6: Building Applications

Databases and applications

Up to now you have been working directly with the **tables** and **queries** in a database. You have learned how to design the database schema and how to design queries to retrieve information from the tables.

However, it's unlikely that you would want the end-users of your database to work with those tables and queries. The users of the GCUTours system, for example, will probably not be database experts who are happy to delve into the tables or to run SQL queries – they just want to book holidays!

To be useful, a database usually needs to be part of an **application**. There are two main reasons for creating an application:

- to provide a **user-friendly interface** for getting information into and out of the database.
- to provide processing of information, or **business logic**, over and above the capability of SQL queries

An example of a piece of business logic in the GCUTours system might be:

> *when a new booking is created, the system should update the value of the sales value for the appropriate package*

People have some different viewpoints on databases and applications:

Building applications

There are many different ways of building an application which uses a database. Here are some examples:

RDBMS application	• uses tools provided by the RDBMS • form and report designers create the interface • forms and reports can be created as web pages or as desktop applications • a programming language built into the RDBMS can be used to write business logic, e.g. Oracle PL/SQL
Client-server application	• desktop client application written in a programming language such as C# or Java • connects to a database server to store and retrieve data • database server is usually on a separate computer which can be accessed over a network by many clients
Web application	• HTML web pages with code included which queries the database and allows results to be embedded in page • needs a server which supports a web application technology such as ASP.NET or Java Server Pages • code written in a programming language such as C# or Java • code to query database runs on a server and HTML page with data embedded is sent to web browser • suitable for very simple applications
Enterprise web application	• uses same technology as web application but designed for larger, more complex applications • application is organised in layers/tiers, each with their own role, e.g. database access • information is represented as classes and objects within the application, and passed to a database for long-term storage

There are many other types of applications which can also use databases, for example games and mobile phone applications.

The types of application listed above look very different, but they actually have a lot in common:

- forms allow users to enter information
- reports or summaries of information are presented to users
- user actions result in business logic being carried out

Regardless of the tools used, **applications must be carefully designed to allow the user to carry out the tasks** associated with the use cases of the system. Users focus

on tasks, and should not be required to understand the structure of the underlying database.

The interface should:

- present the user with information in a way which **matches the task** rather than the database schema – join queries are often needed to gather data from separate tables, and queries often use values input by the user as criteria to search for specific data

- gather information from the user which needs to be stored – this usually involves constructing INSERT, UPDATE or DELETE queries using values input by the user

In all but the simplest cases, an application is not simply a way of browsing and editing database tables. Real applications need to have interface and business logic, which is implemented by writing code in a programming language.

Database servers

In many applications, the database is a resource which needs to be shared by many **clients** at the same time. A client could be, for example. a desktop application or a web page. Databases which need to be accessed **concurrently** are typically located on **database servers**. A database server can be a piece of software running on the same computer as the client, or on a separate computer which clients access over a network. Sometimes the computer which hosts the database is also known as a database server.

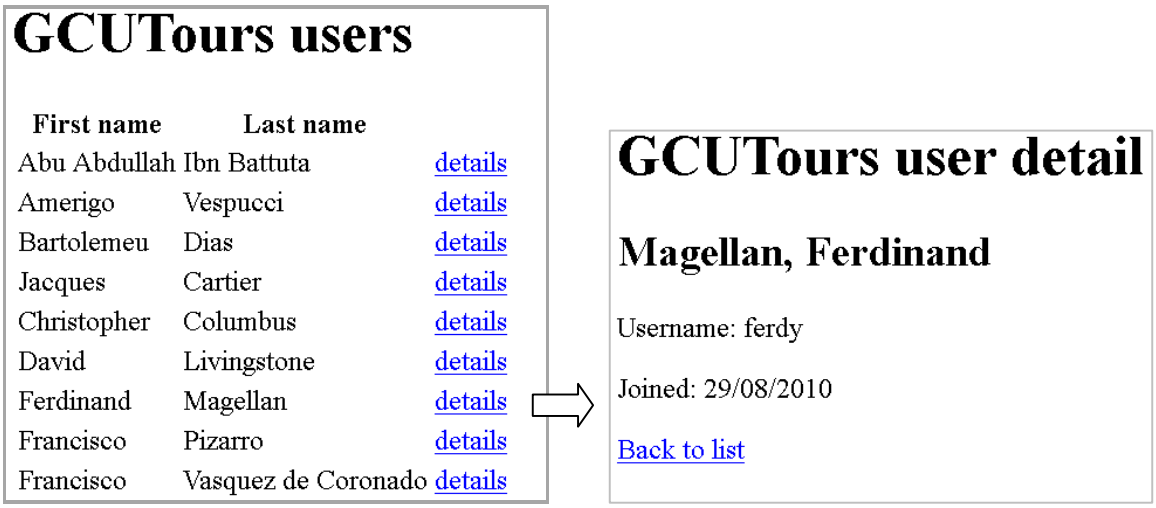
Many popular RDBMSs are designed to work in this client-server mode, for example Oracle, MySQL and Microsoft SQL Server. In contrast, databases such as SQL Server Compact, which you have been using, and Microsoft Access are designed mainly to be embedded within a single application, and are not suitable for large-scale applications. However, the principles of database design and the SQL language are the same. It can be convenient to design and test an application using an SQL Server Compact database and then migrate this to SQL Server when deploying the application.

