Tivoli Identity Manager 5.1

Data synchronization Execution Sequence

Data synchronization consists of the following steps:

1. Scheduling Data Synchronization

When the system administrator creates Data Synchronization schedule, an entry is made in the RESOURCES_SYNCHRONIZATIONS table to save the time of the schedule. A message is also sent to the Scheduler Bean, which contains the recurring schedule object and the destination JMS queue.

2. Execute Data Synchronization

A. Once we schedule a full data synchronization operation, an entry gets created in SCHEDULED_MESSAGE table. Every node in the ITIM cluster runs a daemon or server thread called scheduler that periodically scans the SCHEDULED_MESSAGE table and schedules the messages that are due. Only one node of the cluster will process an entry so the data synchronization operation will get scheduled on one node of the cluster. Once scheduled, the scheduler will send a message to JMS queue for processing. AdhocSyncDataMessageReceiver.onMessage() is main JMS handler method which receives the message from the JMS queue (adhocSyncQueue) and starts its execution. It receives synchronization unit as a parameter and starts data synchronization of selected sync unit.

B. To start the Data Synchronization the TIM Server has to acquire the lock. Before acquiring the lock the TIM Server tries to write its hostname to this table and the table is locked. This prevents other TIM Servers from writing to this table. In AdhocSynchronizationManagerLocal.acquireLock() method, first lock entry is added in SYNCHRONIZATION_LOCK table. (This table is used to avoid race condition when two TIM Servers in a clustered environment start Data Synchronizations at the same time.) After that in createHistoryEntry() method an entry in the history table of synchronization. i.e. SYNCHRONIZATION_HISTORY table. It set synchronization status as ‘started’. Following debug statement is logged at this step:-

(Converted DEBUG_MAX to DEBUG_MID statement)

```
<Trace Level="MID">
 <Time Millis="1296767215000"> 2011.02.04 02:36:55.000+05:30</Time>
 <Server Format="IP">itimwin3.tivlab.raleigh.ibm.com</Server>
 <ProductId>CTGIM</ProductId>
 <Component>com.ibm.itim.apps.ejb.adhocreport.synchronization</Component>
```

```
<ProductInstance>server1</ProductInstance>
<LogText><![CDATA[INSERT INTO itimuser.SYNCHRONIZATION_HISTORY (SYNC_ID, STATUS,
    STARTED_TIME, COMPLETED_TIME, SCHEDULED_TIME, REQUESTOR, REQ_TYPE, REQ_NAME, TEN-
    ANT, STATUS_DETAIL, SERVER_NAME) VALUES (0, 'Started', 1296767215000, 1296767215000, 0, 'ITIM
    Manager', 'DS', 'Data Synchronization', 'ou=ITIM51,dc=com', ?, 'itimwin3Node02Cell/itimwin3Node02/server1')
    - with status_detail = ]]></LogText>
<Source FileName="com.ibm.itim.apps.ejb.adhocreport.synchronization.AdhocSynchronizationManagerBean"
    Method="createHistoryEntry"/>
<Thread>Default : 1</Thread>
</Trace>
```

AdhocSynchronizationManagerBean.synchronizeData() is a method which starts actual synchronization process. This method is also responsible for updating entries in the history table in the DB. This method logs the following debug statement as “Data synchronization called” **(Added New DEBUG_MIN statement)**

```
<Trace Level="MIN">
<Time Millis="1297955672046"> 2011.02.17 20:44:32.046+05:30</Time>
<Server Format="IP">itimwin3.tivlab.raleigh.ibm.com</Server>
<ProductId>CTGIM</ProductId>
<Component>com.ibm.itim.apps.ejb.adhocreport.synchronization</Component>
<ProductInstance>server1</ProductInstance>
<LogText><![CDATA[Data synchronization called ...]]></LogText>
<Source FileName="com.ibm.itim.apps.ejb.adhocreport.synchronization.AdhocSynchronizationManagerBean"
    Method="synchronizeData"/>
<Thread>Default : 1</Thread>
</Trace>
```

C. ‘Separation of Duty Policy’ behavior is introduced in ITIM 5.1 version. In data synchronization, new methods are added to synchronize SoD policies, rules and violations. SynchronizePolicies.run(DistinguishedName) method executes to synchronize SoD policy by evaluating all rules against all those role members.

D. The actual data synchronization process consists of dropping the existing Entity and ACI tables and recreating and populating them. The ENTITY_COLUMN table contains the information needed to drop, create and populate these tables. The AVAILABLE_FOR_REPORTING field of the ENTITY_COLUMN table contains the state of every attribute that was/is mapped. The different states in which an attribute can exist are as follows:

- NEW: A newly mapped attribute.
- DELETED_NEW: An unmapped NEW attribute.
- AVAILABLE: An attribute available for reporting subject to success of the previous synchronization.

- DELETED_ AVAILABLE: An unmapped attribute previously in an AVAILABLE state.
- TABLE_DROPPED: An intermediate state for synchronization. Only a mapped attribute can attain this state.
- TABLE_CREATED: An intermediate state for synchronization. Only a mapped attribute can attain this state.

