITEM_ACTIVATION , YFS_ORDER_HEADER

Once the command is run, the sample XML files are placed in the <INSTALL_DIR>/extn/sampleXML directory as <TABLE_NAME>_sample.xml.
For example in our case, if we run the command :

sci_ant.sh -Dtable=EXTN_ITEM_ACTIVATION -f templateXmlGen.xml

the sample XML that is generated is extn_item_activation_sample.xml :

```
<?xml version="1.0" encoding="UTF-8"?>
<EXTNItemActivation
Createprogid=" " Createts=" " Createuserid=" " ItemActKey=" " LockID=" " Modifyprogid=" " Modifyts=" " Modifyuserid=" " OrderHeaderKey=" " OrderLineKey
=" " SerialNumber=" " >
        <YFSOrderHeader
AdjustmentInvoicePending=" " AllocationRuleID=" " AuthorizationExpirationDate=" " AutoCancelDate=" " BillToID=" " BillToKey=" "BuyerMarkForNodeId=" " B
uyerOrganizationCode=" "
 BuyerReceivingNodeId=" " CancelOrderOnBackorder=" " CarrierAccountNo=" " CarrierServiceCode=" " ChainType=" "  ChargeActualFreightFlag=" " ContactKe
y=" " CreatedAtNode=" "  Createprogid=" " Createts=" " Createuserid=" " CustCustPONo=" " CustomerEMailID=" " CustomerPONo=" "DefaultTemplate=" " Deli
veryCode=" " DepartmentCode=" " Division=" "  DoNotConsolidate=" " DocumentType=" " DraftOrderFlag=" " EnteredBy=" " EnterpriseCode=" "EntryType=" "
ExchangeType=" " FreightTerms=" " HasDerivedChild=" " HasDerivedParent=" " HoldFlag=" " HoldReasonCode=" " InternalApp=" "
 InvoiceComplete=" "IsLineShipComplete=" " IsLineShipSingleNode=" " IsShipComplete=" " IsShipSingleNode=" " Lockid=" " MarkForKey=" " Modifyprogid=" "
Modifyts=" " Modifyuserid=" "NextAlertTs=" " NotificationReference=" " NotificationType=" " NotifyAfterShipmentFlag=" " OptimizationType=" " OrderCompl
ete=" " OrderDate=" " OrderHeaderKey=" "OrderName=" " OrderNo=" " OrderPurpose=" "
 OrderType=" " OriginalTax=" " OriginalTotalAmount=" " OtherCharges=" " Override=" " PaymentRuleId=" " PaymentStatus=" "PendingTransferIn=" " Personal
izeCode=" " PriceOrder=" " PriceProgramKey=" " PriceProgramName=" " PriorityCode=" " PriorityNumber=" " PropagateCancellations=" " PurgeHistoryDate="
" Purpose=" " ReceivingNode=" " ReqCancelDate=" " ReqDeliveryDate=" " ReqShipDate=" "
 ReserveInventoryFlag=" " ReturnByGiftRecipient=" " ReturnOrderHeaderKeyForExchange=" " SCAC=" " SaleVoided=" " SearchCriteria1=" " SearchCriteria2=" "
 SellerOrganizationCode=" " ShipNode=" " ShipToID=" " ShipToKey=" "SourcingClassification=" " TaxExemptFlag=" " TaxExemptionCertificate=" " TaxJurisdictio
n=" " TaxPayerId=" "  TermsCode=" " TotalAdjustmentAmount=" " VendorID=" ">
                <PriceInfo Currency=" " EnterpriseCurrency=" " HeaderTax=" " ReportingConversionDate=" " ReportingConversionRate=" " TotalAmount=" "/>
        </YFSOrderHeader>
</EXTNItemActivation>
```

As we can see, in the above XML all the attributes of YFSOrderHeader are also there as YFSOrderHeader is parent of the hang-off table.

If we execute the command :