Web applications with WebMatrix

WebMatrix is designed for building simple data-driven web applications. It is based on a server technology called ASP.NET Web Pages, and uses the C# programming language. We will look at how it can be used to create some simple web pages which might be part of a GCUTours web application. We will not worry too much about designing nice-looking web pages here - you will do that in another module. You can download these examples from your module website and try out all these examples. The full code listing are given at the end of these notes.

Listing users

The first page we will look at gets a list of the names of the users from the database and displays it. For each user, it shows a hyperlink which links to a page which displays further details about that user. This is an example of **master-detail pages**, which are very common in websites.



The pages are created in WebMatrix as CSHTML files (HTML files with C# code in them). They contain mainly HTML to define the layout of the pages, but also include some special code which queries the database and embeds data in the HTML so that it appears in the page in your web browser. This code is marked with @ symbols, and sections of code can be enclosed in {} brackets. Don't worry if you are not very familiar with HTML – you will learn more about it in another module. HTML mainly consists of elements which mark up elements of a web page, e.g. paragraphs (<p>), tables, rows and cells (<table>, <tr> ,<td>) and hyperlinklinks (<a>).

The user page, *users.cshtml,* includes the following code which connects to and queries the database

```
@{
    var db = Database.Open("gcutourswm");
    var selectQueryString = "SELECT * FROM Users";
 }
```

The results of the query are embedded in an HTML table:

```
@foreach (var row in db.Query(selectQueryString)){
       <tr>
              <td>@row.firstname</td>
              <td>@row.lastname</td>
              <td><a href="userdetail.cshtml?username
                     @row.username">details</a></td>
       </tr>
}
```

