Reading and Writing Database Records

When reading and writing database records, be aware that there are some slight differences in behavior depending on whether your database supports duplicate records. Two or more database records are considered to be duplicates of one another if they share the same key. The collection of records sharing the same key are called a *duplicates set.*

By default, JE databases do not support duplicate records. Where duplicate records are supported, cursors (see below) are used to access all of the records in the duplicates set.

JE provides two basic mechanisms for the storage and retrieval of database key/data pairs:

- The `Database.put()` and `Database.get()` methods provide the easiest access for all non-duplicate records in the database. These methods are described in this section.
- Cursors provide several methods for putting and getting database records. Cursors and their database access methods are described in Using Cursors.

Writing Records to the Database

Database records are stored in the internal BTree based on whatever sorting routine is available to the database. Records are sorted first by their key. If the database supports duplicate records, then the records for a specific key are sorted by their data.

By default, JE sorts both keys and the data portion of duplicate records using unsigned byte-by-byte lexicographic comparisons. This default comparison works well for the majority of cases. However, in some case performance benefits can be realized by overriding the default comparison routine. See Using Comparators for more information.

You can use the following methods to put database records:

- `Database.put()`

  Puts a database record into the database. If your database does not support duplicate records, and if the provided key already exists in the database, then the currently existing record is replaced with the new data.

  Be aware that version of this method exists which accepts a `Put` enum. If `Put.OVERWRITE` is provided, then existing database records are overwritten. If `Put.NO_OVERWRITE` is provided, then existing records will not be overwritten.

- `Database.putNoOverwrite()`

  Disallows overwriting (replacing) an existing record in the database. If the provided key already exists in the database, then this method returns `OperationStatus.KEYEXIST` even if the database supports duplicates.

- `Database.putNoDupData()`

  Puts a database record into the database. If the provided key and data already exists in the database (that is, if you are attempting to put a record that compares equally to an existing record), then this returns `OperationStatus.KEYEXIST`.

When you put database records, you provide both the key and the data as `DatabaseEntry` objects. This means you must convert your key and data into a Java `byte` array. For example:

```
package je.gettingStarted;

import com.sleepycat.je.Database;
import com.sleepycat.je.DatabaseEntry;
```

```
...

// Environment and database opens omitted for clarity.
// Environment and database must NOT be opened read-only.

String aKey = "myFirstKey";
String aData = "myFirstData";

try {
    DatabaseEntry theKey = new DatabaseEntry(aKey.getBytes("UTF-8"));
    DatabaseEntry theData = new DatabaseEntry(aData.getBytes("UTF-8"));
    myDatabase.put(null, theKey, theData);
} catch (Exception e) {
    // Exception handling goes here
}
```

Getting Records from the Database

The `Database` class provides several methods that you can use to retrieve database records. Note that if your database supports duplicate records, then these methods will only ever return the first record in a duplicate set. For this reason, if your database supports duplicates, you should use a cursor to retrieve records from it. Cursors are described in Using Cursors.

You can use either of the following methods to retrieve records from the database:

- `Database.get()`

  Retrieves the record whose key matches the key provided to the method. If no records exists that uses the provided key, then `OperationStatus.NOTFOUND` is returned.

- `Database.getSearchBoth()`

  Retrieve the record whose key matches both the key and the data provided to the method. If no record exists that uses the provided key and data, then `OperationStatus.NOTFOUND` is returned.

Both the key and data for a database record are returned as byte arrays in `DatabaseEntry` objects. These objects are passed as parameter values to the `Database.get()` method.

In order to retrieve your data once `Database.get()` has completed, you must retrieve the `byte` array stored in the `DatabaseEntry` and then convert that `byte` array back to the appropriate datatype. For example:

```
package je.gettingStarted;

import com.sleepycat.je.Database;
import com.sleepycat.je.DatabaseEntry;
import com.sleepycat.je.LockMode;
import com.sleepycat.je.OperationStatus;

...

// Environment and database opens omitted for clarity.
// Environment and database may be opened read-only.

String aKey = "myFirstKey";

try {
    // Create a pair of DatabaseEntry objects. theKey
    // is used to perform the search. theData is used
    // to store the data returned by the get() operation.
    DatabaseEntry theKey = new DatabaseEntry(aKey.getBytes("UTF-8"));
    DatabaseEntry theData = new DatabaseEntry();

    // Perform the get.
    if (myDatabase.get(null, theKey, theData, LockMode.DEFAULT) ==
        OperationStatus.SUCCESS) {

        // Recreate the data String.
        byte[] retData = theData.getData();
        String foundData = new String(retData, "UTF-8");
        System.out.println("For key: '" + aKey + "' found data: '" +
                            foundData + "'.");
    } else {
        System.out.println("No record found for key '" + aKey + "'.");
    }
} catch (Exception e) {
    // Exception handling goes here
}
```

Deleting Records

You can use the `Database.delete()` method to delete a record from the database. If your database supports duplicate records, then all records associated with the provided key are deleted. To delete just one record from a list of duplicates, use a cursor. Cursors are described in Using Cursors.

You can also delete every record in the database by using `Environment.truncateDatabase()`.

For example:

```
package je.gettingStarted;

import com.sleepycat.je.Database;
import com.sleepycat.je.DatabaseEntry;

...

// Environment and database opens omitted for clarity.
// Environment and database can NOT be opened read-only.

try {
    String aKey = "myFirstKey";
    DatabaseEntry theKey = new DatabaseEntry(aKey.getBytes("UTF-8"));

    // Perform the deletion. All records that use this key are
    // deleted.
    myDatabase.delete(null, theKey);
} catch (Exception e) {
    // Exception handling goes here
}
```

Data Persistence

When you perform a database modification, your modification is made in the in-memory cache. This means that your data modifications are not necessarily flushed to disk, and so your data may not appear in the database after an application restart.

Therefore, if you care if your data is durable across system failures, and to guard against the rare possibility of database corruption, you should use transactions to protect your database modifications. Every time you commit a transaction, JE ensures that the data will not be lost due to application or system failure. Transaction usage is described in the *Berkeley DB, Java Edition Getting Started with Transaction Processing* guide.

If you do not want to use transactions, then the assumption is that your data is of a nature that it need not exist the next time your application starts. You may want this if, for example, you are using JE to cache data relevant only to the current application runtime.

If, however, you are not using transactions for some reason and you still want some guarantee that your database modifications are persistent, then you should periodically run environment syncs. Syncs cause any dirty entries in the in-memory cache and the operating system's file cache to be written to disk. As such, they are quite expensive and you should use them sparingly.

Note that by default, a sync is run every time you close an environment. You can also run a sync by calling the `Environment.sync()` method.

For a brief description of how JE manages its data in the cache and in the log files, and how sync works, see Databases and Log Files.

Prev Up Next

Chapter 8. Database Records Home Using Time to Live