AdhocSynchronizationManagerBean.stageData(DistinguishedName, Connection) method stages the ENTITY data to the database. This method will populate the ENTITY tables in the database with data from the ITIM Directory Server. ITIM APIs will be used to fetch the ITIM Directory Server data. In this method it reads 'commitFrequency' property from adhocReporting.properties file. It Checks if the value set for 'commitFrequency' > 1 then the number of uncommited database updates 'numberOfUpdates' are incremented and commit is done everytime when 'numberOfUpdates' is equal to or more than 'commitFrequency'. In this method all database entities like views, tables gets deleted, then database schema is created and tables are populated with data from directory server.

To drop reporting schema, below sequence is followed

Helper.dropReportingViews() – This method drops all reporting views. When all reporting views gets deleted, following debug statement is logged: **(Added New DEBUG_MID statement)**

```
<Trace Level="MID">
 <Time Millis="1297989137640"> 2011.02.18 06:02:17.640+05:30</Time>
 <Server Format="IP">itimwin3.tivlab.raleigh.ibm.com</Server>
 <ProductId>CTGIM</ProductId>
 <Component>com.ibm.itim.adhocreport.util</Component>
 <ProductInstance>server1</ProductInstance>
 <LogText><![CDATA[Dropped reporting views]]></LogText>
 <Source FileName="com.ibm.itim.adhocreport.util.Helper" Method="dropReportingViews"/>
 <Thread>Default : 2</Thread>
</Trace>
```

a) AdhocSynchronizationManagerBean.deleteStagedData(Connection) – This method deletes staged ENTITY data using following methods

a. AdhocSynchronizationManagerBean.deleteDeletedNewTables(Connection) : 'DELETED_NEW' entries of ENTITY_COLUMN table are unmapped new attributes. No processing needs to be done on this attribute during data synchronization. So in this method 'deleted_new' table entries getting deleted from ENTITY_COLUMN table. Following log statement is logged at this stage :

(Converted DEBUG_MAX to DEBUG_MID statement)

<Trace Level="MID">

<Time Millis="1297989291750"> 2011.02.18 06:04:51.750+05:30</Time>

<Server Format="IP">itimwin3.tivlab.raleigh.ibm.com</Server>

<ProductId>CTGIM</ProductId> <Component>com.ibm.itim.apps.ejb.adhocreport.synchronization</Component>

<ProductInstance>server1</ProductInstance>

<LogText><![CDATA[DELETE FROM itimuser.ENTITY_COLUMN WHERE AVAILABLE_FOR_REPORTING = 'deleted_new']]></LogText>

<Source FileName="com.ibm.itim.apps.ejb.adhocreport.synchronization.AdhocSynchronizationManagerBean" Method="deleteDeletedNewTables"/>

<Thread>Default : 2</Thread>

</Trace>

b. AdhocSynchronizationManagerBean.deleteDeletedAvailTables(Connection) : DELETED_ AVAILABLE attribute is an unmapped attribute previously in an AVAILABLE state. During data synchronization, these attribute entries getting deleted from ENTITY_COLUMN table and ENITITY tables created for these attributes also gets deleted. deleteDeletedAvailTables() method deletes entity tables having status 'deleted_available' . Firstly following query is executed to retrieves all the entities which shows status as 'deleted_available' from ENTITY_COLUMN table:

<Trace Level="MAX">

<Time Millis="1297989327406"> 2011.02.18 06:05:27.406+05:30</Time>

<Server Format="IP">itimwin3.tivlab.raleigh.ibm.com</Server>

<ProductId>CTGIM</ProductId>

<Component>com.ibm.itim.apps.ejb.adhocreport.synchronization</Component>

<ProductInstance>server1</ProductInstance>

<LogText><![CDATA[select distinct ENTITY_NAME from itimuser.ENTITY_COLUMN where (AVAILABLE_FOR_REPORTING = 'deleted_available')]]></LogText>

<Source FileName="com.ibm.itim.apps.ejb.adhocreport.synchronization.AdhocSynchronizationManagerBean" Method="getColumns"/>

<Thread>Default : 2</Thread>

</Trace>

c. AdhocSynchronizationManagerBean.deleteTableCreatedTables(Connection) TABLE_CREATED status of entity means that the table has been created but the data is yet to be populated. This method first finds out, which tables need to be deleted. Following query is executed to get entity names from ENTITY_COLUMN table which status as ‘table_created’:

<Trace Level="MAX">
<Time Millis="1297989605687"> 2011.02.18 06:10:05.687+05:30</Time>
<Server Format="IP">itimwin3.tivlab.raleigh.ibm.com</Server>
<ProductId>CTGIM</ProductId>
<Component>com.ibm.itim.apps.ejb.adhocreport.synchronization</Component>
<ProductInstance>server1</ProductInstance>
<LogText><![CDATA[select distinct ENTITY_NAME from itimuser.ENTITY_COLUMN where (AVAILABLE_FOR_REPORTING = 'table_created')]]></LogText>
<Source FileName="com.ibm.itim.apps.ejb.adhocreport.synchronization.AdhocSynchronizationManagerBean" Method="getColumns"/>
<Thread>Default : 2</Thread>
</Trace>

This method drops all child tables created for multivalued attributes and then parent table whose status is ’table_created’. It will also delete ACI related tables associated with these entities. Finally ENTITY_COLUMN table is updated to set status of these entities as ‘table_dropped’.

