Frequently Asked Questions

Q: When should I use a reverse relationship?

A:

- It's optional
- It should never be put on both sides: the whole point of it is to avoid repeating yourself.
- If it's a dependency relationship (e.g. order to order item), then on the parent side makes more sense to me. If it's not a dependency relationship (e.g. user to group), then I don't know of a good rule of thumb.
- A relationship with a parameter can't have a reverse relationship.

Q: How do I define a many-to-many relationship that's realized via a third table?

A: You need the third class, but after you create the xml for it, you can reference it in your many- to-many relationship. Here is an example from Reladomo's User.xml (used for testing):

```
<Relationship name="groups" relatedObject="Group" orderBy="name
 DESC"
          cardinality="many-to-
many" reverseRelationshipName="users">
 UserGroup.ownerId = this.id and Group.id = UserGroup.dependentId
</Relationship>
```

Q: Why does Reladomo generate a "between" clause when using an "in" clause?

A: There is usually a performance benefit for Sybase: the proper index will get picked up. In rare cases, it is actually bad. It can be turned off, like so:

```
Operation op = FooFinder.someAttribute().in(set);
if (op instanceof InOperation)
{
    InOperation in = (InOperation) op;
    in.setUseBetweenClause(false);
}
```

Q: Can I use inheritance when two objects have very similar set of attributes?

A: We've dealt with this type of situation using inheritance. We have an abstract XML that defines all the columns and two tiny xml files that are the concrete subtypes. To see an example, have a look at:

- InventoryItem.xml
- Book.xml
- Phone.xml

From an object perspective, that's a reasonable approach as well. It's conceivable that these two objects would have different behavior (e.g. one can be reprocessed, but the other cannot).

Q: How do I update an object?

A: There is no update method. Calling setter methods writes to the database. Put several setters in a transaction if you intend them to be a single unit of work.

Q: How can I generate a primary key before inserting?

A: You can assign it before insertion by using the method: generateAndSet<attribute name>() If you have not called that method, it will automatically be called when calling insert().

Q: Do I have to put every column in the Reladomo XML?

A: You can define only the attributes you need. Caveat: you will not be able to write that object if those attributes are not nullable.

Q: Does Reladomo cache results?

A: Yes, Reladomo will cache its results using soft references if you specify "partial" as the cacheType in the Object XML. If you know you must go to the database, you can:

```
FooList list = new FooList(someOp);
list.setBypassCache(true);
```

If you're using Finder.findOne, there is also: Finder.findOneBypassCache

Q: Does Reladomo support math operations and other SQL functions?

A: There is very limited support for just a couple of functions:

- Double attributes: absoluteValue, plus, minus, times, dividedBy.
- String attributes: startsWith, endsWith, contains, wildCard* (these translate to "like"); toLowerCase

Other than that, there is no support for functions in Reladomo right now. Please note that calling SQL functions usually means the data is not correctly normalized (first rule:all column values must be atomic). Databases also hate these queries because there is typically no way to use an index. If there are no other clauses, the query will almost certainly require a full table scan.

Q: What are the initial values of nullable primitives?

A: Java rules apply here. Think of it this way: you're creating a Java object (not a row in a table). So integers will be zero, booleans will be false, etc. There is a convenience method to set all nullable primitives to null.

Q: What cacheType should I use?

A: cacheType="partial" is by far the best choice for almost any situation. Use cacheType="full" for tiny static tables. cacheType="full" is not supported for dated objects. Reladomo guarantees that in a transaction the cache is updated properly. You will never see a stale object in a transaction. I would advise against using cacheType="none". The "none" option should be reserved for rare cases where updates to the tables in questions are completely out of your control.

Q: If I start a transaction, then create a new Reladomo object and call insert on it. But before committing this transaction, I try to do a lookup for the just created Reladomo object using a Finder's find method; would I get the newly created and inserted object back?

A: Yes, Reladomo will flush the transaction if it has to ensure proper results.

Q: What's echo 2 used for?

A: The Echo 2 libraries are used in the Reladomo middle tier UI as a temporary measure before we unify the Controllers Technology UI framework. Once the new Controllers Technology framework is available, we'll convert our existing web based UI's (control panel, dashboard, etc). The Echo 2 libraries are not meant to be used for any other purpose.

Q: What value do I have to choose for the business date of a newly created object before inserting it?

A: The business date for which you want that object to start it's life. The object will be valid from that business date to infinity. Reladomo will automatically set the FROM and THRU values.

Q: What value do I have to choose for the processing date for a newly created object before inserting it?

A: Infinity. For the processing date dimension, only objects with infinity processing dates may be inserted (or modified).

Q: What happens if I set some values on a dated object and terminate it in the same transaction?

A: In a single transaction, it makes no sense to set a value and then terminate. The atomic unit of work makes it so that only the termination happens. And no, it can't be implemented by inserting a row and immediately chaining out because it'll cause an index failure for good reason (the operation makes no sense).

Q: How do I join across two tables in two different databases? E.g. ReportedTraderPnl and Account.

A: If this was all in the same database, you'd simply add a relationship from ReportedTraderPnl to Account and do:

```
ReportedTraderPnlFinder.account().type().eq("XYZ")
```

The above operation would then join the two tables. However, these are not in the same database, so you cannot do it like that. You'll have to do the join manually.

- Select the TAMS accounts that have the set of interesting accounts and that type:

  ```
  TamsAccountList accountList = new
   TamsAccountList(TamsAccountFinder.type().eq("XYZ"));
  ```

- do the join:

  ```
  ReportedTradePnlList pnlList =
  new
   ReportedTraderPnlList(ReportedTraderPnlFinder.accountId().in(accountList,
   TamsAccountFinder.accountId()));
  ```

Q: How do I get dated objects for all dates?

A: There is an as of attribute operation that is used to get one object per row in the database:

```
FooFinder.businessDate().equalsEdgePoint()
```

This special operation returns one object per row in the database (not one per milestone).

If you use this operation, Reladomo will not add the milestone information that it would normally add to the SQL it generates (e.g. 't1.OUT_Z = '9999-12-01 23:59:00.000'). This means that you can specify the exact dates you want to retrieve objects for, rather than getting a single object using the AS OF date.

You then can add extra conditions, such as

```
FooFinder.businessDateFrom().lessThan(date)
FooFinder.businessDateTo().greaterThanEquals(date)
```

Here is an example of using equalsEdgePoint to get all the objects between 2 dates :

```
Operation op = FooFinder.businessDate().equalsEdgePoint();
if (endDate != infinity)
{
 op =
 op.and(FooFinder.businessDateFrom().lessThanEquals(endDate);
}
op =
 op.and(FooFinder.businessDateTo().greaterThanEquals(startDate));
FooList foos = new FooList(op);
```

foos will have all the foos between the dates. You can play with the equality signs (or the dates) to have open or closed ranges.