sci_ant.sh -Dtable=YFS_ORDER_HEADER -f templateXmlGen.xml

the xml that is generated is yfs_order_header_sample.xml:

```
<?xml version="1.0" encoding="UTF-8" ?>
- <Order AdjustmentInvoicePending="" AllocationRuleID=""……………………….. Modifyuserid="" SystemDefined="" />
 + <OrderReleases>
 + <OrderHoldTypes>
 + <ChargeTranDistributions>
 + <OrderInvoiceList>
 + <ProductServiceAssocs>
 + <OrderLineReservations>
 + <Awards>
 + <OrderAuditList>
 + <OrderDates>
 + <OrderLineRelationships>
 + <junkHeaderCharges>
 + <Promotions>
 + <References>
 + <PaymentMethods>
 + <ReceivingDescripancies>
 + <KitLineList>
 + <OrderLines>
 + <ChainedOrderLines>
 + <DerivedOrderLines>
 + <ChargeTransactionDetails>
 + <ActivityDemands>
 + <PlannedTasks>
 + <DockAppointmentList>
 + <AnswerSets>
 + <SpecialServices>
 + <Notes>
 + <Instructions>
 + <AdditionalAddresses>
 <Extn>
       <EXTNItemActivationList>
            <EXTNItemActivation
AvailableDate="" Createprogid="" Createts="" Createuserid="" ItemActKey="" LockID="" Message="" Modifyprogid="" Modifyts="" Modifyuserid=""OrderHeaderKey="" OrderLineKey="" RequestType="" RetryCounter="" SerialNumber="" ShipmentKey="" Status="">
                   <YFSOrderHeader
AdjustmentInvoicePending="" AllocationRuleID="" AuthorizationExpirationDate="" AutoCancelDate="" BillToID="" BillToKey=""BuyerMarkForNodeId="" BuyerOrganizationCode="" BuyerReceivingNodeId="" CancelOrderOnBackorder="" CarrierAccountNo="" CarrierServiceCode="" ChainType="" ChargeActualFreightFlag="" ContactKey="" CreatedAtNode="" Createprogid="" Createts="" Createuserid="" CustCustPONo="" CustomerEMailID="" CustomerPONo=""DefaultTemplate="" DeliveryCode="" DepartmentCode="" Division="" DoNotConsolidate="" DocumentType="" DraftOrderFlag="" EnteredBy="" EnterpriseCode="" EntryType=""ExchangeType="" FreightTerms="" HasDerivedChild="" HasDerivedParent="" HoldFlag="" HoldReasonCode="" InternalApp="" InvoiceComplete="" IsLineShipComplete=""IsLineShipSingleNode="" IsShipComplete="" IsShipSingleNode="" Lockid="" MarkForKey="" Modifyprogid="" Modifyts="" Modifyuserid=""
NextAlertTs="" NotificationReference="" NotificationType="" NotifyAfterShipmentFlag="" OptimizationType="" OrderComplete="" OrderDate="" OrderHeaderKey="" OrderName="" OrderNo="" OrderPurpose=""OrderType="" OriginalTax="" OriginalTotalAmount="" OtherCharges="" Override="" PaymentRuleId="" PaymentStatus="" PendingTransferIn="" PersonalizeCode=""PriceOrder="" PriceProgramKey="" PriceProgramName="" PriorityCode="" PriorityNumber="" PropagateCancellations="" PurgeHistoryDate="" Purpose="" ReceivingNode=""ReqCancelDate="" ReqDeliveryDate="" ReqShipDate="" ReserveInventoryFlag=""
ReturnByGiftRecipient="" ReturnOrderHeaderKeyForExchange="" SCAC="" SaleVoided=""SearchCriteria1="" SearchCriteria2="" SellerOrganizationCode="" ShipNode="" ShipToID="" ShipToKey="" SourcingClassification="" TaxExemptFlag="" TaxExemptionCertificate="" TaxJurisdiction="" TaxPayerId="" TermsCode=""
 TotalAdjustmentAmount="" VendorID="">
                         <PriceInfo
Currency="" EnterpriseCurrency="" HeaderTax="" ReportingConversionDate="" ReportingConversionRate="" TotalAmount="" />
                   </YFSOrderHeader>
       </EXTNItemActivation>
  </EXTNItemActivationList>
 </Extn>
</Order>
```

From the above we see that the hang-off table attributes are included in the XML under Extn element.

If the table modifications impact any APIs, we must extend the templates of those APIs by placing the extended API templates in the <INSTALL_DIR>/extensions/global/template/api directory.

As in our case, suppose we want to insert data into EXTN_ITEM_ACTIVATION table while creating order, then template for createOrder should be placed here. It might be like:

```
<?xml version="1.0" encoding="UTF-8"?>
<!--createOrder Output XML-->
 <Order EnterpriseCode="DEFAULT" OrderHeaderKey="Test" DocumentType="0001" OrderNo="Test-1">
<EXTNItemActivation OrderHeaderKey="Test" ItemActKey="200709261450" /
</Order>
```

We will need to build and deploy the extensions after making any such change to the API template by following the below steps:
To **build** extensions to application resources, we must rebuild the resources.jar file by running the deployer.sh(or deployer.cmd on Windows) utility from the INSTALL_DIR/bin directory. For example:

```
deployer.cmd -t resourcejar
```

This applies to all application resources, including:
• Extended APIs, Events, and XSL templates
• Modifications made in the database, resources, and template directories

After we have built the required extensions, we must **deploy** them to make Sterling Selling and Fulfillment Foundation available for use. For this, we have to re-build and deploy the application EAR file.

a. Set up the application server appropriately for deploying the application.
b. Create the EAR package for the application server. To create the application EAR file for a single WAR deployment, run the following command from the <INSTALL_DIR>/bin directory:

```
buildear.cmd -Dappserver=<application server> -Dwarfiles=<war file> -Dearfile=<ear file>
```

Running this command creates the EAR file in the <INSTALL_DIR> /external_deployments directory.

c. Deploy the EAR file on the application server.
For more information about creating and deploying the EAR file, we can also refer to the installation documentation.