d. AdhocSynchronizationManagerBean.deleteAvailableTables(Connection): ‘AVAILABLE’ status shows that entity is available for reporting subject to success of the previous synchronization. deleteAvailableTables() method first find out table entities from ENTITY_COLUMN table which are ‘AVAILABLE’ for reporting with following query :

<Trace Level="MAX">
<Time Millis="1297991389625"> 2011.02.18 06:39:49.625+05:30</Time>
<Server Format="IP">itimwin3.tivlab.raleigh.ibm.com</Server>
<ProductId>CTGIM</ProductId>
<Component>com.ibm.itim.apps.ejb.adhocreport.synchronization</Component>
<ProductInstance>server1</ProductInstance>
<LogText><![CDATA[select distinct ENTITY_NAME from itimuser.ENTITY_COLUMN where (AVAILABLE_FOR_REPORTING = 'available')]]></LogText>
<Source FileName="com.ibm.itim.apps.ejb.adhocreport.synchronization.AdhocSynchronizationManagerBean" Method="getColumns"/>
<Thread>Default : 2</Thread>
</Trace>

This method drops all child tables created for multivalued attributes for an entity. It will also delete ACI related tables associated with each entity and finally parent entity table gets deleted table whose status is ‘available’. ENTITY_COLUMN table is updated to set status of these entities as ‘table_dropped’.

E. The next step is to populate all tables. ACI related tables are getting populated first. The ACIs will be stored in the form of relationship tables between the user and the container entries. It store information about the specific access rights in the database. ACISynchronization.populateParsedACIs() method is invoked to populate ACI tables. Following log statement gets logged when population of ACI table starts: **(Added New DEBUG_MIN statement)**

```
<Trace Level="MIN">
 <Time Millis="1297992081359"> 2011.02.18 06:51:21.359+05:30</Time>
<Server Format="IP">itimwin3.tivlab.raleigh.ibm.com</Server>
 <ProductId>CTGIM</ProductId>
<Component>com.ibm.itim.apps.ejb.adhocreport</Component> <ProductInstance>server1</ProductInstance>
 <LogText><![CDATA[Populating parsed ACI related tables...]]></LogText>
<Source FileName="com.ibm.itim.apps.ejb.adhocreport.ACISynchronization" Method="populateParsedACIs"/>
<Thread>Default : 2</Thread>
</Trace>
```

First it flushes out entries from ACI table and its related tables in ACISynchronization.deleteParsedACIEntries(Connection) method. Here data will be deleted from ‘ACI’, ‘ACI_ROLEDNS’, ‘ACI_PRINCIPALS’, ‘ACI_PERMISSION_ATTRIBUTERIGHT’ and ‘ACI_PERMISSION_CLASSRIGHT’ tables. In ACISynchronization.stageParsedACIs() method, it populates the ACI related tables. This method checks if, for an accessright object the filter is NULL or not. If the filter is not NULL then,

1) It gets the list of attributes for this filter. LDAP to SQL filter parser will do the job. Also, the targetClass for this AccessRight object should be used to map it to some entityName.
2) If this entity and the attributes are not present in the ENTITY_COLUMN table, then they get added in ENTITY_COLUMN table.

ACISynchronization.unmapImplicitAttributesNotRequired() method marks all the implicit attributes as "U" in the ENTITY_COLUMN table which were implicitly mapped previously but no longer being used by any ACI in the system.

F. Next step is to populate authorization owners. ACISynchronization.populateAuthorizationOwners() method populate AUTHORIZATION_OWNERS table. This information will be used while generating

ACI Report. When population of AUTHORIZATION_OWNERS table starts, it logs following debug statement : **(Added New DEBUG_MID statement)**

```
<Trace Level="MID">
 <Time Millis="1298015608156"> 2011.02.18 13:23:28.156+05:30</Time>
 <Server Format="IP">itimwin3.tivlab.raleigh.ibm.com</Server>
 <ProductId>CTGIM</ProductId>
 <Component>com.ibm.itim.apps.ejb.adhocreport</Component>
 <ProductInstance>server1</ProductInstance>
 <LogText><![CDATA[Populating AUTHORIZATION_OWNER table...]]></LogText>
 <Source FileName="com.ibm.itim.apps.ejb.adhocreport.ACISynchronization" Method="populateAuthoriza-
tionOwners"/>
 <Thread>Default : 0</Thread>
</Trace>
```

Here it first delete entries from AUTHORIZATION_OWNERS table, then invokes createcachesForAll-Containers() methods which create the cache having details about containers. In supervisorCache HashMap Location/OU container DN is the key and collection of supervisors(SystemUserEntity) are the values. In adminDomainCache AdminDomain/OU/Location/BPO/Org container DN are the key and collection of Domain Administrators(SystemUserEntity) are the values. In containerCache HashMap Parent container DN is the key having value as a collection of children container. After that it gets a list of system users who can do reporting. This is the set of system users, who are covered by reporting ACIs defined in the system. ProcessAORecursive() method is called recursively to populate AUTHORIZATION_OWNERs table for each system user who can do reporting. Following log statement is logged when population of authorization owner table completes: **(Converted DEBUG_MAX to DEBUG_MID statement)**

```
<Trace Level="MID">
 <Time Millis="1298016806687"> 2011.02.18 13:43:26.687+05:30</Time>
 <Server Format="IP">itimwin3.tivlab.raleigh.ibm.com</Server>
 <ProductId>CTGIM</ProductId>
 <Component>com.ibm.itim.apps.ejb.adhocreport.synchronization</Component>
 <ProductInstance>server1</ProductInstance>
<LogText><![CDATA[AUTHORIZATION_OWNER Table Population Took : 1406328
milliseconds]]></LogText>
<Source FileName="com.ibm.itim.apps.ejb.adhocreport.synchronization.AdhocSynchronizationManagerBean"
Method="stageData"/>
 <Thread>Default : 0</Thread>
</Trace>
```

G. If any ACI filter attributes or relationship attributes are missed, these attributes are added in ENTITY_COLUMN table in createFilterACIEntitythatColumnEntry() method.