The exists() operation can be used in a similar way:

```
ProductList list = new
 ProductList(ProductFinder.scrpCode().eq("ABC").and(ProductFinder.currencySyno
```

In plain English, this says: find products that have an SCRP code of "ABC" which also have at least one currency synonym exists.

Please note that for a x-to-one relationship, exists is the same as "has at least one", and that's how it should be interpreted for an x-to-many relationship. For example, Product has a one-to-many relationship to TipsIndexRatio.

The operation:

```
ProductFinder.tipsIndexRatios().exists()
```

is interpreted as: products that have at least one tipsIndexRatio.

It's because Reladomo guarantees that a persistent object (one that came from the database) exists as a single instance in the VM. In other words == is the same as equals.

But those objects are not "equal". They follow different sets of rules:

- the persistent object will write to the database when a set method is called. It can deleted, but not inserted.

- the non-persistent object will not write to the database when a set method is called. It can be inserted, but not deleted.

Q: When does Reladomo use "distinct" in the generated SQL?

A: All the following must be true for Reladomo to issue distinct:

- There is join (via a relationship)
- The related table is accessed via a set of attributes that does not imply distinctness.

For example, if you have an order with a collection of orderItems (Order.orderId = OrderItem.orderId), then the following operation requires a distinct:

```
OrderFinder.orderItems().status().eq("not shipped") // find all
 orders that have at least one unshipped order item
```

We use distinct here because the join to order items is via the orderId, which is not a unique identifier for order item. If we don't use distinct, we'll get the same order repeated if it has multiple unshipped items.

Q: I need to do some raw JDBC queries before I get to the Reladomo objects and start working with them. Is there a way for me to get a DB connection from the MithraManager so I don't have to create my own connection for the raw JDBC stuff?

A: The connection manager is your object, so feel free to call the getConnection() method on it. Be 100% sure to return the connection back to the pool. In general, you want your code to always look like:

```
Connection con = null;
try
{
    con = connectionManager.getConnection();
    // do some stuff
}
finally
{
    if (con != null)
    {
        con.close();
    }
}
```

Note: This code is not 3-tier safe.

Q: I need to guarantee that the connection manager that I am using for my plain JDBC is the same as that used by Reladomo. I would like to avoid having to mention the connection manager class name twice - once in the Reladomo configuration XML file and once again in the code where plain JDBC stuff is done. We have several connection manager classes and it is possible to refer to 2 different connection manager classes in the 2 locations by mistake.

Is there a way where I can simply ask Reladomo to give me a reference to its connection manager?

A: Every database object keeps a reference to the connection manager. You can get it via:

```
FooFinder.getMithraObjectPortal().getDatabaseObject().getConnectionManager()
```

Please note that this code is not 3-tier safe.

Q: When we do a select all rows from Reladomo, it seems that Reladomo actually tries to load the whole table into memory at the first access to the loaded MithraList object. Is there a way to tune Reladomo so that it will only try to load the table in chunks?

A: There is no real way to do chunks, but you can specify the maximum number of rows to get using the setMaxObjectsToRetrieve method on the list.

A: Moh to verify this answer: Yes, by using the list method forEachWithCursor(). Do not access the list any other way, or the entire list will be forced into memory all at once.

Q: If we use cacheType="partial" then does that allow for the data rows to be changed (say by replication) and the cache to be refreshed dynamically? Here is the scenario:

- We read a few rows from a replicated Account table using Reladomo (with cacheType="partial").
- Replication causes some of those rows to change.
- We try to read some of the rows in #1 (within the same process as #1) using Reladomo.

A: If you're in a transaction, you're guaranteed to get the new data, as long as the object has been marked as "transactional". Read only objects are not refreshed in a transaction. If you're not in a transaction, and the data is in cache, you have several choices:

- You can clear the cache using AmHubAccountFinder.clearQueryCache().

  Or

- You can set a single query to bypass the cache, using either findOneBypassCache, or constructing a list and using setBypassCache(true).

  Or

- Use cacheType="none" and take a serious speed hit.

For objects that are under your control (like the one I helped debug), partial cache is definitely better. If you setup Reladomo's notification feature separate processes that are using Reladomo to write to the database will notify each other of changes, so none of the above will be required.

Q: If I am mostly doing Reladomo lookups on data that can change via replication but I am doing these lookups in transactions, then do I get any benefit from setting cacheType="partial" over cacheType="none"? Seems like in this case your partial cache might degenerate to none cache or how else can it guarantee to provide the new data.

A: The difference between cacheType="partial" and cacheType="none", is that with a partial cache, Reladomo will not have to re-read the same object over and over again in the same transaction; once in a transaction is enough.

Q: Is isProcessingDate="true" a mandatory attribute for "processingDate"?

A: Reladomo tries to guess that using the attribute name, but you should specify it to be 100% sure.

Q: How do Keys and Relationships work in Reladomo? How do you find things?

A: There are two things in Reladomo that catch people off guard, because, frankly, they are somewhat unusual:

- Reladomo is a key-less architecture; that is, there are no key objects. Of course, each object is identified by a primary key (which can be as complex as you like), but there is no encapsulation of the primary key into an object of its own. This immediately raises several questions: How do you find anything? The answer is finders. How do you cache and index? The answer is using TObjectHashStrategy. Reladomo defines a cache as a collection of indices. Indices are not maps, they are actually sets that have two extra properties: they are constructed using a hash strategy; they have an extra get method. (Java sets, curiously, have no get method. The only way to access any object using a regular java set is via iteration, which makes it unsuitable for lookups.) Many people seem to have a difficult time understanding this.

- Reladomo relationships are not pointer based. For example, an order object does not keep a list of order item objects. Of course, the order object still has a getOrderItems() method. To put it differently, the object graph is not physically connected, but the relationships are still present. Relationships are always dynamically resolved via finders. Is this fast enough? Yes, most of the time. There is one situation where we know where it's too slow and we may implement pointer-based relationships at some point in the future. The particular situation arises when a collection of objects is sorted by an attribute of a related object. So for example, if an adjustment history list is being sorted by it's product's description, the getProduct() method gets called O(nLog n) times, which can be very large. For a list of 15000 items, the get method is called around 400,000 times. Even though the get method only takes a few micro (1 millionth) of a second, it's still about 100x slower returning a pointer.

  There are two things in Reladomo that make pointers-less relationships possible: finders and the query cache. Reladomo finders are object oriented. The usability advantages there are obvious, but there is a serious performance benefit as well: no parsing is required to evaluate an operation. The query cache is a map of Reladomo operations to previously found results. The operation knows what it depends on so the query cache can cleverly expire things out of cache.

Q: What is reverseRelationship for?

A: This element in the Relationship clause of the Reladomo object XML is an optional item, which can be used to indicate which side is the 'boss' of the relationship, and so that the backwards relationship and the forewaqrds relationship are both declared in one place. Note the following:

- if it's a dependency relationship (e.g. order to order item), then on the parent (order) side makes more sense. If it's not a dependency relationship (e.g. user to group), then there is no good rule of thumb (maybe don't use it, instead do one - way relationships on each side)

- reverseRelationship should never be put on both sides: the whole point of it is to avoid repeating yourself

- a relationship with a parameter can't have a reverse relationship

Q: What's the life cycle of a Reladomo object?