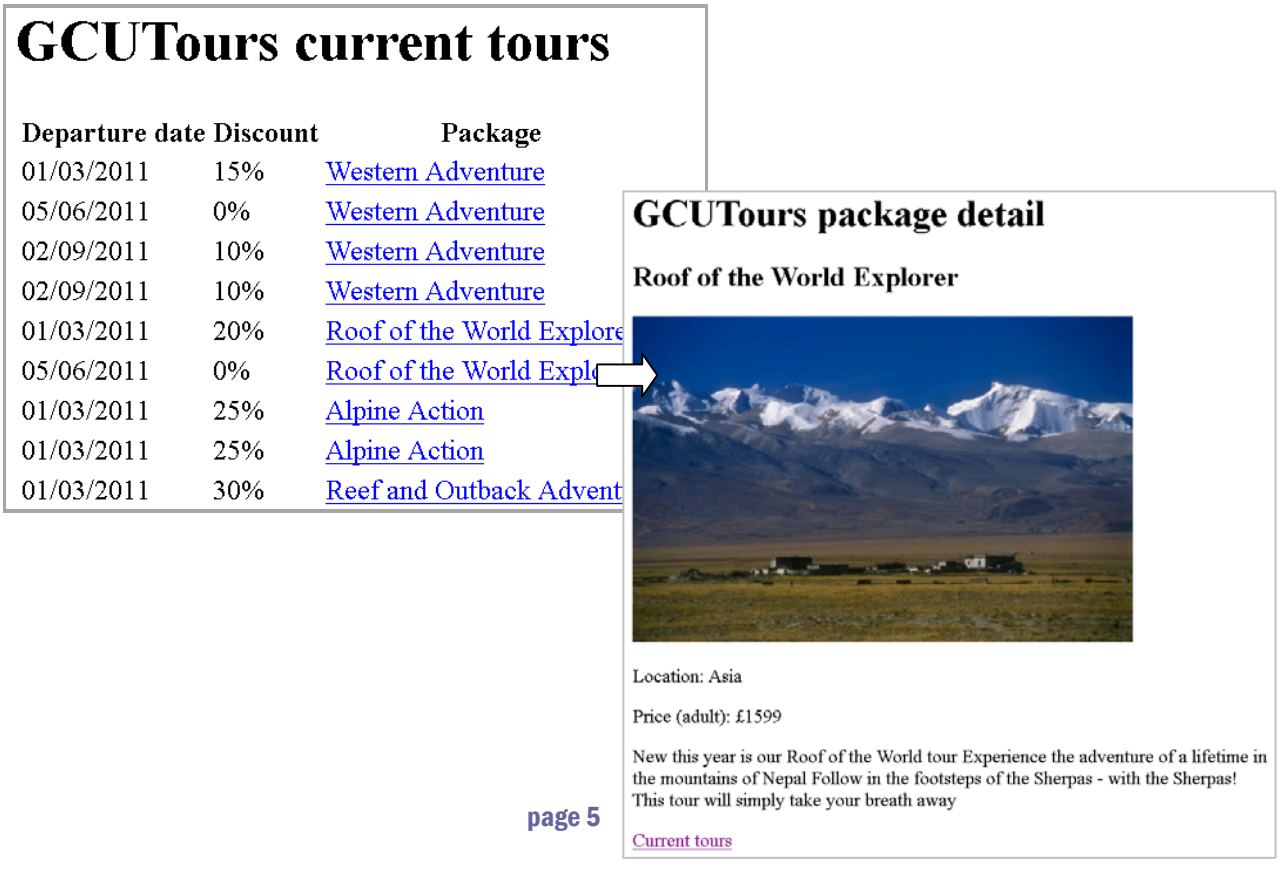
This code uses a **foreach loop** which goes through the results of the query and produces a table row for each user. Note that the third cell in each row displays a hyperlink which includes the username for that user. When a user clicks on the details link for Ferdinand Magellan, for example, the browser will navigate to **userdetail.cshtml?username=ferdy**.

The user detail page, *userdetail.cshtml*, will need to be able to read the username value from this **query string**. It includes the following code to get the username value and uses it as a **parameter** in a query: When the query is executed by *db.Query*, the parameter *@0* is replaced by the value of *username* which has been read from the query string as *Request["username"].*

```
@{
    var username = Request["username"];
    var db = Database.Open("gcutourswm");
    var selectQueryString =
        "SELECT * FROM Users WHERE username=@0";
    var result = db.Query(selectQueryString, username).FirstOrDefault();
 }
```

Tours and packages

The user list example all the data was from the same table. The next example shows master-detail pages where the master page contains data from one table, and the detail page shows data in a related table. First, a list of current tours is shown, and each one contains a link to a page showing the details of the related package.



The code for the query in *tours.cshtml* is similar to the users example. Note that the query needs data from both *Tours* and *Packages* tables as we want to show the package name, which is not in the *Tours* table.

```
@{
    var db = Database.Open("gcutourswm");
    var selectQueryString =
        @"SELECT departuredate, offer, packagename, Tours.packageID
        FROM Tours, Packages
        WHERE Tours.packageID = Packages.packageID";
 }
```

Some slightly complicated-looking C# code has been used to format the date neatly:

```
<td>@string.Format("{0:d}", row.departuredate)</td>
```