The next step of data synchronization is to create staged tables. AdhocSynchronizationManagerBean.creatEStagedTables(Connection) method finds out the tables to be created and creates them. This method updates ENTITY_COLUMN table to set status for newly created table as 'created_new'. And then find out what all tables need to be created in getColumansForTableCreation() method. This method also creates ACI related tables like classright and attributeright tables for ACI. While creating tables for data synchronization, some reserve keywords cannot be used. The code reads the list of reserve keywords from the property "reservedWords" from adhocreporting.properties file.

Also if the entity/attribute name contains one or more of the disallowed characters specified in "disallowedChars" in adhocreporting.proeprtiesm then these characters will be removed from the table/column name.

The characters specified in "disallowedCharsForStart" in adhocreporting.properties will not be used as the starting character of table/column name.

H. If change log is enabled in adhocReporting.properties method, then Helper.setChangeNumber() method, sets the changeNumber of the last processed changelog entry in the database (CHANGELOG table).

I. **Populating database tables** depending on ENTITY_COLUMN table status. AdhocSynchronizationManagerBean.populateDatabase() method is invoked to populate database.

i. If the value of 'availableForNonAdministrators' property is set to true, then ACI permissions related data is synchronized. Property value, 'availableForNonAdministrators = false' indicate that reporting data will be exclusively used by administrators and hence processing/population of user ACI permissions is unnecessary.

To synchronize ACIs (if 'availableForNonAdministrators = true'), it first finds out all target classes from ENTITY_COLUMN table (according to which entities are available in ENTITY_COLUMN table). It then creates the accessCache for a container having accessRightsCollection. The accessCache Format is :-

[(Key) targetClass]

[(Value) HashMap whose format is as follows:-

[(Key) self/supervisor/administrator/roleDN]

[(Value) Collection of AccessRights having the "key" as one of its principals.]

It logs following statement in trace.log:

(Added New DEBUG_MIN statement)

<Trace Level="MIN">

<Time Millis="1298246062687"> 2011.02.21 05:24:22.687+05:30</Time>
<Server Format="IP">itimwin3.tivlab.raleigh.ibm.com</Server>
<ProductId>CTGIM</ProductId>
<Component>com.ibm.itim.apps.ejb.adhocreport.synchronization</Component>
<ProductInstance>server1</ProductInstance>
<LogText><![CDATA[Synchronizing ACIs ... generating dirAccessCache]]></LogText>
<Source FileName="com.ibm.itim.apps.ejb.adhocreport.synchronization.AdhocSynchronizationManager-Bean" Method="populateDatabase"/>
<Thread>Default : 1</Thread>
</Trace>

Next step is to Populate parsed Entitlements – This debug statement is logged which shows synchronization current status - **(Added New DEBUG_MIN statement)**

<Trace Level="MIN">
<Time Millis="1298246789890"> 2011.02.21 05:36:29.890+05:30</Time>
<Server Format="IP">itimwin3.tivlab.raleigh.ibm.com</Server
<ProductId>CTGIM</ProductId>
<Component>com.ibm.itim.apps.ejb.adhocreport.synchronization</Component>
<ProductInstance>server1</ProductInstance>
<LogText><![CDATA[Synchronizing Entitlements ...]]></LogText>
<Source FileName="com.ibm.itim.apps.ejb.adhocreport.synchronization.AdhocSynchronizationManagerBean" Method="populateDatabase"/>
<Thread>Default : 1</Thread>
</Trace>

SynchronizationHelper.populateParsedEntitlements(conn) method populates the parsed entitlement related tables. This method first deletes data from ENTITLEMENT and ENTITLEMENT_PROVISIONINGPARAMS tables. After that it populates provisioning policy entitlements in ENTITLEMENT table and provisioning parameters in ENTITLEMENT_PROVISIONINGPARAMS table. Following Insert SQL statement of ENTITLEMENT table is logged in trace.log :

<Trace Level="MAX">
<Time Millis="1298247581890"> 2011.02.21 05:49:41.890+05:30</Time>
<Server Format="IP">itimwin3.tivlab.raleigh.ibm.com</Server>
<ProductId>CTGIM</ProductId>
<Component>com.ibm.itim.apps.ejb.adhocreport</Component
<ProductInstance>server1</ProductInstance>
<LogText><![CDATA[insert into itimuser.Entitlement (dn, type, servicetargettype, servicetargetname, processDN) values (?, ?, ?, ? , ?)]]></LogText>

<Source FileName="com.ibm.itim.apps.ejb.adhocreport.SynchronizationHelper" Method="populateParsedEntitlements"/>