A: Currently, Reladomo objects have a very simple life cycle. Everything is explicitly inserted or deleted via calling insert() or delete() respectively. This works out fairly well in the scenario below (where a Strategy has many StrategyPositions): The code looks like:

```
StrategyPosition sp = new StrategyPosition();
sp.setX(x);
sp.setY(y);
sp.insert();
```

This is a little ugly, because you have to manually do things like sp.setStrategyId(strategy.getId()); It's not the end of the world, but I think we can do better. Note that you only have to set the primitive attributes and all relationships work automagically as described above. In a very real sense, Reladomo is a thinner layer over the database (at least at the moment) than object manager.

The next time you call strategy.getStrategyPositions(), the query cache knows that the previous answer is no longer valid (because a new strategy position has been inserted) and will take appropriate action to re-resolve that operation. If everything is cached and indexed (as happens in the Object Manager), this is a quick, all in-memory operation. If not, we'll hit the database (or 2nd tier cache). For 50,000 that could be pretty expensive. I have all sorts of ideas about how to make that better, but not a real plan yet. One approach would be to mark certain queries as expensive (e.g. the result is very large) and try to massage the result as relevant events happen (insert, update, delete).

BTW, delete works the same way (there is just no need to construct and populate the object). So the following code works:

```

Strategy s;
StrategyPosition sp = s.getStrategyPositions().get(0);
sp.delete();
```

Calling s.getStrategyPositions() will return a list that no longer contains the deleted element. If you keep a reference to the list that had the strategy position in it, it will not be removed from that list.

In the long run, we want to make certain life cycle handling more closely aligned with the way object manager handles things, which is nicer in many ways, especially because it's closer to Java's understanding of an object's life cycle. Specifically, we want to be able to have the following methods do the right thing:

strategy.getStrategyPositions().add(x): equivalent to insert, but also sets the relevant fields to ensure consistency (e.g. strategyId) prior to insert**strategy.getStrategyPositions().remove(x)**: equivalent to delete The implementation of these methods also creates a fast path for updating our query cache.

Q: How do you set up caching options for each class?

A: Reladomo requires a xml runtime configuration file. See the xsd for possible values.

Q: Can I pass parameters to a relationship getter?

A: We can declare parameterized relationships in the xml and deep fetch them as well. A parameterized relationship takes parameters. It's declared in the xml like so (from Order.xml):

```

<Relationship name="itemForProduct"
          relatedObject="OrderItem"
          cardinality="one-to-one"
          parameters="int productId">
 OrderItem.orderId = this.orderId and OrderItem.productId =
 {productId}
</Relationship>
```

There can be multiple parameters (E.g. parameters="Timestamp asOfDate, String foo, int bar"). Note the curly braces. This is a new addition: you can use curly braces for arbitrary java snippets of code, as well as parameters. The values within the curly braces must be a constant. For example, you can do:

```
{com.example.foo.Bar.SOME_CONSTANT} or {Math.sin(1.4)+17}
```

Note: if your relationship specification uses greater-than or less-than operations, you need to enclose the text in a CDATA section:

```
<![CDATA[
... and EodAcctIfPnl.processingDateFrom > {checkpoint}
]]>
```

The above relationship generates a method on Order that looks like

```
public OrderItem getItemForProduct(int productId)
```

To deep fetch this relationship, you can:

```
OrderList list = new OrderList(...);
list.deepFetch(OrderFinder.itemForProduct(12));
```

So now, list.getOrderAt(0).getItemForProduct(12) will no longer hit the database. This can be used in many places, notably the dated details of product.

Q: How do I debug Reladomo generated sql?

A: Add a line to your log4j configuration:

log4j.logger.com.gs.fw.common.mithra.sqllogs=DEBUG

for batched statements, you will need:

log4j.logger.com.gs.fw.common.mithra.batch.sqllogs=DEBUG

Warning: SQL logging is very expensive in terms of runtime performance. Do not turn these options on unless absolutely necessary.

Note: If you have an 'old' version of Log4J, you may need to change the '.logger.' for '.category.'.

Q: How can I tell what I can use in the xml for my Reladomo-enabled class?

A: Look at the XML schema : mithraobject.xsd The Reladomo test code provides good examples, it is in the source tree at reladomo/test/src Also, copy and paste from other classes!

By providing a reference to the XSD, IDEs can usually help out by providing "code assist" type help. Consider adding a reference to the mithraobject.xsd to the MithraObject element:

```
<MithraObject objectType="transactional"
              xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
              xsi:noNamespaceSchemaLocation="mithraobject.xsd">
```

Q: How do you create many to many relationships utilizing a join table?

A: This is an example:

Table RISK_POINT has a primary key of RISK_POINT.OBJECT_ID. Table DIMENSION has a primary key of DIMENSION.OBJECT_ID. Join table RISK_POINT_DIMENSION has columns of RISK_POINT_OID and DIMENSION_OID, and is used to relate RiskPoint to Dimension.

There is a many-to-many relationship between risk points and dimensions. This is what you do:

Create a Reladomo XML file for each table, e.g. RiskPoint.xml, RiskPointDimension.xml, Dimension.xml. In RiskPointDimension specify 2 attributes - riskPointId that uses RISK_POINT_OID, and dimensionId that uses DIMENSION_OID.

Then in RiskPoint.xml you can specify:

```
<Relationship name="dimensions" relatedObject="Dimension" cardinality="many-
to-many">
RiskPointDimension.riskPointId = this.id and Dimension.id =
 RiskPointDimension.dimensionId
</Relationship>
```

And in Dimension.xml you can specify:

```
<Relationship name="riskPoints" relatedObject="RiskPoint" cardinality="many-
to-many">
RiskPointDimension.dimensionId = this.id and RiskPoint.id =
 RiskPointDimension.riskPointId
</Relationship>
```

Q: Which Reladomo files do I check into my VCS (Version Control System)?

A: The first time Reladomo generates code for a new class, it will also generate 3 (initially empty) concrete classes:

- <MithraClassName>.java
- <MithraClassName>DatabaseObject.java
- <MithraClassName>List.java

All 3 files should be checked into a VCS even if you do not make any changes to them. As a rule of thumb, the Reladomo generated files in the src directory needs to be checked into a VCS, while those generated in the generatedsrc directory do not have to be checked into a VCS.

Q: If I retrieve a Reladomo object (from the DB) and then proceed to do the following, what will happen?

- I set some of the attributes of the object to the same values that they already contain.
- I set some of the attributes of the object first to null and then back to their original values (within the same transaction).

Will Reladomo recognize that no material change has happened to the object in both cases?

A: Case 1 is easy: no change will be done to the object or the database.