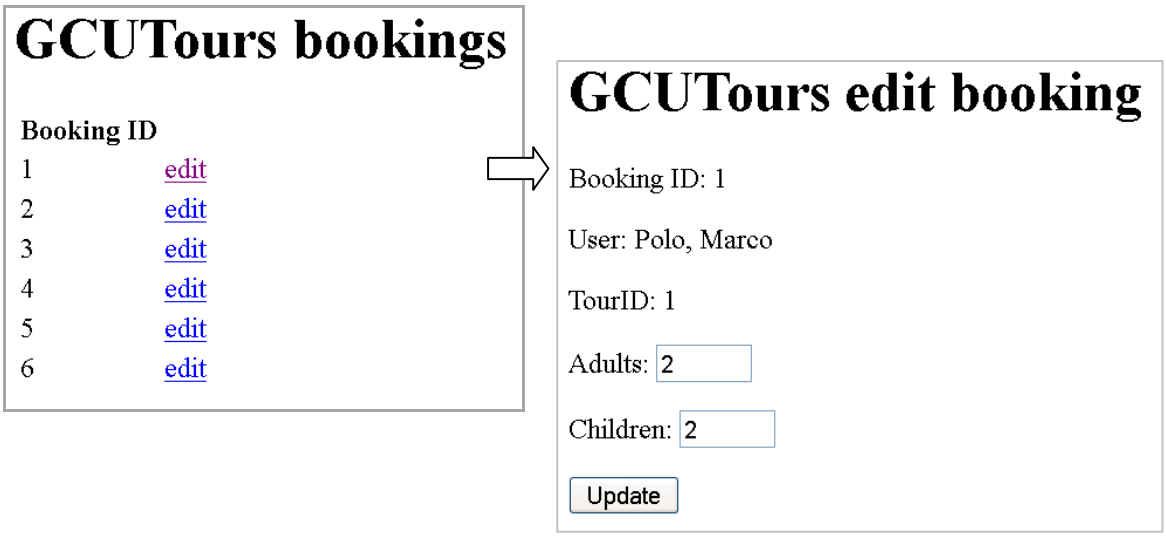
The page *packages.cshtml* is similar to the user details page. One extra feature is the way it displays an image using an <img> HTML element. The code embeds the path to an image file, which is in the *images* folder of the website:

```
<img src="@result.tourpicurl" />
```

The **path** to the image file is stored in a field in the database. Note that the **image itself is not stored**. This is a common way of displaying images in data-driven websites.

Editing bookings

The final example demonstrates a form which allows the user to update data. A page shows a list of bookings with a link beside each one which leads to a page which allows the number of persons in the booking to be updated.



The edit form page, *editbooking.cshtml,* is a bit more complicated than the other ones we've seen, so don't worry if you don't follow what the code is doing. The HTML contains a <form> element.

```
<form method="post" action="">
            <input type="hidden" name="bbookingID" value="@id" />
            <p>Booking ID: @result.bookingID</p>
            <p>User: @result.lastname, @result.firstname</p>
            <p>TourID: @result.tourID</p>
            <p>Adults: <input name="adults" type="text" size="5"
                    value="@result.adults" /></p>
            <p>Children: <input name="children" type="text" size="5"
                    value="@result.children" /></p>
            <p><input type="submit" value="Update" /></p>
</form>
```

The form contains <input> elements which display the text boxes for editing adults and children. There is also a hidden input element which is simply there to keep track of the current booking ID.

When the user clicks the Update button the values entered in the adults and children boxes are sent, or **posted**, back to the same page, and can be read, along with the booking id which is posted by the hidden input.

```
var id = Request["bookingID"];
var adults = Request["adults"];
var children = Request["children"];
```

The code checks to see whether data has been posted from a form (*IsPost*) and if so, an update query is executed. The result of this should be to update the values of *adults* and *children* for that booking in the *Bookings* table. After updating, the browser is redirected back to the list of bookings.

```
if (IsPost) {
       var updateQuery = @"UPDATE Bookings
             SET adults = @0, children = @1
             WHERE bookingID = @2";
       db.Execute(updateQuery, adults, children, id);

       Response.Redirect(@Href("~/bookings.cshtml"));
}
```

You can test this page by using it to edit a booking. When you have done so, go to the Databases workspace and look at the data in the *Bookings* table to see if the update has worked. You may have to refresh the table.

