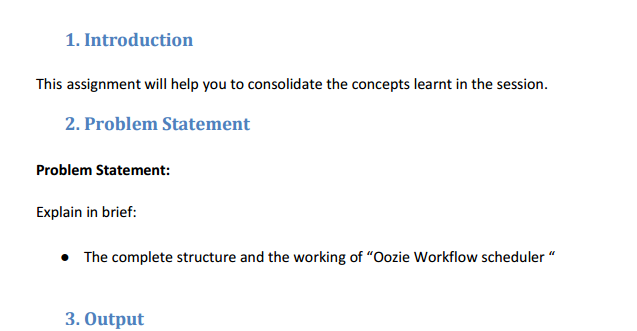
**Assignment 34.1**

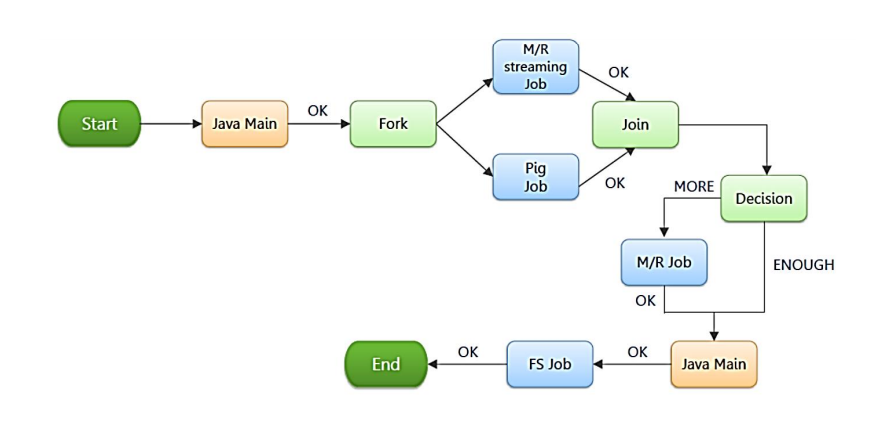
****

* **The complete structure and the working of Oozie Workflow scheduler**

Workflow in Oozie is a sequence of actions arranged in a control dependency DAG (Direct Acyclic Graph). The actions are in controlled dependency as the next action can only run as per the output of current action. Subsequent actions are dependent on its previous action. A workflow action can be a Hive action, Pig action, Java action, Shell action, etc.

One advantage of the Oozie framework is that it is fully integrated with the Apache Hadoop stack and supports Hadoop jobs for Apache MapReduce, Pig, Hive, and Sqoop. In addition, it can be used to schedule jobs specific to a system, such as Java programs.

**Oozie workflow**



An Oozie workflow is a collection of actions arranged in a directed acyclic graph (DAG). This graph can contain two types of nodes:

1. **Control nodes**

2. **Action nodes.**

1. **Control Nodes -** Control nodes, which are used to define job chronology, provide the rules for beginning and ending a workflow and control the workflow execution path with possible decision points known as fork and join nodes.

Control node achieves this by using -

1. **Start Control Node**

2. **End Control Node**

3. **Kill control Node**

4. **Decision Control Node**

5. **Fork and Join Control Nodes**

1. **Start Control Node**

The start node is the entry point for a workflow job, it indicates the first workflow node the workflow job must transition to.When a workflow is started, it automatically transitions to the node specified in the start .

A workflow definition must have one start node.

**Syntax:**

<workflow-app name="[WF-DEF-NAME]" xmlns="uri:oozie:workflow:0.1">

...

<start to="[NODE-NAME]"/>

...

</workflow-app>

The to attribute is the name of first workflow node to execute.

Example:

<workflow-app name="foo-wf" xmlns="uri:oozie:workflow:0.1">

...

<start to="firstHadoopJob"/>

...

</workflow-app>

1. **Kill control Node**

The kill node allows a workflow job to kill itself.

When a workflow job reaches the kill it finishes in error (KILLED).

If one or more actions started by the workflow job are executing when the kill node is reached, the actions will be killed.