For case 2, there is really no way for Reladomo to detect that. In fact, theoretically, it should not be optimized. Consider the following sequence in a transaction:

```
someObj = findSomeObjectByPrimaryKey();
someObj.setSomeAttr(null);
someObjList = findAllObjectsWithAttrNull();
```

The last step depends on the change in the second step. Reladomo detects this and issues the update before doing the last find.

Reladomo does delay database operations as much as possible (to get better performance through batching), but only as long as the results can be guaranteed to be correct. Case #2 would be quite difficult to implement. I suggest you implement it in your code. One way to do it would be to get a temporary copy of the object, make *all* your changes and then set the values on the persistent (non-temporary) object. Reladomo objects have the following method that might help: getNonPersistentCopy()

Q: Does Reladomo have support for schemas?

A: Schemas can be declared in the runtime configuration. That way, it's easier to deploy into various environments (qa, prod, etc) where the schema names may not be the same.

See MithraConfigPartialCache.xml for an example.

Q: I have to do a query to select "distinct accounts" from a Reladomo table. I am pasting some code below to show you what I am doing so you can tell me if I should desist from doing similar stuff or if there is a better way to do this:

```
List marketMakerAccountsList =

 PrimeEntityProductRoleFinder.getMithraObjectPortal().computeFunction
 (PrimeEntityProductRoleFinder.role().eq("MM"),
                                    null,
                                    "distinct " +
 PrimeEntityProductRoleFinder.entityAccountNumber().getColumnName(),
                                    new
 StringResultSetParser());
```

Here "StringResultSetParser" is my own class built on the pattern of "IntegerResultSetParser".

A: We would discourage this kind of use. You can still get distinct accounts if you have a relationship from Prime entity product role to an account object:

AccountFinder.primeEntityProductRole().role().eq("MM")

Q: Why would a Reladomo Finder return the same Reladomo object over and over again in a Reladomo list?

A: This problem stems from a bad primary key. The easiest way to ensure this doesn't happen is to make sure:

- The table has a unique index

- The same columns in the unique index are declared with primaryKey="true" in the Reladomo XML

Q: Is there a way to have the returned list sorted by a particular attribute (the equivalent of "order by" in SQL)?

A:

```
FooList list = new FooList(someOperation);
list.setOrderBy(FooFinder.someAttribute().ascendingOrderBy().
and(FooFinder.someOtherAttribute().descendingOrderBy()));
```

You can even specify order by in a relationship declaration.

Q: Why can't I order by the processing date?

A: Processing date is a virtual attribute and it's typically the same for all objects in the query.

Q: What does the SourceAttribute tag in the Reladomo XML mean?

A: A SourceAttribute is a logical attribute (that is, it's not persisted anywhere) that corresponds to the location of actual storage.

Q: Can I have a single Reladomo object mapped to a table that resides in multiple databases?

Do I need to put the same class under 2 different DB connection managers within the MithraConfiguration.xml file?

A: No, Reladomo has the ability to handle objects of the same type stored in tables residing in different databases. This is supported via the SourceAttribute tag.

Here are the steps to take:

- Add a <SourceAttribute> tag to the xml. Two types of source attributes are supported: string and int. Use string for now.
- Create a connection manager that implements ObjectSourceConnectionManager. The getConnection method now takes a parameter, which is the source attribute of the object in question. Based on this parameter, you should give out one connection or the other.
- All find operations must specify the source attribute as an extra operation.

  E.g. ProductFinder.source().eq("firstDb").and(ProductFinder.cusip().eq("1234"))
- You must set the source attribute on an object before inserting.

Q: Are the following 2 lines of code effectively equivalent (given that I am using JDK 1.5)?

```
MyMithraClassList myObjectList = new
 MyMithraClassList(MyMithraClassFinder.all());

List<MyMithraClass> myObjectList = new
 MyMithraClassList(MyMithraClassFinder.all());
```

A: No, they are not. The second form is just a genericised List, which doesn't have any of the Reladomo methods (e.g. insertAll(), relationship getters, deepFetch, etc)

```
MithraList<MyMithraClass> myObjectList = new
 MyMithraClassList(...);
```

Is a little bit better, as you get some common methods (deepFetch, setOrderBy, etc), but it still is not as good as the specific class, which has methods specific to that object (e.g. relationship getters)

As a rule-of-thumb, use the specific list-class generated by Reladomo for you, so that you always have access to all the appropriate Reladomo methods. The List<T> allows the MithraList to work fairly seamlessly with most libraries with genericised methods, including GS Collections.

Q: Why do my generated Reladomo classes not have an insert() method?

A: By default, a Reladomo object is read-only and methods for insert, update, or delete are not generated. If you wish to generate a transactional object (a Reladomo object that can be inserted, updated, or deleted) then you have to define it explicitly in the Reladomo xml using the objectType attribute in the MithraObject tag.

Example:

```

<MithraObject objectType="transactional">
```

Q: What is the proper way to get everything from a database table if we are using Reladomo? In other words, how do you translate a relational query having no where clause to a Reladomo query?

A:
```
FooList fooList = new FooList(FooFinder.all())
```

Q: We are looking to use Reladomo in some of our feed processing flows. We currently have a ETL library (built in house) called ODIN that can insert input data (from any source) into DB tables. It doesn't do any milestoning though so we need to use Reladomo. Thus, our processing flow would look something like the following:

- Use ODIN to process records input data records into memory
- Pass the "in-memory record" to an adapter that will convert it to the appropriate Reladomo class (that maps to the correct DB table)
- Use Reladomo to insert into the database.

The complication in this scenario is that the input data records reside in a staging table in a database. Happily, Reladomo is supposed to write its output to the same database albeit to a different table. We want to share the DB connection between ODIN and Reladomo in this scenario. We know that ODIN can not accept a DB connection from outside. Can Reladomo be handed a DB connection to use?

A: The short answer is: yes, you can hand the connection in. However, doing so is a little tricky. The good news is, the connection manager is a runtime configuration, so you won't have to make any changes to your Reladomo objects themselves. But you will have to write a custom connection manager for this purpose.

Here is what you need to do:

- Create a connection manager class that can be initialized with a connection (e.g. the one that comes from ODIN).
- the getConnection() method will not return the plain ODIN connection, but a simple wrapper connection around it. The wrapper delegates all methods to the ODIN connection, except for

close(). Reladomo expects the close() method to return the connection back to the pool. So the close method in the wrapper connection would be empty.

In the long run, it would be good to have our entire infrastructure (OBJ, ODIN, etc) understand the concept of a connection pool. That way, different parts of our infrastructure can "play nice" and share the same connection by the virtue of having the same connection manager.

Q: How can I get a handle to the Reladomo DB connection manager in my client code so I can set its connection attribute?

A: That's up to you. You control the connection manager. A singleton pattern works well in these situations.

When Reladomo reads the configuration xml file, it does the following:

- call the static method "getInstance()" on the configured connection manager
- instantiate the database objects for all configured objects and set the connection manager (that was gotten in 1) on the database object

To get a reference to the connection manager, you should therefore call the same getInstance() method.

