Advanced Repository Concepts

Presenter's Name
Presenter's Title

Safe Harbor Statement

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions.
The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

Oracle Training Materials – Usage Agreement

Use of this Site ("Site") or Materials constitutes agreement with the following terms and conditions:

1. Oracle Corporation ("Oracle") is pleased to allow its business partner ("Partner") to download and copy the information, documents, and the online training courses (collectively, "Materials") found on this Site. The use of the Materials is restricted to the non-commercial, internal training of the Partner's employees only. The Materials may not be used for training, promotion, or sales to customers or other partners or third parties.

2. All the Materials are trademarks of Oracle and are proprietary information of Oracle. Partner or other third party at no time has any right to resell, redistribute or create derivative works from the Materials.

3. Oracle disclaims any warranties or representations as to the accuracy or completeness of any Materials. Materials are provided "as is" without warranty of any kind, either express or implied, including without limitation warranties of merchantability, fitness for a particular purpose, and non-infringement.

4. Under no circumstances shall Oracle or the Oracle Authorized Delivery Partner be liable for any loss, damage, liability or expense incurred or suffered which is claimed to have resulted from use of this Site of Materials. As a condition of use of the Materials, Partner agrees to indemnify Oracle from and against any and all actions, claims, losses, damages, liabilities and expenses (including reasonable attorneys' fees) arising out of Partner's use of the Materials.

5. Reference materials including but not limited to those identified in the Boot Camp manifest can not be redistributed in any format without Oracle written consent.

Agenda

- ATG Repository Transaction Management
- Developing and Testing a SQL Repository

Learning Objectives

At the end of this lesson, you should be able to:

- Apply transaction management features to ensure data integrity
- Learn about ATG Trasaction support
- Use Transaction demarcation to initialize transaction behavior
- Demarcate sections in JSP Pages using Trasaction droplets
- Understand the role of Two Phase Commit in ATG applications
- Use startSQLRepository to perform various repository operations

Section 1:
Repository Transaction Management

Transaction Overview

- Transaction is a set of actions that are treated as an atomic unit, all or none principle.
- Transaction is defined by ACID properties:
  - Atomicity,
  - Consistency,
  - Isolation,
  - Durability.
- Transaction management is a complex task.
  - Single request might require many actions to be completed.
  - One transactional resource is involved in the transaction.
  - Multiple transactions run within an App server.
- Application servers manage transactions.

Transaction Object

- Each active transaction is represented by a transaction object, which implements javax.transaction.Transaction.
- Transaction object keeps track of its own status such as active, committed, or rolled back.
- It also keeps tracks of resources enlisted with it.
- A transaction object is created when a transaction starts and is discarded when a transaction ends.
- It is associated with a thread. Only one transaction can be associated with a thread at any one time, which is the current transaction.
- A running application may have many transactions at once associated with different threads running in the server.

Transaction Manager

- Transaction manager is a central service which keeps track of transactions and the threads they are associated with.
- It creates the transaction objects and discards them.
- Java Transaction API (JTA) provides two main interfaces for managing transactions:
  - **TransactionManager** is intended to be used by ***application servers*** and provides a full range of methods.
  - **UserTransaction** is intended for use by the ***J2EE applications*** and provides a subset of functionality like begin, commit, and rollback. It cannot suspend, resume, or directly access the transaction object.
- Both managers always operate in the context of the calling thread. Hence a begin would start a new transaction associated with that thread.

ATG Transaction Support

- ATG exposes the JTA transaction managers as nucleus components:
  - /atg/dynamo/transaction/TransactionManager,
  - /atg/dynamo/transaction/UserTransaction.
- ATG TransactionManager typically points to the appropriate TransactionManager implementation based on the application server.
- It keeps track of transactions running in ATG applications and which threads are associated with which transactions.
- ATG exposes the managers to standard J2EE components through the JNDI name:
  - dynamo:/atg/dynamo/transaction/TransactionManager,
  - dynamo:/atg/dynamo/transaction/UserTransaction.

Working with Transactions

- As a developer, there are two ways of working with transactions:
  - Using the Java transaction API and the transaction manager,
  - Transaction demarcation.
- Using the transaction manager directly gives a lot of power to the developer.
- It is harder to implement correctly.
- Transaction demarcation is easier to use and depends on marking sections of code enclosed in some sort of transaction behavior.