<Thread>Default : 1</Thread>

</Trace>

Following Insert statement of ENTITLEMENT_PROVISIONINGPARAMS table is logged in trace.log:

<Trace Level="MAX">

<Time Millis="1298247871828"> 2011.02.21 05:54:31.828+05:30</Time>

<Server Format="IP">itimwin3.tivlab.raleigh.ibm.com</Server>

<ProductId>CTGIM</ProductId>

<Component>com.ibm.itim.apps.ejb.adhocreport</Component>

<ProductInstance>server1</ProductInstance>

<LogText><![CDATA[insert into itimuser.Entitlement_ProvisioningParams (dn, name, attributevalue, enforcement, exprType) values (?, ?, ?, ?, ?)]]></LogText>

<Source FileName="com.ibm.itim.apps.ejb.adhocreport.SynchronizationHelper" Method="populateParsedEntitlements"/>

<Thread>Default : 1</Thread>

</Trace>

When it completes with entitlement synchronization, it logs following debug statement

(Added New DEBUG_MID statement)

<Trace Level="MID">

<Time Millis="1298248052093"> 2011.02.21 05:57:32.093+05:30</Time>

<Server Format="IP">itimwin3.tivlab.raleigh.ibm.com</Server>

<ProductId>CTGIM</ProductId>

<Component>com.ibm.itim.apps.ejb.adhocreport</Component>

<ProductInstance>server1</ProductInstance>

<LogText><![CDATA[Populated parsed entitlements related tables.]]></LogText>

<Source FileName="com.ibm.itim.apps.ejb.adhocreport.SynchronizationHelper" Method="populateParsedEntitlements"/>

<Thread>Default : 1</Thread>

</Trace>

ii. For some entity types, the data synchronization can be executed using an old strategy or a new strategy. Under most circumstances, the new strategy yields better performance than the old strategy. The new strategy is supported for the following entity types:

- Accounts (all types, including ITIM accounts)

- Authorization Owners

- Groups (all types, including ITIM groups)

- Organizational Containers (all types)

- People (all types, including Business Partner People)

- Roles (dynamic and static)

- Services

The customer can specify which strategy should be executed for the above entity types via properties in the ReportDataSynchronization.properties file under <ITIM_HOME>/data directory. The supported value for each property is either "old" or "new". For each of these properties, the default value is "old".

accountSynchronizationStrategy=old

authorizationOwnerSynchronizationStrategy=old

groupSynchronizationStrategy=old

organizationalContainerSynchronizationStrategy=old

personSynchronizationStrategy=old

roleSynchronizationStrategy=old

serviceSynchronizationStrategy=old

The method AdhocSynchronizationManagerBean.populateDatabase() consults a utility ReportDataSynchronizationStrategy to determine if any of the entity types should be synchronized using the new strategy. If any entity types should be synchronized using the new strategy, the appropriate entity types are synchronized using the “new” strategy (as described below). After the new synchronization strategy has been performed, the remaining entities are synchronized using "old" strategy (as described below).

iii. Population using “new” strategy

The method AdhocSynchronizationManagerBean.performSynchronizationUsingNewStrategy() consults ReportDataSynchronizationStrategy to determine which entity types should be synchronized using the new strategy. Each entity type has its own specific Synchronizer implementation.

At the beginning of each execution, a DEBUG_MID log will be written to indicate that the new strategy is being used for the entity type.

<Trace Level="MID">

<Time Millis="1337103098135"> 2012.05.15 13:21:38.135-04:00</Time>

<Server Format="IP">itimwin3.tivlab.raleigh.ibm.com</Server

<ProductId>CTGIM</ProductId>

<Component>com.ibm.itim.apps.ejb.adhocreport.synchronization</Component>

<ProductInstance>server1</ProductInstance>

<LogText><Executing new Account synchronization….></LogText>

<Source FileName="com.ibm.itim.apps.ejb.adhocreport.synchronization.AdhocSynchronizationManagerBean" Method=”logSynchronizationUsingNewStrategy"/>

<Thread>Default : 1</Thread>

</Trace>

For each entity type for which the new synchronization strategy should be used, the AdhocSynchronizationManagerBean.performSynchronizationUsingNewStrategy() method constructs an instance of a Synchronizer that is specific for the entity type, and synchronizes the data by calling the synchronize() method. The order in which the synchronizers are invoked is important in cases where interdependencies on cached data exist.

Below is the order of execution for the Synchronizers:

1. AllGroupsSynchronizer

The AllGroupsSynchronizer synchronizes both managed groups as well as ITIM groups. Group synchronization must be performed before account synchronization, because the group member information that is synchronized as part of account synchronization depends on a cache created by the group synchronization. The categories associated with Group synchronization are “Group” and “SystemRole”.

2. AccountsAndGroupMembersSynchronizer

AccountsAndGroupMembersSynchronizer synchronizes Owned and Orphan managed accounts, ITIM accounts and Dummy accounts. Accounts are synchronized along side Group Member synchronization. New account synchronization executes if either account or group synchronization is requested. Account synchronization must be performed if group synchronization is requested, so as to synchronize group member information. The categories associated with Account synchronization are “Account” and “SystemUser”.

3. AllPeopleSynchronizer

AllPeopleSynchronizer synchronizes Person (Person and Business Partner Person) and Dummy Person data. The category associated with People synchronization is “AllPersons”.