Q: Is there any way to configure Reladomo such that it will accept FROM_Z and THRU_Z columns with column type "date" in the database (instead of datetime)?

A: That's not possible with the current code and it would be a significant departure from the current architecture.

Q: Is there a published XML schema for Reladomo (or even better, a sample XML document that includes all elements with comments to explain which are mandatory / optional and what they are for)?

A: mithraobject.xsd is the schema definition for the MithraObject xml.

It is well documented and it does define mandatory and possible values. In addition, there is documentation on the mithraobject.xsd

Q: What does the following tag means?

<SuperClass name = "AbstractParaTransaction" generated="true"/>

A: It means two things:

- There is an "AbstractParaTransaction.xml" that defines AbstractParaTransaction
- This class extends AbstractParaTransaction and therefore inherits everything in it. Subclasses have an option to override attribute and relationship definitions in a compatible way.

Q: What does the superclassType attribute in the MithraObject tag mean?

A: An object's super class type can be "table-per-subclass" or "table-for-all-subclasses". For table-per-subclass objects, all the attributes of the object must appear in each table.

For table-for-all-subclasses, all objects live in the same table are differentiated typically by some attribute. It is up to the implementation to override the database object's createObject method.

Q: How does table-for-all-subclasses work?

A: Table for all subclasses works like this:

- Add superClassType="table-for-all-subclasses" to the object definition
- Create (in your IDE) subclasses of the concrete class. For example, if you have a class called Trade and you want two subclasses, SecurityTrade and ContractualTrade, you create those two file like any normal java class, subclassing Trade.
- After regenerating the code, there will be a new abstract method in the abstract database object that you must implement in your concrete database object. It'll look like this:

```
protected Trade constructTrade(MithraDataObject data)
```

Typically, you'd implement the method along the lines of:

```
TradeData tradeData = (TradeData) data;
if (tradeData.getType().equals(“SEC”)) // this can be any
 condition based on the data for this object
{
    return new SecurityTrade();
}
else if (tradeData.getType().equals(“CON”))
{
    return new ContractualTrade();
}
```

Q: A Model has a list of account IDs for which it is Observable, but we don't want to have relationships with account, since model is living in a different database. Instead, we just want to persist account IDs as Strings. So the table for them would be model ID - account ID. And Model would have method List getObservableAccountIds()

A: If I understand this correctly, you have an object called Model that corresponds to a MODEL table and another object called ModelAccount that corresponds to MODEL_ACCOUNT table (with just model id, account id in it). There is a natural relationship between these two objects: model.getModelAccounts() would return a list of ModelAccount objects. If you want just the account ID's, you'll have to write a trivial loop:

```
public List getObservableAccountIds()
{
    List result = new ArrayList();
    for(Iterator it = this.getModelAccounts().iterator();
 it.hasNext(); )
    {
        result.add(it.next().getAccountId());
    }
    return result;
}
```

Q: How can I enable timezone conversion in Reladomo?

A: You can enable UTC time for a timestamp or AsOf attribute by adding: timezoneConversion="convert-to-utc" or timezoneConversion="convert-to-database-timezone" to the attribute declaration.

The date will be stored as UTC in the database. I suggest you put UTC in the column name to avoid confusion (e.g. IN_UTC, OUT_UTC). The conversion is done at points of entry and exit into the VM. The infinity date is treated specially: it must be created in the VM's local timezone and Reladomo will take care not to convert it. This allows using the same infinity date for both business and processing dates. There is no need to do manual conversions for querying, inserts, updates or deletes.

Please don't create relationships between different types of dates. It will break.

Q: How do I handle timezone conversion using Reladomo?

A: Select the right conversion type in the xml definition. From then on, don't worry about it and imagine you live in one timezone.

Under the covers, this is what happens:

```
now = new Timestamp(System.currentTimeMillis());  // e.g.
 2005-10-10 1:05 pm EST
```

Like all java dates, this is internally stored as a long, which is in UTC

```
checkpoint.setTimestamp(now)
```

Nothing interesting happens in the VM at this point. When we create the SQL statement, we make sure we do:

```
update CHECKPOINT set TIMESTAMP='2005-10-10 6:05 pm'
```

The thing to understand about the database is that you can't issue the statement:

```
update CHECKPOINT set TIMESTAMP='2005-10-10 6:05 pm UTC'
```

Because the database never stored timezones. That's why I recommend you put "UTC" in the column name.

Q: How can I do Reladomo and non-Reladomo database operations in the same transaction?

A: You can do it. You need to flush the Reladomo transaction using the method executeBufferedOperations().

Here is an example :

```
MithraManager.getInstance().executeTransactionalCommand(new
 TransactionalCommand() {

    public Object executeTransaction(MithraTransaction
 mithraTransaction) throws Throwable {

        harsha.setFirstName(newName);
        mithraTransaction.executeBufferedOperations();  //
 required to flush the Reladomo transaction
```

```
        Connection conn = null;

        try
        {
            conn = ((ObjectSourceConnectionManager)
 EmployeeFinder.getMithraObjectPortal().getDatabaseObject().
             getConnectionManager()).getConnection("NYC");
            CallableStatement cstmt = conn.prepareCall(
            " { call pri_tne_employee_card (?,?,?,?) } ");

            cstmt.setString(1, "123456789");
            cstmt.setString(2, harsha.getEmployeeId());
            cstmt.setString(3, "AMXBAN");
            cstmt.setString(4, "10-Jan-2008");

            cstmt.execute();
        }
        finally
        {
            if(conn != null) conn.close();
        }

        return null;
    }
});
```

Q: Is there a Gradle plug-in for various Reladomo build tasks (generate code, graphs, etc)?

A: No. Gradle supports ant tasks natively; use Reladomo's ant tasks directly in gradle.

For example :

```
apply plugin: 'java'

// Make sure apply plugin: 'java' is present
configurations {
  mithraGenConfig
}

dependencies {
mithraGenConfig "com.gs.mithra:mithra:${versions.mithra}"
  mithraGenConfig "com.gs.mithra:mithragen:${versions.mithra}"
}

task mithraGenerateSources<< {
  ant.taskdef(name: 'mithraGen',
 classname:'com.gs.fw.common.mithra.generator.MithraGenerator',
          loaderRef: 'mithraGenerator',
          classpath: configurations.mithraGenConfig.asPath,
  )
  ant.mithraGen(
          xml:'src/main/resources/Mithra/database/
MithraClassList.xml',
          generatedDir:'generated/java',
          nonGeneratedDir:'src/main/java',
          generateConcreteClasses:'true',
          generateGscListMethod:'true'
  )
}

task mithraClean {
  delete 'generated/java'
}

clean.dependsOn(mithraClean)

build.dependsOn(mithraGenerateSources)

compileJava.dependsOn mithraGenerateSources
sourceSets {
  main {
    java {
      srcDir 'generated/java'
    }
  }
}
```

Q: Is there a Maven plug-in to generate the Reladomo classes / DDL / GraphML-model?

