SleepyCatIntegration

craig: The intended use in the JDO project is to store Java objects in the database. We have our own form of serialization that creates one byte[] for each persistent Java instance. Each such persistent instance is identified by a key which is either generated (a naive implementation uses a long) or user-defined, in which case we serialize the user's key.

mark: Ok, it sounds like you're not in need of help with bindings per se, and I'll just use a byte array binding as an example below.

craig: There are two primary access mechanisms: by primary key; or by table or index scan. The reason I asked about multiple bindings is that for some cases (return all instances of a particular class) we might just want you to do the work and return an iterator to the entrySet over the Map. Other cases, we want to scan a range of index values and I'm guessing that I can't just use your Map interface. If you actually do have range scans over secondary key values, this would be of great interest!

mark: I'm happy to say that we do have range scans over secondaries and I'll give you a small example. With the collections API, to access by secondary key you simply pass the secondary database to the stored map constructor instead of the primary database. This maps secondary key to the primary record. So the key binding in this case will be for the secondary key and the data binding will be for the primary record.

craig: Just to be specific, if I define my database to have a Map interface, I'd like to iterate over the {primary-key byte[], value byte[]}. In a separate connection, perhaps, I'd like to iterate over a {secondary-key value, primary-key byte[], value byte[]}. The value type isn't as important as the ability to get an iterator, for example by using a method on Map:

mark: This is not a problem. The only slightly tricky part, for access by secondary key, is that to return both the primary key and data we'll need an entity binding. Imagine a map of secondary key to primary record. Abstractly if we do this:

```
   value = map.get(secondaryKey);
```

then to get both the primary key and data we'll need for the value returned by Map.get to be a composite (entity) of the primary key and data. In my example I'll just use a two element array for the map entity value, where the first element is the byte[] of the primary key and the second is the byte[] of the primary data.

```
   byte[] secondaryKey = ...;
   byte[][] value = (byte[][]) map.get(secondaryKey);
   byte[] primaryKey = value[0];
   byte[] primaryData = value[1];
```

We need the casting because we don't currently support generics.

craig:

```
public class IndexEntrySet {
public Object secondaryKey;
public byte[ ] primaryKey;
public byte[ ] value;
}
public Iterator<IndexEntrySet> indexScan(Index secondaryIndex, Object lowValue, Object highValue);
```

mark: To iterate over a key range using the collections API, you use SortedMap.subMap, headMap or tailMap. These create very lightweight wrappers that enforce the key range. Our StoredSortedMap also has extension subMap, headMap and tailMap methods that support both inclusive and exclusive end points:

http://www.sleepycat.com/jedocs/java/com/sleepycat/collections/StoredSortedMap.html

Note that the range scan could also be done using the base API rather than the collections API. The base API provides a range lookup, but not a range iterator. Cursor.getSearchRangeKey positions the cursor to the first key that is greater or equal to the given key. You can iterate (getNext) from there, but you have to manually enforce the end point of the range. The collections API enforces the end point via the submap.

