CORAL Database Copy Tools

By
Gitika Khare
RRCAT, Indore, India

Objective

- Several experiment applications are based on LCG database abstraction layer CORAL for accessing relational data from several supported back-ends (*Oracle, MySql, SQLite*).
- To simplify deployment of CORAL based applications, a set of associated tools are required to allow copying of individual tables or complete schemas between existing databases and technologies.

CoralTools

- The CoralTools package provides a set of export tools for the CORAL framework.
- The mapping between data types of source and destination schema follows the CORAL mapping rules.
- This would allow the CORAL based applications to run unchanged against data copies in a different database backend.
- The tools support schema and data copy between the following relational databases – Oracle, MySQL and SQLite.
- The CoralTools package is developed in Python and is an implementation of the PyCoral interface developed using the python/C API.

schemautils Python Package

The following set of CoralTools have been developed for the CORAL framework:

- listobjects(schema)
  Returns list of tables ordered by hierarchy
- dumpobjectlist(schema)
  Dumps the list of tables (and views) grouped and ordered by hierarchy, specifying the existing constraints and indexes.
- copyschema()
  Copies the schema objects from source to destination, without to copy data
- copydata()
  Copies the schema objects from source to destination, including data

schemautils Python Package

- copytableschema(tablename)
  Copies the specified tableschema. No data copy.

- copytabledata(tablename, selectionclause, selectionparameters)
  Copies the specified tableschema, including data

- copytablelistschema(tablelist)
  Copies the specified list of tables ordered by hierarchy. No data copy.

- copytablelistdata(tablelist)
  Copies the specified list of tables ordered by hierarchy, including data.
  tablelist is the list of tables with selectionclause and selectionparameters

Implementation Details

Resolution of Table Hierarchies

- Export of a tree of hierarchically related objects from source to specified database required the knowledge of table dependency trees.
- A Python class is developed for providing schema object relationship information.
- The relational dependency class provides an implementation for building table dependency trees for a specific table and an entire schema. It scans all the tables in the schema to construct the dependencies between the tables.
- Circular dependency between the tables is not handled since it requires special treatment.
- This class is used for the implementation of CoralTools.

Implementation Details

Scalability

- The tools are targeted to copy of big volumes of data.
- Implementation uses bulk operations on both reading and writing side, taking care to balance the data transfer from the source to the destination (minimize the roundtrips), and the memory required on the client side for caching.
- For copying of data buffer protocol is used, which allows reading and writing of data into a buffer without additional memory requirement.

Use Case 1

1. Export of all the objects from a specified schema from a source database into a specified database ordered by hierarchy.

   Source schema objects in destination database do not exist .

1.1 Schema only, no data
1.2 Schema + data

Use Case 2

2. Export of a tree of hierarchically related objects from a source database into a specified database. For the specified list of tables, the export considers ordering of tables by:

a) all the upper level tables which are referenced by foreign key constraints.
b) all of dependent tables (referencing columns of this table by foreign key constraint)
c) The export also considers data selection on the specified table columns.

2.1 Tables and related objects (constraints, indexes) do not exist in the destination database. The required tables and related objects are consistently created.

2.2 Tables and related objects exist , possibly with data . The existing set of tables involved is checked. The export is successful if no primary key violations are found during the new data insertion.

CoralTools Usage (initialization)

```
//Initialize Connection Service
svc = coral.ConnectionService()

//open session proxy for source schema providing logical service name & access mode
session1 = svc.connect( ‘write_test', accessMode = coral.access_ReadOnly )
transaction1 = session1.transaction()

//open session proxy for destination schema providing logical service name& accessmode
session2 = svc.connect( ‘coral_test', accessMode = coral.access_Update )
transaction2 = session2.transaction()

sourceSchema=session1.nominalSchema()
destSchema=session2.nominalSchema()
exp=exporter( sourceSchema,destSchema )
```

UseCase 1 (schema+data)

copydata()

```
transaction1.start(True)
transaction2.start()
try:
    exp.copydata()
    transaction2.commit()
    print "Data copied"
except Exception, e:
    transaction2.rollback()
    print "Test Failed"
    print str(e)
transaction1.commit()
```

UseCase 2 (schema+data)

copytabledata(tablename, selectionclause, selectionparameters)

```
transaction1.start(True)
transaction2.start()
try:
    tablename = "T1"
    selectionclause= "id  > :idmin and id < :idmax"
    selectionparameters = coral.AttributeList()
    selectionparameters.extend( "idmin","int")
    selectionparameters.extend( "idmax","int")
    selectionparameters["idmin"].setData(1)
    selectionparameters["idmax"].setData(10)
    exp.copytabledata(tablename,selectionclause,selectionparameters)
    transaction2.commit()
    print "Data copied"
except Exception, e:
    transaction2.rollback()
    print "Test Failed"
    print str(e)
transaction1.commit()
```

UseCase 2 (schema only)

copytablelistschema(tablelist)

```
transaction1.start(True)
transaction2.start()
try:
    tablelist = ['T3','T1','T2']
    exp.copytablelistschema(tablelist)
    transaction2.commit()
    print "Tables created"
except Exception, e:
    transaction2.rollback()
    print "Test Failed" print str(e)
transaction1.commit()
```

UseCase 2 (schema+data)

copytablelistdata(tablelist)

```
transaction1.start(True)
transaction2.start()
try:
    table1 = "T3"
    selectionclause1= "id >= 0 and id<10"

    table2 = "T2"
    selectionclause2= "id  >= :idmin and id < :idmax"
    selectionparameters2 = coral.AttributeList()
    selectionparameters2.extend( "idmin","int")
    selectionparameters2.extend( "idmax","int")
    selectionparameters2["idmin"].setData(0)
    selectionparameters2["idmax"].setData(25)

    table3 = "T1"
```

UseCase 2 (schema+data)

```
    tablelist = [[table1,selectionclause1],
                 [table2,selectionclause2,selectionparameters2],
                 [table3]]
    exp.copytablelistdata(tablelist)
    transaction2.commit()
    print "Data copied"
except Exception, e:
    transaction2.rollback()
    print "Test Failed"
    print str(e)
transaction1.commit()
```

Exception Handling

- Circular dependency exists between the tables.
- Tables exists in destination schema.
- Unique constraint violated - (if data exists in the table)
- table or view does not exist - (Upper level table referenced by foreign key constraint does not exist in destination schema)
- Integrity constraint for foreign key violated – (data not found in upper level table which are referenced by foreign key constraint)

Status

- The CoralTools have been developed and tested using the test cases with Oracle, MySQL and SQLite.
- The tools for both UseCase 1 & UseCase 2 have been tested with a case with POOL-ORA application with Oracle, SQLite and MySQL database.
- Documentation has been prepared in compliance with the CORAL doc framework.
- Testing for Scalability, Performance analysis and Optimization is in progress.
- The tools have to be tested with a case with COOL application.