A: No. Use the Maven Antrun plugin instead.

Here is an example:

```

<properties>
    <mithra-version>15.4.2</mithra-version>
</properties>

<dependencies>
    <dependency>
        <groupId>com.gs.mithra</groupId>
        <artifactId>mithra</artifactId>
        <version>${mithra.version}</version>
    </dependency>

    <dependency>
        <groupId>com.gs.mithra</groupId>
        <artifactId>mithragen</artifactId>
        <version>${mithra.version}</version>
    </dependency>

    <dependency>
        <groupId>com.gs.mithra</groupId>
        <artifactId>mithragendb</artifactId>
        <version>${mithra.version}</version>
        <scope>compile</scope>
    </dependency>
</dependencies>

<plugin>
   <groupId>org.apache.maven.plugins</groupId>
   <artifactId>maven-antrun-plugin</artifactId>
   <executions>
       <execution>
           <id>generateMithra</id>
           <phase>generate-sources</phase>
           <goals>
               <goal>run</goal>
           </goals>
           <configuration>
               <tasks>
                   <echo>Start of Reladomo code generation</echo>
                   <taskdef name="mithra-gen"

  classname="com.gs.fw.common.mithra.generator.MithraGenerator"
                            loaderRef="mithraGenerator">

  <classpath refid="maven.compile.classpath"/>
                   </taskdef>
                   <mithra-gen xml="${basedir}/src/main/
resources/path-to-your-mithra-xml-files/FullMithraClassList.xml"
                               generatedDir="${basedir}/target/
mithra/generated/src/main/java"
                               nonGeneratedDir="${basedir}/src/
main/java"
                               generateConcreteClasses="true">
                   </mithra-gen>
               </tasks>
               <echo>End of Reladomo code generation</echo>
           </configuration>
       </execution>
   </executions>
</plugin>

<plugin>
    <groupId>org.codehaus.mojo</groupId>
    <artifactId>build-helper-maven-plugin</artifactId>
    <executions>
        <execution>
            <id>add-source</id>
            <phase>generate-sources</phase>
            <goals>
                <goal>add-source</goal>
            </goals>
            <configuration>
                <sources>
                    <source>src/main/java</source>
                    <source>target/mithra/generated/src/main/
java</source>
```

```
                </sources>
            </configuration>
        </execution>
    </executions>
</plugin>
```

Q: Is there a Maven plug-in to reverse-engineer Reladomo object XML files from an existing database schema?

A: No. Use the Maven Antrun plugin instead.

For more details, see the Maven AntRun [http://maven.apache.org/plugins/maven-antrun-plugin/usage.html] documentation.

Q: How can I work with Stored Procedures in Reladomo?

A: Short answer: You can't.

Long answer:

- Stored procedures are considered an anti-pattern. They are not supported by Reladomo, and never will be.
- When working with Stored Procedures, the core facilities of Reladomo are no longer meaningful:
  - Finders don't make sense
  - "transactional" could not be supported
  - Relationships become impossible (no deep fetch, no search)
  - Caching is mostly meaningless
  - Aggregation over is impossible
- There is no easy way to test a Stored Procedure
- Stored procedures in Reladomo would introduce problems with the having duplicate/inconsistent values of objects in memory. Reladomo ensures that can’t happen, but if this stored procedure is reading some data that Reladomo has control over, there is a good chance for inconsistency.

Q: Why does setMaxObjectsToRetrieve() sometimes work, and sometimes doesn't?

A: This isn’t the first time this has come up. We could make them consistent, but the point of setMaxObjectsToRetrieve is to reduce IO, not artificially limit the length of the list. Obviously, with a cache hit, there is no point in reducing IO. The fact that you’re noticing this implies you’re retrieving this information anyway, so why are you limiting it in one spot, but not the other? Queries that use setMaxObjectsToRetrieve will not get cached, so your inconsistent usage is actually hurting the cache performance.

Q: Can you bulk-insert to a table with nullable columns if those columns are not mapped in Reladomo?

A: For bulk insert, all columns must be mapped.

Q: Why can't we bulk-insert varchar columns bigger than 256?

A: The BCP protocol that bulk-insert is implemented upon is old, and simply doesn't support varchar columns larger than 256.

Q: What schema spaces can Reladomo use to create temporary tables? The runtime uses an application ID that is part of DACT_RW

A: By default Reladomo is configured to create temporary tables in SESSION. Your user ID must be granted create/drop table permissions for SESSION.

You can ask your DBA to give create table permission to DACT_RW on the SESSION schema (One needs to grant creattab on database to dact_rw group; createin, alterin, dropin schema permissions on SESSION schema to dact_rw group; and use of user tablespace(s) to dact_rw group).

Or, you can create a schema for Reladomo's temporary usage, with proper permissions similar to above. You then have to create a subclass of Udb82DatabaseType that overrides all the methods that deal with SESSION (getTableNameForNonSharedTempTable, getSqlPrefixForNonSharedTempTableCreation, appendSharedTempTableCreatePreamble, createIndexSql). Use your subclass in your connection manager.

Or, you can use a different schema (as of Reladomo 12.5.0) for temp tables by calling Udb82DatabaseType.setTempSchemaName

Q: I have multiple JVMs running Reladomo and accessing the same database, some reading, some writing, and some doing both. How do I keep the caches up-to-date?

A: Reladomo's TCP Notification Server is what you need, and easy to set up and try out.

- (a) Setup the notification server. It should be something like:

```
java -classpath < all_the_required_jars > -
Dport=< some_port_number >
 com.gs.fw.common.mithra.notification.server.NotificationServer
```

- (b) In the processes that read or write Reladomo objects, configure them for notification:

```
MithraManagerProvider.getMithraManager().setNotificationEventManager(
    new MithraNotificationEventManagerImpl(new
 TcpMessagingAdapterFactory(host, port)));
```

- The host and port in (b) should point to where (a) is running

Q: I have set up a notification server -- how do I get transparency into what is happening?

A: Set the notification logging level to "DEBUG" on each of the VMs.

```
com.gs.fw.common.mithra.notification
```

Q: Where can I get training for Reladomo?

A: Look at all the documentation released with Reladomo.

Also take a look at the Reladomo Kata which is a combination of presentation material and hands-on exercises designed to get you working with the Reladomo API. This is offered as a solo, self-paced course, or with notice, can be given as a GSU-style training session to a group of developers at one-time.

Q: How can I test if my deep-fetch is working?