Transaction Demarcation

- Transaction demarcation wraps a sequence of actions.
- The demarcation initializes some transactional behavior before the demarcated area begins, then ends the transactional behavior when the demarcated area ends.
- The application server uses these demarcations to determine the appropriate calls to the Transaction Manager object.
- There is a helper method to simplify transaction management.
- It is a recommended best practice unless the functionality provided does not suffice to adequately satisfy the business requirements.

Using Transaction Demarcation

- Transaction demarcation has two key methods:
  - **`begin`**() and **`end`**().
- There are six transaction demarcation modes that can be used to determine transaction boundaries:
  - Required, RequiresNew, NotSupported, Supports, Mandatory, and Never.
- Transaction boundaries are set using constants on the TransactionDemarcation object.
- Transaction demarcation can be used:
  - In Java code programatically,
  - In JSP pages using ATG servlet beans.

Transaction Demarcation Modes

- The six transaction modes are as follows:

Mode	Explanation
REQUIRED	Transaction must be in place. If one is not in place, a new one is started.
REQUIRES_NEW	A new transaction is needed. If one is in place, it is suspended and restarted.
NOT_SUPPORTED	A transaction must not be used. If one is in place, it is suspended and restarted.
SUPPORTS	A transaction, if in place is used. If not, the area is not in a transaction.
MANDATORY	A transaction is required. If no transaction is in place, an exception is thrown.
NEVER	A transaction must not be in place. If it is, an exception is thrown.

Transaction Demarcation Classes

- ATG platform provides two classes:
  - **UserTransactionDemarcation** to be used for basic transaction demarcation. It uses UserTransaction object.
  - **TransactionDemarcation** can be used for fine grained demarcation of code. This uses the TransactionManager object.
- UserTransactionDemarcation's **`begin`**() method only implements REQUIRED transaction Mode.
- TransactionDemarcation's **`begin`**() method takes an argument to specify one of the 6 transaction modes.
- The code must ensure the **`end`**() method is always called, typically by using a final block.
- Both **`begin`**() and **`end`**() methods can throw TransactionDemarcationException().

Example: Use TransactionDemarcation

```
TransactionDemarcation td = new TransactionDemarcation();

try {
   try {
     td.begin(getTransactionManager(),td.REQUIRED);

      . . . create new item in repository…

   }catch (RepositoryException re) {
     getTransactionManager().setRollbackOnly();
   }finally {
     td.end();
   }
}catch (TransactionDemarcationException tde) {
 ... handle the exception ...
}
```

- Example shows the proper use of begin and end using transaction demarcation.

Example: Using UserTransactionDemarcation

```
UserTransactionDemarcation td
                = new UserTransactionDemarcation ();

try {
    try {
          td.begin ();
          ... do transactional work ...
    }finally {
          td.end ();
    }
}catch (TransactionDemarcationException exc) {
    ... handle the exception ...
}
```

- No mode needs to be specified on UserTransactionDemarcation.
- It is always REQUIRED.

Demarcation in Pages (1)

- ATG's DSP tag libraries include several tags that you can use to demarcate transactions in JSPs:
  - **dsp:beginTransaction** initiates a transaction and tracks its status.
  - **dsp:commitTransaction** commits the current transaction.
  - **dsp:rollbackTransaction** rolls back the current transaction.
  - **dsp:demarcateTransaction** begins the transaction, executes one or more operations within the transaction, and then commits the transaction.
  - **dsp:setTrasactionRollbackOnly** marks the current transaction for rollback only.
  - **dsp:transactionStatus** returns the status of the current transaction.

Demarcation in Pages (2)

- In addition to transaction handling tags, ATG has servlet beans for demarcating transactions.
  - **Transaction** Droplet marks the bounds of a transaction within a JSP.
  - **EndTransaction** Droplet commits or rolls back the current operation.

```
<dsp:droplet
       name="/atg/dynamo/transaction/droplet/Transaction">
   <dsp:param name="transAttribute" value="requiresNew"/>
   <dsp:oparam name="output">

     ... portion of page executed in demarcated area ...

   </dsp:oparam>
</dsp:droplet>
```

Marking Transactions for Rollback

- As a result of error condition or exception, an application can determine that the current transaction should be rolled back.
- A good way to do this is to mark the transaction for rollback only. This sets a flag on the transaction that it cannot be committed.
- You can either use the TransactionManager or UserTransaction's methods:
  - **`getStatus`**() gets the status of the transaction,
  - **`getRollbackOnly`**() and **`setRollbackOnly`**() to get/ set rollback flag.
- You can also do this:
  - Using **`dsp:setTransactionRollbackOnly`** tag.
  - By calling transaction demarcation's **`end(true)`** method. The **`true`** indicates that the transaction should be marked for rollback only.