Please let me know whether this example is clear and gives you what you need to know.

```
import java.io.File;
import java.io.IOException;
import java.util.Collection;
import java.util.Iterator;
import java.util.Map.Entry;
import java.util.SortedMap;

import com.sleepycat.bind.ByteArrayBinding;
import com.sleepycat.bind.EntryBinding;
import com.sleepycat.bind.EntityBinding;
import com.sleepycat.collections.StoredIterator;
```

```
import com.sleepycat.collections.StoredSortedMap;
import com.sleepycat.je.Database;
import com.sleepycat.je.DatabaseConfig;
import com.sleepycat.je.DatabaseEntry;
import com.sleepycat.je.DatabaseException;
import com.sleepycat.je.Environment;
import com.sleepycat.je.EnvironmentConfig;
import com.sleepycat.je.SecondaryConfig;
import com.sleepycat.je.SecondaryDatabase;
import com.sleepycat.je.SecondaryKeyCreator;

/**
 * An example of using submaps with secondary keys.
 */
public class IndexRange {

    private static final byte[] ONE   = {1};
    private static final byte[] TWO   = {2, 2};
    private static final byte[] THREE = {3, 3, 3};
    private static final byte[] FOUR  = {4, 4, 4, 4};
    private static final byte[] FIVE  = {5, 5, 5, 5, 5};

    /**
     * Usage: IndexRange -h ENVHOME
     */
    public static void main(String[] args) {
        try {
            if (args.length != 2 || !"-h".equals(args[0])) {
                System.out.println("Usage: IndexRange -h ENVHOME");
                System.exit(2);
            } else {
                String envHome = args[1];
                IndexRange app = new IndexRange(envHome);
                app.exec();
                app.close();
                System.exit(0);
            }
        } catch (Exception e) {
            e.printStackTrace();
            System.exit(1);
        }
    }

    private Environment env;
    private Database priDb;
    private SecondaryDatabase secDb;
    private SortedMap priMap;
    private SortedMap secMap;

    /**
     * Opens the environment and databases.
     */
    private IndexRange(String envHome)
        throws IOException, DatabaseException {

        EnvironmentConfig envConfig = new EnvironmentConfig();
        envConfig.setAllowCreate(true);
        env = new Environment(new File(envHome), envConfig);

        DatabaseConfig priConfig = new DatabaseConfig();
        priConfig.setAllowCreate(true);
        priDb = env.openDatabase(null, "pri", priConfig);

        SecondaryConfig secConfig = new SecondaryConfig();
        secConfig.setAllowCreate(true);
        secConfig.setSortedDuplicates(true);
        secConfig.setKeyCreator(new KeyCreator());
        secDb = env.openSecondaryDatabase(null, "sec", priDb, secConfig);

        EntryBinding entryBinding = new ByteArrayBinding();
        EntityBinding entityBinding = new ByteArrayEntityBinding();
```

```
    priMap = new StoredSortedMap(priDb, entryBinding, entryBinding, true);
    secMap = new StoredSortedMap(secDb, entryBinding, entityBinding, true);
}

/**
 * Closes the environment and databases.
 */
private void close()
    throws DatabaseException {

    secDb.close();
    priDb.close();
    env.close();
}

/**
 * Derives the secondary key from the entire primary data value.
 */
private static class KeyCreator implements SecondaryKeyCreator {

    public boolean createSecondaryKey(SecondaryDatabase db,
                                      DatabaseEntry primaryKey,
                                      DatabaseEntry primaryData,
                                      DatabaseEntry secondaryKey)
        throws DatabaseException {

        secondaryKey.setData(primaryData.getData());
        return true;
    }
}

/**
 * Binds a byte[][] entity to a byte[] key and byte[] data.
 */
private static class ByteArrayEntityBinding implements EntityBinding {

    public Object entryToObject(DatabaseEntry key, DatabaseEntry data) {
        return new byte[][] {key.getData(), data.getData()};
    }

    public void objectToData(Object object, DatabaseEntry data) {
        byte[][] val = (byte[][]) object;
        data.setData(val[1]);
    }

    public void objectToKey(Object object, DatabaseEntry key) {
        byte[][] val = (byte[][]) object;
        key.setData(val[0]);
    }
}

/**
 * Tries a few things.
 */
private void exec() {

    priMap.put(ONE, FIVE);
    priMap.put(TWO, FOUR);
    priMap.put(THREE, THREE);
    priMap.put(FOUR, TWO);
    priMap.put(FIVE, ONE);

    System.out.println("Primary Database");
    print(priMap.entrySet(), false);

    System.out.println("\nSecondary Database");
    print(secMap.entrySet(), true);

    System.out.println("\nPrimary Database Submap");
    print(priMap.subMap(TWO, FOUR).entrySet(), false);
```

```
        System.out.println("\nSecondary Database Submap");
        print(secMap.subMap(TWO, FOUR).entrySet(), true);
    }

    /**
     * Prints the collection of Map.Entry objects.
     */
    private void print(Collection mapEntries, boolean entityValues) {

        Iterator i = mapEntries.iterator();
        try {
            while (i.hasNext()) {
                Entry entry = (Entry) i.next();
                byte[] key;
                byte[] data;
                if (entityValues) {
                    byte[][] entity = (byte[][]) entry.getValue();
                    key = entity[0];
                    data = entity[1];
                } else {
                    key = (byte[]) entry.getKey();
                    data = (byte[]) entry.getValue();
                }
                System.out.println("  " + format(key) + ": " + format(data));
            }
        } finally {
            StoredIterator.close(i);
        }
    }

    /**
     * Formats a byte array as a string.
     */
    private String format(byte[] bytes) {
        StringBuffer buf = new StringBuffer();
        for (int i = 0; i < bytes.length; i++) {
            buf.append(bytes[i]);
            buf.append(' ');
        }
        return buf.toString();
    }
}
```