A: You need to verify that the number of calls you make to a DB are as many as you expect.

```
// deep-fetch a list
EmployeeList employees =
 EmployeeFinder.findMany(EmployeeFinder.all());
employees.deepFetch(EmployeeFinder.demographics());
employees.deepFetch(EmployeeFinder.personalInfo().addresses());
// ensure the resulting list is resolved, e.g. by calling
 forceResolve()
// verify we got the expected number of employees
assertEquals(15, employees.size());
// Now grab a count of DB retrieves so far:
int count =
 MithraManagerProvider.getMithraManager().getDatabaseRetrieveCount();
// do something interesting that exercises the list you fetched,
 e.g.
int totalAddressCount = 0; // (there's better ways of getting
 this, but this is just a made-up example)
for (Employee employee : employees)
{
    totalAddressCount += employee.getAddresses().size();
}
// if your deep-fetches were good, then there should not have
 been any additional retrieves made, so:
assertEquals(count,
 MithraManagerProvider.getMithraManager().getDatabaseRetrieveCount());
```

Q: What connection manager should I use?

A: You only have two choices:

- XaConnectionManager that comes with Reladomo
- Jolt + its connection manager

You must not use some random connection manager with Reladomo.

Here is how to use XaConnectionManager (which is far more advanced than the pool in jndi-jdbc):

```
connectionManager = new XAConnectionManager();
connectionManager.setLdapName(ldapName);
connectionManager.setJdbcUser(user);
connectionManager.setJdbcPassword(password);
connectionManager.setDefaultSchemaName(schemaName);
connectionManager.setPoolName(ldapName + " connection pool");
connectionManager.setInitialSize(1);
connectionManager.setPoolSize(100);
connectionManager.setUseStatementPooling(true);
connectionManager.setProperty("com.gs.fw.aig.jdbc.global.DataSourceImpl", "com
connectionManager.setProperty("com.gs.fw.aig.jdbc.global.ConnectionPoolDataSou
connectionManager.setProperty("com.sybase.jdbc4.jdbc.SybDataSource.REPEAT_READ
connectionManager.initialisePool();
```

and your SourcelessConnection manager will hold that object and implement:

```
public Connection getConnection()
{
    return this.connectionManager.getConnection();
}
```

Q: Is there a way that I can use Spring framework for transaction management?

A: Although this is relatively straight forward to do (by providing implementation of a few Spring interfaces that delegate to Reladomo), we strongly recommend against this.

Spring has no notion of full code re-execution on deadlock/timeout/optimistic lock failure. That's a significant reliability concern for production applications.

Transaction management with Reladomo is simple, flexible and industrial strength. See the documentation for MithraManager.executeTransactionalCommand and the TransactionStyle object.

There is not a significant need to use an IoC container with anything that Reladomo needs. For individual apps, that’s really up to each app; however, don’t mix the concerns of transaction management with the rest of what IoC containers provide.

Q: How do I "restore" a previous version of a audit-only dated object?

A: You need to assign the values of a old object to the current one, or create a new one if there is no current instance:

```
Foo oldFoo = ... // find it with
 processingDate().eq(restoreFromTime)
Foo currentFoo = ... // find it with [implicit]
 processingDate().eq(infinity)
if (currentfoo == null)
{
    currentFoo = new Foo();
    currentFoo.copyNonPrimaryKeyValuesFrom(oldFoo);
    currentFoo.setProcessingDateFrom(null);
    currentFoo.setProcessingDateTo(null);
    currentFoo.insert();
}
else
{
    currentFoo.copyNonPrimaryKeyValuesFrom(oldFoo);
}
```

Q: How do I perform a case-insensitive search of a column? Should I use wildCardEq()?

A:

```
// Reladomo supports toLowerCase() and contains() on attributes:
description().toLowerCase().contains(searchString.toLowerCase());

// The following also works, but is less elegant:
```

```
description().toLowerCase().wildCardEq("*" +
 searchString.toLowerCase() + "*");
```

Q: I am using DB2 and performance seems poor, particularly of prepared statements, and we don't think the indexes are being used. What's the problem?

A: The basic configuration of DB2 makes some bad choices as its defaults, particularly around optimization of prepared statements. You may need the following lines in your connection manager configuration to tell DB2 to behave the right way:

```
// Performance enhancement to tell DB2 to take distribution
 statistics into account
// for query parameters, and to re-optimize parameterized queries
String REOPT_OPTION = "NULLIDR1";
this.connectionManager.setProperty("currentPackageSet",
 REOPT_OPTION);
this.connectionManager.setProperty("jdbcCollection",
 REOPT_OPTION);
```

You will also need to ensure that your DBAs have installed the "NULLIDR1" and "NULLIDRA" packages on your DB2 instance.

Note: If REOPT_OPTION = "NULLIDR1" doesn't work for you, try "NULLIDRA" instead.

Q: Why does Reladomo use IntHashSet instead of Set<Integer>?

A: Primitives are far superior to boxed objects, e.g. int vs Integer. Ideally, IntHashSet would be part of the JDK.

Reladomo uses GS Collections IntHashSet because:

- The runtime performance is much better
- IntSet has a much lower memory footprint than UnifiedSet<Integer>, which is in turn much better than JDK's HashSet, and moreover, has serious repercussions for cache coherency, locality and GC pauses

Reladomo encourages usage of the efficient IntHashSet by providing the API to use it with finder operations, e.g. in(IntHashSet), notIn(IntHashSet). Reladomo discourages usage of the inefficient Set implementations by not providing direct support in it's API for them.

Q: Which repos can I get Reladomo from?

A: Maven central.

Q: How can I tell if Reladomo's queries match up to the indexes I have declared on my database tables?

A: Use the Reladomo Index Reconciliation Tool: this tool verifies your XML is consistent and well-formed, and validates Reladomo object dependencies. Additionally, it will verify that your DB tables and indexes match what Reladomo expects.

Reladomo knows what attributes are used to define relationships between objects, and hence can tell you what indices it needs to perform efficiently.

Class name is DatabaseIndexValidator. It has a main method with usage example. It can also be run as ant target. build.xml in Reladomo has ant target usage example:

```
<target name="validate-mithra-db-indices-test" depends="compile-
generate-mithra-xml">

    <taskdef name="mithra-validate-xml"

  classname="com.gs.fw.common.mithra.generator.objectxmlgenerator.DatabaseInde
            loaderRef="indexValidator">
        <classpath refid="mithraxmlgen.classpath"/>
    </taskdef>

    <mkdir dir ="${mithra.home}/build/mithra/tmp/generated"/>

    <mithra-validate-
xml userName="username" password="password" databaseType="sybase"
            xml="${mithra.home}/xml/mithra/test/maxlengenerator/
TestClassList.xml"
            driver="com.sybase.jdbc4.jdbc.SybDriver"
            url="jdbc:sybase:Tds:host:port/schema"
            schema="some_schema" />
</target>
```

Q: How can I verify if the Reladomo xmls which I wrote match up the database table definitions?

A: There are 4 utilities which can be used. These utilities verify if your XML is consistent and well-formed, and validates Reladomo object dependencies.

All the classes viz. DatabaseTableValidator, DatabaseIndexValidator, MaxLenValidator, NullableColumnValidator have a main method with usage example. These can also be run as ant target. The build.xml in Reladomo has ant target usage example.