Repositories and Transactions

- The default behavior of the repository API:
  - If a JTA transaction is in progress, all repository changes are part of that transaction, and are committed when it completes.
  - If there is no transaction in place, each SQL repository operation call that modifies the repository occurs in its own transaction.
- If no transaction exists:
  - Begin JTA transaction.
  - Call setPropertyValue.
  - Commit JTA transaction. SQL is issued and changes are committed.
- The SQL repository implements transaction isolation.
  - If an item is accessed in a transaction, it is guaranteed to be up to date at that time.
  - If another transaction changes the item, those changes are not visible in this transaction.

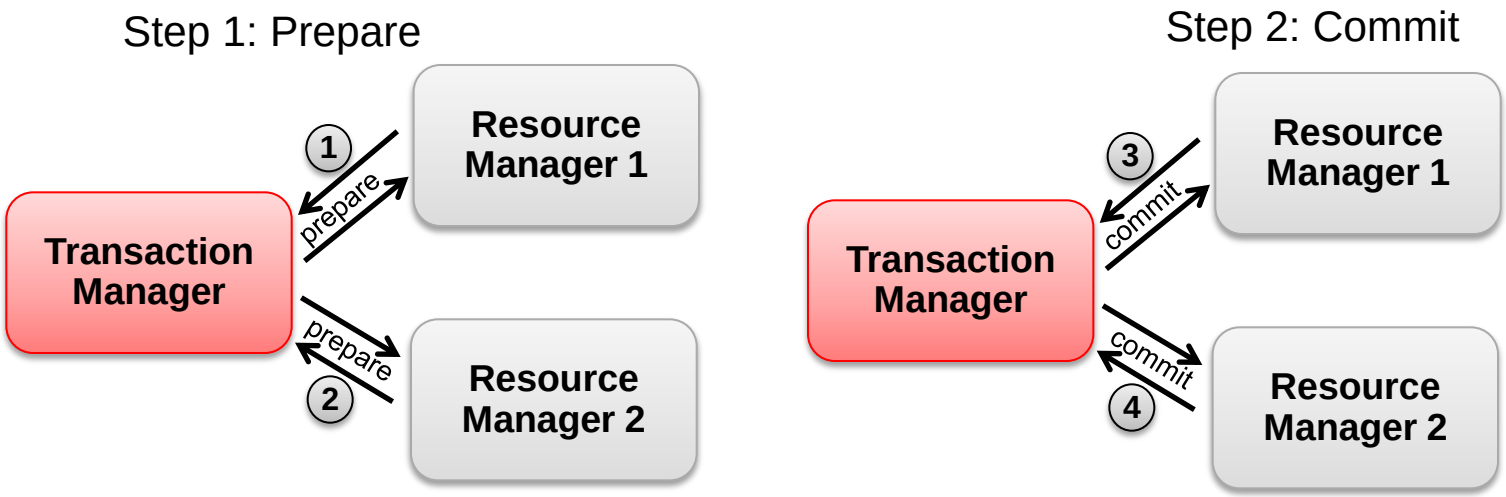
Managing Transactions with Repositories

- Though repositories changes occur by default in their own transaction, it is not optimal.
- Any code making repository changes should create their own transactions for efficiency and consistency.
- ATG provides different methods to manage transactions while dealing with repositories:
  - Use the transaction servlet bean,
  - Use JTA programatically,
  - Use a FormHandler using TransactionalFormHandler.
- Transaction form handler automatically wraps a transaction around all property get method calls and another transaction around all set and handle classes.

Two Phase Commit Protocol (1)

- Two phase commit is required when:
  - More than one resource in involved in a transaction.
  - e.g. ‘More than one database’ or ‘one database and one JMS system.’
- The two phase commit protocol consists of a prepare phase and a commit phase.
  - Phase 1: Prepare
    - Each resource votes to commit or rollback.
  - Phase 2: Commit
    - If all resources voted to commit, then the transaction object issues a commit against each resource.
    - If any resource votes to rollback, then all resources are instructed to rollback.