4. ServicesSynchronizer

ServicesSynchronizer synchronizes services data. The category associated with service synchronization is “Service”.

5. OrganizationalContainersSynchronizer

OrganizationalContainersSynchronizer synchronizes organizational containers. Additionally, it maintains the organizational container information in an internal cache so that the AuthorizationOwnersSynchronizer does not have to access LDAP again to retrieve the same information. The category associated with organizational container synchronization is “Container”.

6. AuthorizationOwnersSynchronizer

AuthorizationOwnersSynchronizer synchronizes authorization owners.

7. RolesSynchronizer

RolesSynchronizer synchronizes roles. The category associated with Role synchronization is “Role”.

In general, Synchronization adopts the following execution flow:

1. Construct the Synchronizer object. This may internally construct sub synchronizer components such as Dummy synchronizers etc. Each Synchronizer uses its own Extractor, that is specific to its entity type, to extract data from LDAP. The Transformer component of the Synchronizer converts each extracted entry into tabular representation. Transformer contains specific evaluator implementations for evaluating attribute values. The Loader is used to load the transformed data into the database.

2. Log the synchronization start.

<Trace Level="MAX">
<Time Millis="1337103098135"> 2012.05.15 13:31:38.135-04:00</Time>
<Server Format="IP"> itimwin3.tivlab.raleigh.ibm.com </Server>
<ProductId>CTGIM</ProductId>
<Component>com.ibm.itim.adhocreport.synchronization.synchronizer</Component>
<ProductInstance>server1</ProductInstance>
<LogText><![CDATA[Starting synchronization of Orphan Managed Accounts]]></LogText>
<Source FileName="com.ibm.itim.adhocreport.synchronization.synchronizer.AbstractSynchronizer"
Method="logSynchronizationStart"/>
<Thread>Default : 0</Thread>
</Trace>

3. Perform any necessary pre-synchronization work such as dropping entity tables, using the method preSynchronize().

4. Perform the actual synchronization task. The Synchronizer invokes the Extractor's extractFromOrganization() method to extract data for the given entity type for each organization from the ITIM Directory server. The extracted data is then transformed using the

Transformer.transform() method. Finally the transformed data is loaded into the database using the Loader.load() method.

5. Perform any necessary post-synchronization work such as creating primary key constraints and indexes, using the method postSynchronize().

6. If applicable, update the AVAILABLE_FOR_REPORTING status associated with the Synchronizer entities in the Schema table (ENTITY_COLUMN).

7. Perform any necessary cleanup by invoking the cleanup() method.

8. Log the synchronization end.

<Trace Level="MAX">
<Time Millis="1337103098135"> 2012.05.15 13:31:38.135-04:00</Time>
<Server Format="IP"> itimwin3.tivlab.raleigh.ibm.com </Server>
<ProductId>CTGIM</ProductId>
<Component>com.ibm.itim.adhocreport.synchronization.synchronizer</Component>
<ProductInstance>server1</ProductInstance>
<LogText><![CDATA[Ending synchronization of Orphan Managed Accounts]]></LogText>
<Source FileName="com.ibm.itim.adhocreport.synchronization.synchronizer.AbstractSynchronizer"
Method="logSynchronizationEnd"/>
<Thread>Default : 0</Thread>
</Trace>

iv. Population using “old” strategy

1. To populate ENTITY tables, it first finds out all containers in SynchronizationHelper.getAllContainers() method. This method will return a collection of objects in the directory. This Object type definition depends on the category of the entity or DNs of the Container entity object. Then SynchronizationHelper.populateIndexAttributesMap() method creates attribute index hashmap which will be used during Entity population. It creates index hashmap for indexes mentioned in adhocreporting.properties# reportIndexes property.

2. Now according to entity list which is retrieved from ENTITY_COLUMN table, we need to populate each entity. SynchronizationHelper.populateEntity() method populates each entity from ENTITY_COLUMN. First, it sets entity status as ‘available’ in ENITITY_COLUMN table - Following debug statement is logged which shows SQL query to update ENTITY_COLUMN table :

(Converted DEBUG_MAX to DEBUG_MID statement)

<Trace Level="MID">
<Time Millis="1298248756984"> 2011.02.21 06:09:16.984+05:30</Time>
<Server Format="IP">itimwin3.tivlab.raleigh.ibm.com</Server>

<ProductId>CTGIM</ProductId>
<Component>com.ibm.itim.apps.ejb.adhocreport</Component>
<ProductInstance>server1</ProductInstance>
<LogText><![CDATA[UPDATE itimuser.ENTITY_COLUMN SET AVAILABLE_FOR_REPORTING = 'available' WHERE LOWER(ENTITY_NAME) = LOWER('ADFeed')]]></LogText>
<Source FileName="com.ibm.itim.apps.ejb.adhocreport.SynchronizationHelper" Method="populateEntity"/>
<Thread>Default : 1</Thread>
</Trace>

It finds out columns for each entity and separates column list in single valued and multivalued attributes. For any entity, only mapped attributes are returned. But for organizationalContainers all attributes are returned.