- DatabaseTableValidator: This is used to verify the database columns and their data types.

```
<target name="validate-mithra-xml-test">
    <taskdef name="mithra-validate-
xml" classname="com.gs.fw.common.mithra.generator.objectxmlgenerator.Databas
        <classpath refid="mithraxmlgen.classpath"/>
    </taskdef>
    <mkdir dir ="${mithra.home}/build/mithra/tmp/generated"/>
    <mithra-validate-
xml userName="username" password="password" databaseType="sybase"
            xml="${mithra.home}/xml/mithra/test/maxlengenerator/
TestClassList.xml"
            driver="com.sybase.jdbc4.jdbc.SybDriver"
            url="jdbc:sybase:Tds:host:port/schema"
            schema="some_schema" />
</target>
```

- DatabaseIndexValidator: This is used to verify the defined indices.

```
<target name="validate-mithra-xml-test">
```

```
    <taskdef name="mithra-validate-
xml" classname="com.gs.fw.common.mithra.generator.objectxmlgenerator.Databas
        <classpath refid="mithraxmlgen.classpath"/>
    </taskdef>
    <mkdir dir ="${mithra.home}/build/mithra/tmp/generated"/>
    <mithra-validate-xml
        userName="userName"
        password="pass"
        databaseType="sybase"
        xml="${mithra.home}/xml/mithra/test/maxlengenerator/
TestClassList.xml"
        driver="com.sybase.jdbc4.jdbc.SybDriver"
        url="jdbc:sybase:Tds:hostname.example.com:5000/
mithra_qa/mithra_qa" schema="mithra_qa" />
</target>
```

- MaxLenValidator: This is used to verify the defined maxLength attribute of String columns.

```

<target name="validate-mithra-xml-test">
    <taskdef name="mithra-validate-
xml" classname="com.gs.fw.common.mithra.generator.objectxmlgenerator.MaxLenV
        <classpath refid="mithraxmlgen.classpath"/>
    </taskdef>
    <mkdir dir ="${mithra.home}/build/mithra/tmp/generated"/>
    <mithra-validate-
xml userName="username" password="password" databaseType="sybase"
            xml="${mithra.home}/xml/mithra/test/maxlengenerator/
TestClassList.xml"
            driver="com.sybase.jdbc4.jdbc.SybDriver"
            url="jdbc:sybase:Tds:host:port/schema"
            schema="some_schema" />
</target>
```

- NullableColumnValidator: This is used to verify if the columns are defined as nullable or non-nullable correctly.

```

<target name="validate-mithra-xml-test">
    <taskdef name="mithra-validate-
xml" classname="com.gs.fw.common.mithra.generator.objectxmlgenerator.Nullabl
        <classpath refid="mithraxmlgen.classpath"/>
    </taskdef>
    <mkdir dir ="${mithra.home}/build/mithra/tmp/generated"/>
    <mithra-validate-
xml userName="username" password="password" databaseType="sybase"
            xml="${mithra.home}/xml/mithra/test/maxlengenerator/
TestClassList.xml"
            driver="com.sybase.jdbc4.jdbc.SybDriver"
            url="jdbc:sybase:Tds:host:port/schema"
            schema="some_schema" />
</target>
```

Q: What's the best way to update a set of existing data based on a new incoming set of detached data?

A: First, you should avoid using List's contains/remove methods -- they require an equals contract, but it is usually not the best solution. Also, updating large (e.g. > 100) lists of data will be slow, as it's multiple O(n^2) loops.

The best practice is to use a FullUniqueIndex on your current/existing data to optimize access to help you create a delta. Something like the following:

```
FullUniqueIndex index = new FullUniqueIndex("",
 ProductFinder.getPrimaryKeyAttributes());
index.addAll(fileProducts);
for (int i = detachedDbProducts.size() - 1; i >= 0; i--)
{
    Product detachedDbProduct = detachedDbProducts.get(i);
    Product fileProduct = index.remove(detachedDbProduct);
    if (fileProduct == null)
    {
        // Removed item
        detachedDbProducts.remove(i);
    }
    else
    {
        // Updated item -- typically this would be an update,
 e.g.

 detachedDbProduct.copyNonPrimaryKeyValuesFrom(fileProduct);
    }
}
// New items (i.e. anything remaining in the index after the loop
 has finished)
detachedDbProducts.addAll(index.getAll());
...
detachedDbProducts.copyDetachedValuesToOriginalOrInsertIfNewOrDeleteIfRemoved(
```

For larger amounts of data, the multi-threaded loader (mtloader) / matcher thread is the way to go.

Q: Does Reladomo support result-set paging?

A: No. Not all database systems support pagination (notably Sybase and DB2).

The best we can do right now is limit the results fetched using `setMaxObjectsToRetrieve()`

Q: Does Reladomo support Views?

A: More or less. You are able to map a View to a Reladomo object, since from a SQL technical perspective a View is very similar to a Table, however it's behaviour is not guaranteed, especially with updates or when the underlying tables are mapped to other Reladomo objects.

In general, Views are an anti-pattern when using an ORM system, like Reladomo. They also tend to perform very poorly.

Q: Text fields / How can I store long text values using Reladomo?

A: In your DB schema, use "text" (or equivalent) as the column type, and in your Reladomo Object XML use javaType="String".

Q: Do I have any control for clearing objects that Reladomo has cached?

A: First, ask yourself why you need or _think_ you need to clear the cache -- quite often what you really need is your Reladomo-based app to work in a friendly way with other Reladomo-based apps. In this case, you should investigate Reladomo "Notification", which is a powerful and efficient way of managing cache coherency across multiple JVM instances.

The programmatic way to clear all cache entries in the JVM is:

```
MithraManager.getInstance().clearAllQueryCaches()
```

You can clear the cache on an individual finder with:

```
FooFinder.clearQueryCache()
```

You can also set timeouts for caches on a per-object basis in the runtime XML. If you choose a decent timeout (several minutes), then you get most of the benefit of caching without having to worry about clearing it. For example, a 2 minute timeout:

```
<MithraObjectConfiguration className="com.gs.fw.common.mithra.test.domain.Orde
    cacheType="partial"
    relationshipCacheTimeToLive="120000"
    cacheTimeToLive="120000"/>
```

Q: What's the difference between `delete()`, `terminate()`, and `purge()`?

A: All the methods are used for removing an object, but are used in different contexts and different actions:

- **`delete()`** is used when the object is not dated (i.e. does not have an AsOfAttribute), and removes the data entirely from the DB.
- **`terminate()`** is used when an object is dated, and will "chain out" a record, meaning that it will no longer be visible to queries as-of "now", but it’s history is still available and it can be queried "as-of".
- **`purge()`** is used when you are saying I don’t want that history any more (and for most of our systems, it means it must be archived off someplace else beforehand -- hence, caution is required when using this method).

The short of it is that delete() or terminate() is usually what you want, and if you feel the need to use purge() you should check with your team-lead and a Reladomo expert first to ensure it’s being used appropriately and correctly.