Two Phase Commit Protocol (2)



- Most ATG applications use and require two phase commit capability.
- Developers and administrators must ensure that a two phase commit capable XA data source and database drivers are used.

Section 1 ✔
Check Your Understanding

Name the ACID Properties that define a transaction.

Answer:

Atomicity, consistency, isolation, durability.

Section 1 ✔
Check Your Understanding

What keeps track of the status of a transaction, whether it is committed, rolled back, or still active?

Answer:

The transaction object.

Section 1 
Check Your Understanding

What is the primary purpose of a transaction manager?

Answer:

Transaction manager is a central service which keeps track of transactions and the threads they are associated with.

Section 1 ✔
Check Your Understanding

What is the difference between transaction manager and UserTransaction manager?

Answer:

UserTransaction is intended for use by the J2EE applications and provides a subset of functionality provided by the transaction manager.

Section 1 ✔
Check Your Understanding

Name the six transaction demarcation modes.

Answer:

Required, RequiresNew, NotSupported, Supports, Mandatory, and Never.

Summary

- Transaction is a set of actions that are treated as an atomic unit, all or none principle.
- Each active transaction is represented by a transaction object.
- Transaction manager is a central service which keeps track of transactions and the threads they are associated with.
- ATG exposes transaction manager as a nucleus component for use by developers.
- ATG developers should be using transaction demarcation as it is simpler and less error prone.

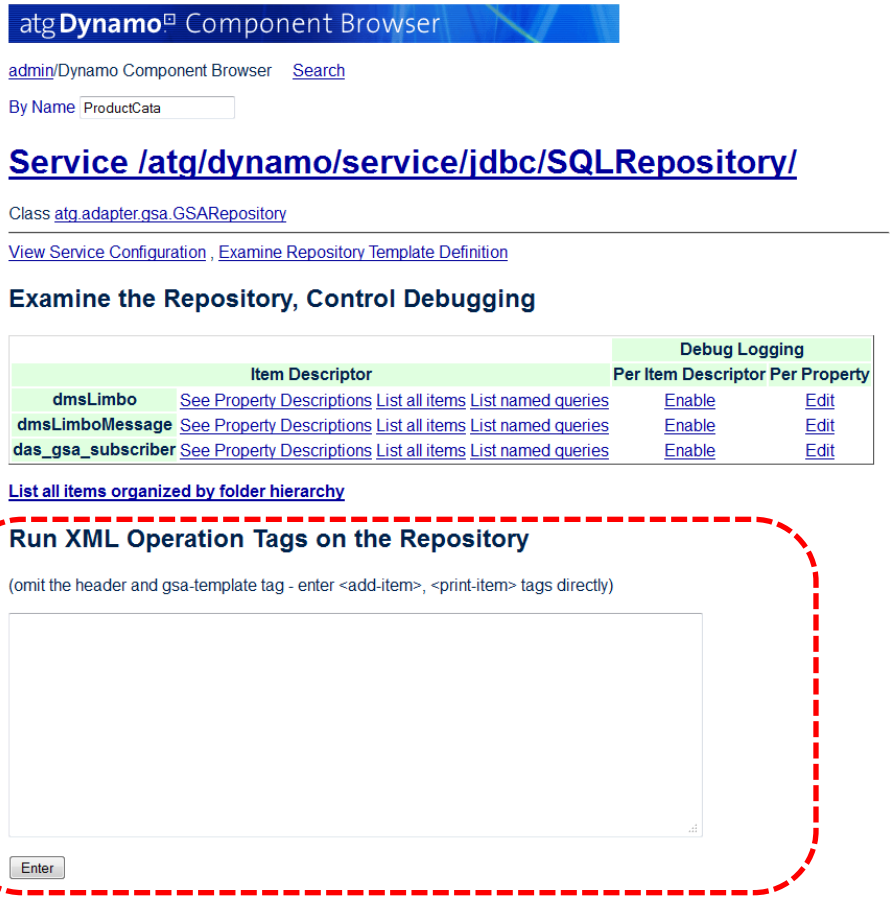
Section 2:
Developing and Testing a SQL Repository

Developing and Testing a SQL Repository

- ATG provides an easy way for developers to create and test SQL repositories.
- The XML document type definition for the SQL repository includes operation tags whose primary purpose is to help you develop, test, import/ export, and debug your SQL repositories.
- What can you do?
  - Add items,
  - Update items,
  - Remove items,
  - Query items,
  - Import/ export items and DDLs.