Configuring Services for Hang-off APIs

The APIs generated for the hang-off tables can only be invoked as service. While creating the service, we should choose the option Extended Database API. In the API Name dropdown, we will be able to see all the hang-off related APIs. We can choose appropriately and create the corresponding service.

From the HTTP API Tester, these services can be called to insert, get, modify and delete records from the hang-off table by passing the corresponding input XML. For more details on service configuration, we can refer the Application_Platform_Configuration_Guide.pdf

Inserting/Modifying records in the hang-off table

- via Base APIs

Lets approach this by considering the hang-off table EXTN_ITEM_ACTIVATION which has the parent YFS_ORDER_HEADER. Assuming that AUDITREQUIRED is 'N' in the extensions.xml for this hang-off table.

If we want to insert records in the hang-off table while the order is created i.e while createOrder API is invoked then we can pass the below XML to the createOrder API:

```
<Order  BuyerOrganizationCode=""  EnterpriseCode="DEFAULT"  OrderName="ConfirmAssgn-3" OrderNo="Test-2007"
SellerOrganizationCode="DEFAULT"ReqDeliveryDate="2006-10-17" ReqShipDate="2006-10-09" OrderHeaderKey="Test" >
        <OrderLines>
                <OrderLine  OrderedQty="1" ReservationMandatory="N"  ShipNode="KNODE" >
                        <Item  ItemID="Watch" ProductClass="A" Quantity="1"/>
                        <PersonInfoShipTo Country="US"  ZipCode="12345" />
                </OrderLine>
        </OrderLines>
        <PersonInfoShipTo Country="US"  ZipCode="12345" />
        <PersonInfoBillToCountry="US"  ZipCode="12345"/>
        <PersonInfoContact Country="US"  ZipCode="12345" />
  <Extn>
                <EXTNItemActivationList>
                        <EXTNItemActivation Createprogid=" "  Createts=" " Createuserid=" " ItemActKey=" " LockID=" " Modifyprogid=" " Modifyts=" "
        Modifyuserid=" " OrderHeaderKey="Test1111" OrderLineKey=" " SerialNumber=" ">
        <YFSOrderHeader BuyerOrganizationCode=" "
OrderHeaderKey="Test1111" OrderNo=" " SellerOrganizationCode=" " ShipNode=" " ShipToID=" " ShipToKey=" " >
        </YFSOrderHeader>
      </EXTNItemActivation>
    </EXTNItemActivationList>
  </Extn>
</Order>
```

In the above XML please note that the hang-off table attributes are present under EXTN element. After the XML is passed to the createOrder API, we can see that corresponding records are inserted in the Order related tables as well as the EXTN_ITEM_ACTIVATION table.

Similarly, if we want to make any changes to the values in EXTN_ITEM_ACTIVATION table, we can do it by passing relevant XML to changeOrder API :

```
<Order Action="MODIFY" OrderHeaderKey="Test1111" >
      <Extn>
           <EXTNItemActivationList>
             <EXTNItemActivation  SerialNumber="1111">
               <YFSOrderHeader  OrderHeaderKey="Test1111" />
                       </EXTNItemActivation>
           </EXTNItemActivationList>
      </Extn>
</Order>
```

At the Order level, we pass the Order Header Key for reference. At the EXTN level, we pass the value that we want to modify. Here we are trying to modify the earlier Serial_Number to a new value i.e 1111.

- via Services

To insert/modify/getDetails/getList/delete data to/from the hang-off table, we can use services as extended Database APIs cannot be called directly.

If we insert data to EXTN_ITEM_ACTIVATION table by invoking service configured for createEXTNItemActivation API from HTTP API Tester, we can pass the sample xml generated from the command below:

```
sci_ant.sh -Dtable=EXTN_ITEM_ACTIVATION -f templateXmlGen.xml
```

This will insert the corresponding record in EXTN_ITEM_ACTIVATION table.

Now to modify this record we can call a service which in turn invokes modifyEXTNItemActivation API. This will audit no records in yfs_order_audit and yfs_order_audit_detail table. If the Auditrequired is 'Y' in the Extensions.xml, then the audits will be recorded in the yfs_audit table else not.

Audits generated for hang off tables

When modifications are done on the hang off table then audits will get generated.

- If a base API is called to modify a column of the hang off table then audit records will be inserted in the parent audit table.
For eg, in our example if changeOrder API is used to modify the a column of the hang off table which has AuditRequired is Y (in extensions.xml) then corresponding audit records will be inserted in the yfs_order_audit and yfs_order_audit_detail table provided the modified column is referenced under the Audit References tag as well.

# If parent table is YFS_ORDER_HEADER then audit will go to YFS_ORDER_AUDIT table irrespective whether AuditRequired is 'Y' or 'N'.

- If extended database API is called through service to modify a column of hang off table which has AuditRequired is Y (in extensions.xml) then, audit will come in YFS_AUDIT table only. This will insert no records in yfs_order_audit and yfs_order_audit_detail table.