BlackBerry: Pushing Changes to the Server

This is the last of the BlackBerry chapters, and it’s where we pull the functionality of the application together. As of this point, we have a synchronization class that will pull the latest version of the bookmarks from the server and an ORM layer that will put them in the database. We also have forms for logging on and showing the bookmarks implemented. What we’re going to do in this chapter is create two new forms—one for asking the user which bookmark needs to be edited and one that actually edits the bookmark. We’ll then complete the synchronization routine by making it send inserts, updates, or deletes back to the server. This is going to include modifying the SqlFilter functionality to allow it to constrain data and doing more work with classes that handle HTTP communications.

Note In this chapter, although we have an entity type that relates only to bookmarks, the code has been designed to be generic enough to be easily adapted to work with multiple types of entities.

Capturing Local Changes

We’ll start by creating the UI to capture the changes, and in the second part of this chapter, we will look at how those changes can be streamed up to the server. Firstly, we’ll look at modifying our SqlFilter class to make it more sophisticated.

Constraining SQL Filters

When we built SqlFilter in the last chapter, we used it only to return all items. We’re going to need more sophistication in this chapter, and so we’ll add the ability to add constraints to the query.

In BootFX, the open source ORM tool that the database work in this book is based on, which we discussed in Chapter 3, there are two kinds of constraints, field constraints and free constraints. A field constraint is bound to a single field and is used to say things like “I want this field to be equal to this value.” A free constraint allows you to patch in a snippet of SQL if you need more sophistication. The advantage of field constraints is that they’re really easy to code up, but they are limited. The disadvantage of free constraints—when working with traditional database servers that run on server hardware—is that they are SQL dialect–specific, i.e., if you write one that works on SQL Server, you’re not guaranteed it will run on MySQL, Oracle, etc. Here, just to illustrate the point and for consistency with BootFX, we’re going to build a base class called SqlConstraint and a field constraint implementation in SqlFieldConstraint, but we will not build SqlFreeConstraint.
When we come to build our query (recall we do this in the `GetStatement` method on `SqlFilter`), we'll walk the list of constraints that we have and call a method on the constraint called `Append`, which will be responsible for actually modifying the query. `SqlConstraint` will be an abstract class with an abstract `Append` method, like so:

```java
package com.amxMobile.SixBookmarks.Runtime;
package com.amxmobile.SixBookmarks.Database;

public abstract class SqlConstraint
{
    public SqlConstraint()
    {
    }

    public abstract void Append(SqlStatement sql, StringBuffer builder) throws Exception;
}
```

The field constraints will need to know three things: the field that they're binding to, the value to match, and the operator to use for the matching. In this book, I've created just two operators—equal to or not equal to. In a real-world implementation, you'd likely need more operators. We defined these constants when we built the `SqlFilter` class originally.

When we're building our SQL statement, the constraint just has to add the small portion of the `WHERE` clause that contains the expression to match. Here's a sample statement with the constraint's area of responsibility underlined:

```
SELECT * FROM BOOKMARKS WHERE LocalDeleted=0
```

In other words, it's just a small part of the statement that the constraint cares about.

Here's the implementation of `SqlFieldConstraint` that holds the field, the operator, and the value:

```java
package com.amxmobile.SixBookmarks.Database;
import javax.microedition.global.*;
import com.amxmobile.SixBookmarks.Entities.*;

public class SqlFieldConstraint extends SqlConstraint
{
    private EntityField _field;
    private Object _value;
    private int _operator;

    public SqlFieldConstraint(EntityField field, Object value)
    {
        this(field, SqlFilter.OPERATOR_EQUALTO, value);
    }

    public SqlFieldConstraint(EntityField field, int op, Object value)
    {
        this(field, op, value);
    }
```