3. Each entity is processed in SynchronizationHelper.populateEntity() method . If selected entity is organization container, then it invokes getOrganizationalContainerData() method which retrieve all container objects in given container i.e. organizational unit, location, Business partner organization and admin domain. All other entity data is fetched using SynchronizationHelper.fetchDirectoryData() method. This method fetches the data from the ITIM Directory Server for the given ENTITY. It uses ITIM APIs for fetching the data from Directory Server. If the value of 'normalizeDN' property in adhocreporting.properties is set to 'true', all DN attributes will be normalized during data synchronization.

When data retrieval is complete for each entity, it logs following log statements:

(Converted DEBUG_MAX to DEBUG_MID statement)

<Trace Level="MID">
<Time Millis="1298249274765"> 2011.02.21 06:17:54.765+05:30</Time>
<Server Format="IP">itimwin3.tivlab.raleigh.ibm.com</Server>
<ProductId>CTGIM</ProductId>
<Component>com.ibm.itim.apps.ejb.adhocreport</Component>
<ProductInstance>server1</ProductInstance>
<LogText><![CDATA[Data Population for ADFeed in this container OVER]]></LogText>
<Source FileName="com.ibm.itim.apps.ejb.adhocreport.SynchronizationHelper" Method="processEntity"/>
<Thread>Default : 1</Thread>
</Trace>

<Trace Level="MID">
<Time Millis="1298249283921"> 2011.02.21 06:18:03.921+05:30</Time>
<Server Format="IP">itimwin3.tivlab.raleigh.ibm.com</Server>
<ProductId>CTGIM</ProductId>
<Component>com.ibm.itim.apps.ejb.adhocreport</Component>
<ProductInstance>server1</ProductInstance>
<LogText><![CDATA[0 data populated in msecs: 4625]]></LogText>
<Source FileName="com.ibm.itim.apps.ejb.adhocreport.SynchronizationHelper" Method="processEntity"/>
<Thread>Default : 1</Thread>

</Trace>

4. According to entity columns, for single valued attribute parent table of each entity is populated.

If there is any multivalued attribute present, child table for this entity is populated accordingly. 'sql-BatchSize' property from adhocreporting.properties, is used to insert number of records in one SQL Batch.

5. If value for 'createIndex' property from adhocreporting.properties is set to' true' then, indexes have been created for entity tables. Following debug statement is logged as this stage:

<Trace Level="MAX">
<Time Millis="1298250299015"> 2011.02.21 06:34:59.015+05:30</Time>
<Server Format="IP">itimwin3.tivlab.raleigh.ibm.com</Server>
<ProductId>CTGIM</ProductId>
<Component>com.ibm.itim.apps.ejb.adhocreport</Component>
<ProductInstance>server1</ProductInstance>
<LogText><![CDATA[create index TIM828186159951847 on ADFeed (CONTAINERDN asc)]]></Log-Text>
<Source FileName="com.ibm.itim.apps.ejb.adhocreport.SynchronizationHelper" Method="createSin-gleIndices"/>
<Thread>Default : 1</Thread>
</Trace>

When entity population completes it logs following debug statement;

(Converted DEBUG_MAX to DEBUG_MID statement)

<Trace Level="MID">
<Time Millis="1298250694640"> 2011.02.21 06:41:34.640+05:30</Time>
<Server Format="IP">itimwin3.tivlab.raleigh.ibm.com</Server>
<ProductId>CTGIM</ProductId>
<Component>com.ibm.itim.apps.ejb.adhocreport.synchronization</Component>
<ProductInstance>server1</ProductInstance>
<LogText><![CDATA[Entity Table Population Took : 1966625 milliseconds]]></LogText>
<Source FileName="com.ibm.itim.apps.ejb.adhocreport.synchronization.AdhocSynchronizationManagerBean" Method="populateDatabase"/>
<Thread>Default : 1</Thread>
</Trace>

v. Once the "new" or "old" synchronization strategy has been performed, SynchronizationHelper.popu-lateServiceAccountInfo() method populates service profiles and their corresponding account profiles in a table called SERVICE_ACCOUNT_MAPPING.

Following debug statement is logged at this stage of synchronization:

(Converted DEBUG_MAX to DEBUG_MID statement)

<Trace Level="MID">
<Time Millis="1298250694640"> 2011.02.21 06:41:34.640+05:30</Time>
<Server Format="IP">itimwin3.tivlab.raleigh.ibm.com</Server>
<ProductId>CTGIM</ProductId>
<Component>com.ibm.itim.apps.ejb.adhocreport.synchronization</Component>

<ProductInstance>server1</ProductInstance>
<LogText><![CDATA[Populating SERVICE_ACCOUNT_MAPPING table ...]]></LogText>
<Source FileName="com.ibm.itim.apps.ejb.adhocreport.synchronization.AdhocSynchronizationManagerBean" Method="populateDatabase"/>
<Thread>Default : 1</Thread>
</Trace>

vi. If Recertification Policies were synchronized using “old” strategy, SynchronizationHelper.populateRecertifierDetails() method is called to populate RECERERTIFIER_DETAILS_INFO table. Following debug statement is logged at this stage of synchronization: **(Added New DEBUG_MID statement)**

