Chapter 10. Secondary Databases

Table of Contents

Usually you find database records by means of the record's key. However, the key that you use for your record will not always contain the information required to provide you with rapid access to the data that you want to retrieve. For example, suppose your `Database` contains records related to users. The key might be a string that is some unique identifier for the person, such as a user ID. Each record's data, however, would likely contain a complex object containing details about people such as names, addresses, phone numbers, and so forth. While your application may frequently want to query a person by user ID (that is, by the information stored in the key), it may also on occasion want to locate people by, say, their name.

Rather than iterate through all of the records in your database, examining each in turn for a given person's name, you create indexes based on names and then just search that index for the name that you want. You can do this using secondary databases. In JE, the `Database` that contains your data is called a *primary database*. A database that provides an alternative set of keys to access that data is called a *secondary database*, and these are managed using `SecondaryDatabase` class objects. In a secondary database, the keys are your alternative (or secondary) index, and the data corresponds to a primary record's key.

You create a secondary database by using a `SecondaryConfig` class object to identify an implementation of a `SecondaryKeyCreator` class object that is used to create keys based on data found in the primary database. You then pass this `SecondaryConfig` object to the `SecondaryDatabase` constructor.

Once opened, JE manages secondary databases for you. Adding or deleting records in your primary database causes JE to update the secondary as necessary. Further, changing a record's data in the primary database may cause JE to modify a record in the secondary, depending on whether the change forces a modification of a key in the secondary database.

Note that you can not write directly to a secondary database. While methods exist on `SecondaryDatabase` and `SecondaryCursor` that appear to allow this, they in fact always throw `UnsupportedOperationException`. To change the data referenced by a `SecondaryDatabase` record, modify the primary database instead. The exception to this rule is that delete operations are allowed on the `SecondaryDatabase` object. See Deleting Secondary Database Records for more information.

Note

Secondary database records are updated/created by JE only if the `SecondaryKeyCreator.createSecondaryKey()` method returns `true`. If `false` is returned, then JE will not add the key to the secondary database, and in the event of a record update it will remove any existing key.

See Implementing Key Creators for more information on this interface and method.

When you read a record from a secondary database, JE automatically returns the key and data from the corresponding record in the primary database.

Opening and Closing Secondary Databases

You manage secondary database opens and closes using the `Environment.openSecondaryDatabase()` method. Just as is the case with primary databases, you must provide `Environment.openSecondaryDatabase()` with the database's name and, optionally, other properties such as whether duplicate records are allowed, or whether the secondary database can be created on open. In addition, you must also provide:

- A handle to the primary database that this secondary database is indexing. Note that this means that secondary databases are maintained only for the specified `Database` handle. If you open the same `Database` multiple times for write (such as might occur when opening a database for read-only and read-write in the same application), then you should open the `SecondaryDatabase` for each such `Database` handle.
- A `SecondaryConfig` object that provides properties specific to a secondary database. The most important of these is used to identify the key creator for the database. The key creator is responsible for generating keys for the secondary database. See Secondary Database Properties for details.

Note

Primary databases *must not* support duplicate records. Secondary records point to primary records using the primary key, so that key must be unique.

So to open (create) a secondary database, you:

1. Open your primary database.
2. Instantiate your key creator.
3. Instantiate your `SecondaryConfig` object.
4. Set your key creator object on your `SecondaryConfig` object.
5. Open your secondary database, specifying your primary database and your `SecondaryConfig` at that time.

For example:

```
package je.gettingStarted;

import com.sleepycat.bind.tuple.TupleBinding;

import com.sleepycat.je.Database;
import com.sleepycat.je.DatabaseConfig;
import com.sleepycat.je.DatabaseException;
import com.sleepycat.je.Environment;
import com.sleepycat.je.SecondaryDatabase;
import com.sleepycat.je.SecondaryConfig;

import java.io.File;

...

DatabaseConfig myDbConfig = new DatabaseConfig();
SecondaryConfig mySecConfig = new SecondaryConfig();
myDbConfig.setAllowCreate(true);
mySecConfig.setAllowCreate(true);
// Duplicates are frequently required for secondary databases.
mySecConfig.setSortedDuplicates(true);

// Open the primary
Environment myEnv = null;
Database myDb = null;
SecondaryDatabase mySecDb = null;
try {
    String dbName = "myPrimaryDatabase";

    myEnv = new Environment(new File("/tmp/JEENV"), null);
    myDb = myEnv.openDatabase(null, dbName, myDbConfig);
```

```
        // A fake tuple binding that is not actually implemented anywhere
        // in this manual. The tuple binding is dependent on the data in use.
        // Tuple bindings are described earlier in this manual.
        TupleBinding myTupleBinding = new MyTupleBinding();

        // Open the secondary.
        // Key creators are described in the next section.
        FullNameKeyCreator keyCreator =
            new FullNameKeyCreator(myTupleBinding);

        // Get a secondary object and set the key creator on it.
        mySecConfig.setKeyCreator(keyCreator);

        // Perform the actual open
        String secDbName = "mySecondaryDatabase";
        mySecDb = myEnv.openSecondaryDatabase(null, secDbName, myDb,
                                              mySecConfig);
    } catch (DatabaseException de) {
        // Exception handling goes here ...
    }
```

To close a secondary database, call its close() method. Note that for best results, you should close all the secondary databases associated with a primary database before closing the primary.

For example:

```
try {
    if (mySecDb != null) {
        mySecDb.close();
    }

    if (myDb != null) {
        myDb.close();
    }

    if (myEnv != null) {
        myEnv.close();
    }
} catch (DatabaseException dbe) {
    // Exception handling goes here
}
```