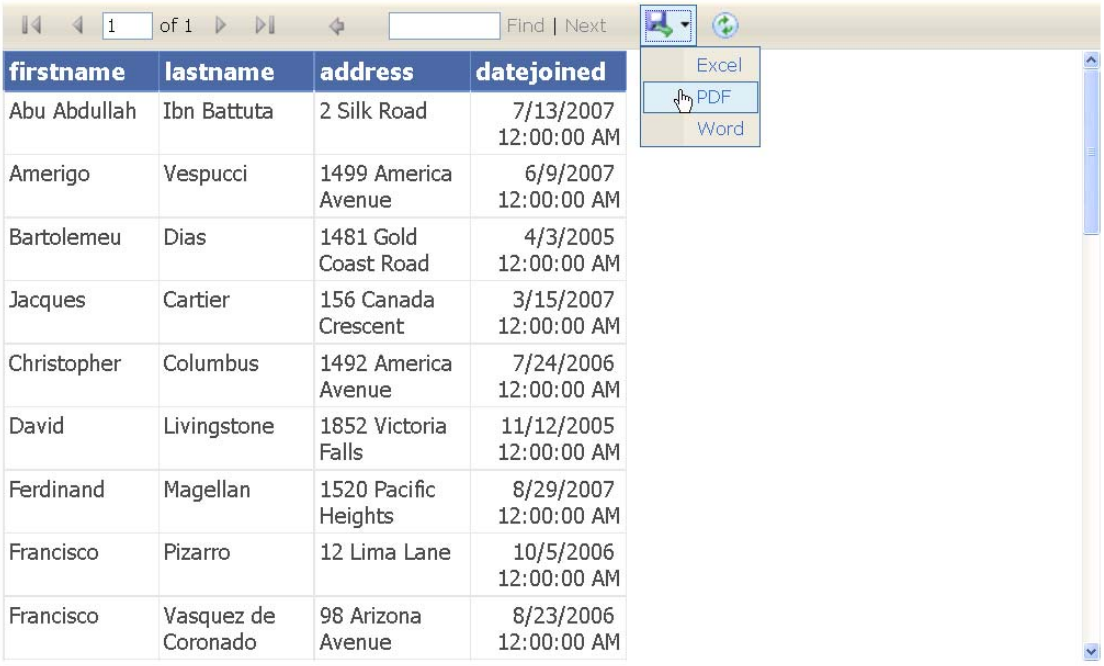
Reports

A report should give an **easily readable summary** of some part of the data in a database. Reporting tools are used to produce attractively formatted reports which might, for example, be used to present information to company managers. Reports can be produced in a form for printing or can be displayed in web pages.

The data in a report can be based on **tables** or **saved queries**, and can be **formatted** in a variety of ways. Data can be **grouped** and **summarised** to show meaningful information which could be difficult to discern simply by looking at the raw data in the tables.

Some RDBMSs, such as Oracle and Access, include reporting tools, while some programming tools, such as Microsoft Visual Studio, can generate reports. There are also specialised reporting tools such as Crystal Reports. The figure below shows a web report created in Visual Studio. The report has options to save to Excel, PDF or Word for printing.

WebMatrix doesn't have any support built in for database reports, although, confusingly, it does have a workspace for reports on web page usage.



firstname	lastname	address	datejoined
Abu Abdullah	Ibn Battuta	2 Silk Road	7/13/2007 12:00:00 AM
Amerigo	Vespucci	1499 America Avenue	6/9/2007 12:00:00 AM
Bartolemeu	Dias	1481 Gold Coast Road	4/3/2005 12:00:00 AM
Jacques	Cartier	156 Canada Crescent	3/15/2007 12:00:00 AM
Christopher	Columbus	1492 America Avenue	7/24/2006 12:00:00 AM
David	Livingstone	1852 Victoria Falls	11/12/2005 12:00:00 AM
Ferdinand	Magellan	1520 Pacific Heights	8/29/2007 12:00:00 AM
Francisco	Pizarro	12 Lima Lane	10/5/2006 12:00:00 AM
Francisco	Vasquez de Coronado	98 Arizona Avenue	8/23/2006 12:00:00 AM

Connecting to a database

In the WebMatrix examples we have seen here, the database is a SQL Server Compact database which is included in the application. To access the database, the code simply opens the database file.

```
var db = Database.Open("gcutourswm");
```

In many cases, however, applications which use databases are **separate** from the application. Often the application is not even on the on the same computer as the database, and they communicate over a network. Many users or applications may be making use of the same database at the same time. In this case, the application is known as a **client**, which makes use of a database **server**..

An application which wants to make use of a database must **connect** to the database. The application usually needs to specify the following information:

- the **database driver** – a piece of software which provides an interface between the application language (e.g. Java, Visual Basic) and a specific **type** of database (e.g. Access, Oracle, JavaDB)
- the **network address** or **name** of the computer where the database is located
- the **name** of the specific **database** on that computer (e.g. the *gcutours* database)

These are often combined into a string of characters called the **database URL**.

It's possible to connect to just about *any* type of database from just about *any* programming language, as long as you have a suitable database driver.

Once the connection has been made the application usually communicates with the database by sending SQL queries.