<Trace Level="MID">
<Time Millis="1298250694687"> 2011.02.21 06:41:34.687+05:30</Time>
<Server Format="IP">itimwin3.tivlab.raleigh.ibm.com</Server>
<ProductId>CTGIM</ProductId>
<Component>com.ibm.itim.apps.ejb.adhocreport.synchronization</Component>
<ProductInstance>server1</ProductInstance>
<LogText><![CDATA[Populating RECERTIFIER_DETAILS_INFO table ...]]></LogText>
<Source FileName="com.ibm.itim.apps.ejb.adhocreport.synchronization.AdhocSynchronizationManagerBean" Method="populateDatabase"/>
<Thread>Default : 1</Thread>
</Trace>

vii. The next step is to synchronize user access rights. SynchronizationHelper.synchronizeUserRights() method populates all user tables for all entities passed as collection. It logs following debug statements for synchronization of each ACI class rights and user rights:

(Converted DEBUG_MAX to DEBUG_MID statement)

<Trace Level="MID">
<Time Millis="1298251803609"> 2011.02.21 07:00:03.609+05:30</Time>
<Server Format="IP">itimwin3.tivlab.raleigh.ibm.com</Server>
<ProductId>CTGIM</ProductId>
<Component>com.ibm.itim.apps.ejb.adhocreport</Component>
<ProductInstance>server1</ProductInstance>
<LogText><![CDATA[INSERT INTO itimuser.classright_ADFeed (SET_ID, USERDN, USER_OWNER, CONTAINERDN, RELATIONSHIP, RELATIONSHIP_SQL) VALUES (?, ?, ?, ?, ?, ?)]]></LogText>
<Source FileName="com.ibm.itim.apps.ejb.adhocreport.ACISynchronization" Method="getClassRightStmt"/>
<Thread>Default : 1</Thread>
</Trace>

<Trace Level="MID">
<Time Millis="1298251805906"> 2011.02.21 07:00:05.906+05:30</Time>
<Server Format="IP">itimwin3.tivlab.raleigh.ibm.com</Server>
<ProductId>CTGIM</ProductId>
<Component>com.ibm.itim.apps.ejb.adhocreport</Component>
<ProductInstance>server1</ProductInstance>

<LogText><![CDATA[INSERT INTO itimuser.attrsright_ADFeed (erattrmapfilename, description, erservices-somapping, erpersonsearchfilter, erevaluatesod, ernamingattribute, namingcontexts, ernamingcontexts, owner, erparent, erpersonprofilename, erservicename, erprerequisite, erurl, eruid, eruseworkflow, SET_ID) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)]]></LogText>
<Source FileName="com.ibm.itim.apps.ejb.adhocreport.ACISynchronization" Method="getPrepStmtFinal"/>
<Thread>Default : 1</Thread>
</Trace>

J. After creation of all entity tables, we need to create reporting views. Helper.createReportingViews() method creates the required views on Reporting Entity table.

K. This completes data synchronization process. Following debug statement is logged which shows total time required for data synchronization :

<Trace Level="MIN">
<Time Millis="1298253746765"> 2011.02.21 07:32:26.765+05:30</Time>
<Server Format="IP">itimwin3.tivlab.raleigh.ibm.com</Server>
<ProductId>CTGIM</ProductId>
<Component>com.ibm.itim.apps.ejb.adhocreport.synchronization</Component>
<ProductInstance>server1</ProductInstance>
<LogText><![CDATA[Synchronization completed in 7882141 milliseconds]]></LogText>
<Source FileName="com.ibm.itim.apps.ejb.adhocreport.synchronization.AdhocSynchronizationManagerBean" Method="stageData"/>
<Thread>Default : 1</Thread>
</Trace>

L. It updates synchronization status in SYNCHRONIZATION_HISTORY table as follows:

<Trace Level="MID">
<Time Millis="1298253746765"> 2011.02.21 07:32:26.765+05:30</Time>
<Server Format="IP">itimwin3.tivlab.raleigh.ibm.com</Server>
<ProductId>CTGIM</ProductId>
<Component>com.ibm.itim.apps.ejb.adhocreport.synchronization</Component>
<ProductInstance>server1</ProductInstance>
<LogText><![CDATA[UPDATE itimuser.SYNCHRONIZATION_HISTORY SET STATUS = ?, STATUS_DE-TAIL = ?, COMPLETED_TIME = 1298253746765 WHERE STARTED_TIME=1298245859484 AND SYNC_ID = 0 - with status = Success and status_detail =]]></LogText>
<Source FileName="com.ibm.itim.apps.ejb.adhocreport.synchronization.AdhocSynchronizationManagerBean" Method="updateHistoryEntry"/>
<Thread>Default : 1</Thread>
</Trace>

M. In AdhocSynchronizationManagerBean.releaseLock() method database lock is released after updating synchronization status. Here following debug statement is logged:

(Added New DEBUG_MID statement)

<Trace Level="MID">
<Time Millis="1298253746765"> 2011.02.21 07:32:26.765+05:30</Time>
<Server Format="IP">itimwin3.tivlab.raleigh.ibm.com</Server>

<ProductId>CTGIM</ProductId>
<Component>com.ibm.itim.apps.ejb.adhocreport.synchronization</Component>
<ProductInstance>server1</ProductInstance>
<LogText><![CDATA[Released lock acquired for data synchronization after updating the status.]]></LogText>
<Source FileName="com.ibm.itim.apps.ejb.adhocreport.synchronization.AdhocSynchronizationManagerBean"
Method="releaseLock"/>
<Thread>Default : 1</Thread>
</Trace>