Use Operation Tags in the Repository Administration Interface

- Developers can use these operations through the Dyn Admin interface.
- In the UI text field, you can enter any XML operations tags against the current repository.
- Click ‘Enter’ and the page displays the output.



URL: http://***host:port***/nucleus/atg/dynamo/service/jdbc/SQLRepository

Query Items

- A repository definition file can include **<query-items>** tags in order to cache query results on application startup.
- Preloading items on startup can improve application performance in some cases.
- The query uses the Repository Query Language (RQL).

```
<query-items item-descriptor="users">
        username="Marty"
</query-items>
```

Add Items

- You can use the **<add-item>** tag to add items to the repository.
- Each <add-item> tag must include an item descriptor attribute.
- Nest <set-property> tags to set property values.
- If you specify the ID of an existing repository item, you update that item.

```
<add-item item-descriptor="users" id="1">
    <set-property name="username" value="Marty"/>
</add-item>
```

Update Items

- Update repository items with the **<update-item>** tag.
- Developers can use the add or remove attributes to add or remove values from multi-item properties without overwriting the whole property value.
- Set skip-update attribute to avoid the update item call until the transaction is committed which has better performance.

```
<update-item item-descriptor="user" id="1"
                                skip-update="true">
    <set-property name="dependents"
                        value="1799" add="true"/>
</update-item>
```

Remove Items

- Remove items from the repository with the **<remove-item>** tag.
- To remove reference items set remove-references-to attribute to true.

```
<remove-item item-descriptor="users" id="1"
        remove-references-to="true"/>
```

startSQLRepository

- startSQLRepository is a SQL multi-utility for developers.
- It can be used for:
  - Verifying the XML is correctly formed and complies with the DTD.
  - Parsing and processing optional operation tags.
  - Generating SQL statements that are required to create the appropriate database table structure.
  - Returning results of <query-items> and <print-item>.
  - Importing from XML to a repository.
  - Exporting repository into XML data.
- It is a command line tool that is part of ATG installation.
- It is located in <ATG_HOME>/bin folder.

startSQLRepository Parameters

- The startSQLRepository scripts take the following arguments:

List of Parameters	List of Parameters
-m startup-module -s server-name -debug vendor -encoding encoding -export "item-type[,...]" file -export all file -exportRepositories repository-path[,...] file -exportRepositories all file	-import input-file -noTransaction -output file -outputSQL Outputs -outputSQLFile file -repository path -skipReferences -verboseSQL

Examples

- This example loads an XML template whose configuration pathname is /atg/test.xml in a repository /atg/userprofiling/ProfileAdapterRepository:

```
bin/startSQLRepository -m DPS
  -repository /atg/userprofiling/ProfileAdapterRepository
   /atg/test.xml
```

- This example exports everything from product catalog repository in DCS module to a file called export-all.xml:

```
bin/startSQLRepository -m DCS -export all export-all.xml
  -repository /atg/commerce/catalog/ProductCatalog
```

Section 2 
Check Your Understanding

What are the possible operations you can perform with the XML document type for the SQL repository?

Answer:

You can add, update, remove, and query the items.

Section 2 ✔
Check Your Understanding

What interface can be used to execute some of the XML operation tags?

Answer:

Dyn Admin can be used to execute the tag operations.

Section 2 
Check Your Understanding

What is query tag used for?

Answer:

To check the repository or to preload the items on startup to improve performance.

Section 2
Check Your Understanding

How do you avoid update item call until the transaction is committed using an update items tag?

Answer:

Set skip-update attribute to true.

Section 2 
Check Your Understanding

Name a few uses of StartSQLRepository.

Answer:

Verify the XML is correctly formed and complies with the DTD, parse and process optional operation tags, etc.

Summary

- The XML document type definition for the SQL repository includes operation tags whose primary purpose is to help you develop, test, import/ export, and debug your SQL repositories.
- Developers can test operations through the Dyn Admin interface.
- A repository definition file can include <query-items> tags in order to cache query results on application startup.
- You can use the <add-item> tag to add items to the repository.
- Update repository items with the <update-item> tag.
- Remove items from the repository with the <remove-item> tag.
- startSQLRepository is a SQL multi-utility for developers.

Q&A

ORACLE IS THE INFORMATION COMPANY


ORACLE®