* A workflow definition may have zero or more kill nodes.

**Syntax:**

<workflow-app name="[WF-DEF-NAME]" xmlns="uri:oozie:workflow:0.1">

...

<kill name="[NODE-NAME]">

<message>[MESSAGE-TO-LOG]</message>

</kill>

...

</workflow-app>

The name attribute in the kill node is the name of the Kill action node.

The content of the message element will be logged as the kill reason for the workflow job.

A kill node does not have transition elements because it ends the workflow job, as KILLED .

Example:

<workflow-app name="foo-wf" xmlns="uri:oozie:workflow:0.1">

...

<kill name="killBecauseNoInput">

<message>Input unavailable</message>

</kill>

...

</workflow-app>

1. **End Control Node**

The end node is the end for a workflow job, it indicates that the workflow job has completed successfully.

When a workflow job reaches the end it finishes successfully (SUCCEEDED).

If one or more actions started by the workflow job are executing when the end node is reached, the actions will be killed. In this scenario the workflow job is still considered as successfully run.

A workflow definition must have one end node.

**Syntax:**

<workflow-app name="[WF-DEF-NAME]" xmlns="uri:oozie:workflow:0.1">

...

<end name="[NODE-NAME]"/>

...

</workflow-app>

The name attribute is the name of the transition to do to end the workflow job.

Example:

<workflow-app name="foo-wf" xmlns="uri:oozie:workflow:0.1">

...

<end name="end"/>

</workflow-app>

1. **Decision Control Node**

A decision node enables a workflow to make a selection on the execution path to follow.

The behavior of a decision node can be seen as a switch-case statement.

A decision node consists of a list of predicates-transition pairs plus a default transition. Predicates are evaluated in order or appearance until one of them evaluates to true and the corresponding transition is taken. If none of the predicates evaluates to true the default transition is taken.

Predicates are JSP Expression Language (EL) expressions (refer to section 4.2 of this document) that resolve into a boolean value, true or false . For example:

${fs:fileSize('/usr/foo/myinputdir') gt 10 \* GB}

**Syntax:**

<workflow-app name="[WF-DEF-NAME]" xmlns="uri:oozie:workflow:0.1">

...

<decision name="[NODE-NAME]">

<switch>

<case to="[NODE\_NAME]">[PREDICATE]</case>

...

<case to="[NODE\_NAME]">[PREDICATE]</case>

<default to="[NODE\_NAME]"/>

</switch>

</decision>

...

</workflow-app>

The name attribute in the decision node is the name of the decision node.

Each case elements contains a predicate an a transition name. The predicate ELs are evaluated in order until one returns true and the corresponding transition is taken.

The default element indicates the transition to take if none of the predicates evaluates to true .

All decision nodes must have a default element to avoid bringing the workflow into an error state if none of the predicates evaluates to true.

**Example:**

<workflow-app name="foo-wf" xmlns="uri:oozie:workflow:0.1">

...

<decision name="mydecision">

<switch>

<case to="reconsolidatejob">

${fs:fileSize(secondjobOutputDir) gt 10 \* GB}

</case>

<case to="rexpandjob">

${fs:filSize(secondjobOutputDir) lt 100 \* MB}

</case>

<case to="recomputejob">

${ hadoop:counters('secondjob')[RECORDS][REDUCE\_OUT] lt 1000000 }

</case>

<default to="end"/>

</switch>

</decision>

...

</workflow-app>

**5 Fork and Join Control Nodes**

A fork node splits one path of execution into multiple concurrent paths of execution.

A join node waits until every concurrent execution path of a previous fork node arrives to it.

The fork and join nodes must be used in pairs. The join node assumes concurrent execution paths are children of the same fork node.

**Syntax:**

<workflow-app name="[WF-DEF-NAME]" xmlns="uri:oozie:workflow:0.1">

...

<fork name="[FORK-NODE-NAME]">

<path start="[NODE-NAME]" />

...

<path start="[NODE-NAME]" />

</fork>

...

<join name="[JOIN-NODE-NAME]" to="[NODE-NAME]" />

...

</workflow-app>

The name attribute in the fork node is the name of the workflow fork node. The start attribute in the path elements in the fork node indicate the name of the workflow node that will be part of the concurrent execution paths.

The name attribute in the join node is the name of the workflow join node. The to attribute in the join node indicates the name of the workflow node that will executed after all concurrent execution paths of the corresponding fork arrive to the join node.

Example:

<workflow-app name="sample-wf" xmlns="uri:oozie:workflow:0.1">

...

<fork name="forking">

<path start="firstparalleljob"/>

<path start="secondparalleljob"/>

</fork>

<action name="firstparallejob">

<map-reduce>

<job-tracker>foo:9001</job-tracker>

<name-node>bar:9000</name-node>

<job-xml>job1.xml</job-xml>

</map-reduce>

<ok to="joining"/>

<error to="kill"/>

</action>

<action name="secondparalleljob">

<map-reduce>

<job-tracker>foo:9001</job-tracker>

<name-node>bar:9000</name-node>

<job-xml>job2.xml</job-xml>

</map-reduce>

<ok to="joining"/>

<error to="kill"/>

</action>

<join name="joining" to="nextaction"/>

...

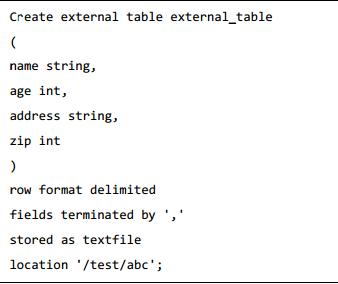
</workflow-app>

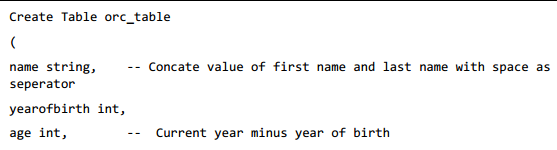
* **Action Node**

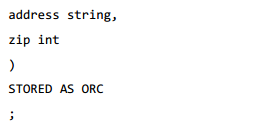
An action node represents a workflow task, e.g., moving files into HDFS, running a MapReduce, Pig or[Hive](http://www.guru99.com/hive-tutorials.html)jobs, importing data using Sqoop or running a shell script of a program written in Java.

**Example** - Suppose I want to load a data from external hive table to an ORC Hive table

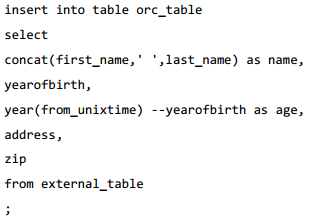
Step 1: **creating Hive external table (say external.hive) in text format**



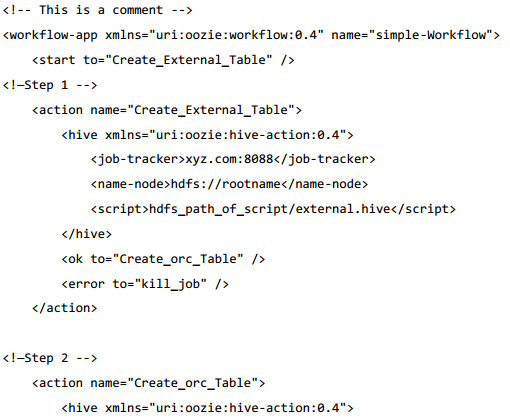
Step 2: **creating Hive external table (say external.hive) in ORC format** 



Step3: **Inserting into Table orc from Text Table**



**Instead of doing this manually we can use Oozie scheduler by using the following configuration File**





**Explanation**

Action Nodes in the above example defines the type of job that the node will run. Hive node inside the action node defines that the action is of type hive. This could also have been a pig, java, shell action, etc. as per the job you want to run. Each type of action can have its own type of tags. In the above job we are defining the job tracker to us, name node details, script to use and the param entity. The Script tag defines the script we will be running for that hive action. The Param tag defines the values which we will pass into the hive script. (In this example we are passing database name in step 3)