Connecting to a JavaDB database

The GCUTours case study application is an enterprise application written in Java, which connects to a JavaDB database. The following lines of Java code sets up the connection information. In this case, the database is on the same computer as the application, so the network name is *localhost*[1].

```
private static String DRIVER_NAME =
"org.apache.derby.jdbc.ClientDriver";
private static final String DATABASE_URL =
"jdbc:derby://localhost:1527/C:/work/netbeansfirstyear/Databases/
JavaDB/gcutours";
```

[1] 1527 is the TCP port

Appendix: Code listings for WebMatrix web pages

users.cshtml

```
@{
    var db = Database.Open("gcutourswm");
    var selectQueryString = "SELECT * FROM Users";
 }

<!DOCTYPE html>
<html>
  <head>
    <title>GCUTours users</title>
  </head>
  <body>
    <h1>GCUTours users</h1>
    <table>
        <thead>
            <tr>
                <th>First name</th>
                <th>Last name</th>
                <th></th>
            </tr>
        </thead>
        <tbody>
            @foreach (var row in db.Query(selectQueryString)){
             <tr>
                  <td>@row.firstname</td>
                  <td>@row.lastname</td>
                  <td><a href="userdetail.cshtml?username=
                    @row.username">details</a></td>
             </tr>
            }
        </tbody>
    </table>
  </body>
</html>
```

userdetail.cshtml

```
@{
    var username = Request["username"];
    var db = Database.Open("gcutourswm");
    var selectQueryString =
        "SELECT * FROM Users WHERE username=@0";
    var result = db.Query(selectQueryString, username).FirstOrDefault();
 }

<!DOCTYPE html>

<html lang="en">
    <head>
        <meta charset="utf-8" />
        <title>GCUTours user detail</title>
    </head>
    <body>
     <h1>GCUTours user detail</h1>
        <h2>@result.lastname, @result.firstname</h2>
        <p>Username: @result.username</p>
        <p>Joined: @string.Format("{0:d}", result.datejoined)</p>
        <p><a href="users.cshtml">Back to list</a></p>
    </body>
</html>
```

tours.cshtml

```
@{
    var db = Database.Open("gcutourswm");
    var selectQueryString =
        @"SELECT departuredate, offer, packagename, Tours.packageID
        FROM Tours, Packages
        WHERE Tours.packageID = Packages.packageID";
 }

<!DOCTYPE html>
<html>
  <head>
    <title>GCUTours current tours</title>
  </head>
  <body>
    <h1>GCUTours current tours</h1>
    <table>
        <thead>
            <tr>
                <th>Departure date</th>
                <th>Discount</th>
                <th>Package</th>
            </tr>
        </thead>
        <tbody>
            @foreach (var row in db.Query(selectQueryString)){
             <tr>
                  <td>@string.Format("{0:d}", row.departuredate)</td>
                  <td>@row.offer%</td>
                  <td><a href="package.cshtml?packageID=
                    @row.packageID">@row.packagename</a></td>
             </tr>
            }
        </tbody>
    </table>
  </body>
</html>
```

package.cshtml

```
@{
    var id = Request["packageID"];
    var db = Database.Open("gcutourswm");
    var selectQueryString =
        "SELECT * FROM Packages WHERE packageID=@0";
    var result = db.Query(selectQueryString, id).FirstOrDefault();
 }

<!DOCTYPE html>

<html lang="en">
    <head>
        <meta charset="utf-8" />
        <title>GCUTours package  detail</title>
    </head>
    <body>
     <h1>GCUTours package detail</h1>
        <h2>@result.packagename</h2>
        <img src="@result.tourpicurl" />
        <p>Location: @result.location</p>
        <p>Price (adult): £@result.adultprice</p>
        <p>@result.description</p>
        <p><a href="tours.cshtml">Current tours</a></p>
    </body>
</html>
```

bookings.cshtml

```
@{
    var db = Database.Open("gcutourswm");
    var selectQueryString = "SELECT * FROM Bookings";
 }

<!DOCTYPE html>
<html>
  <head>
    <title>GCUTours bookings</title>
  </head>
  <body>
    <h1>GCUTours bookings</h1>
    <table>
        <thead>
            <tr>
                <th>Booking ID</th>
                <th></th>
                <th></th>
            </tr>
        </thead>
        <tbody>
            @foreach (var row in db.Query(selectQueryString)){
             <tr>
                  <td>@row.bookingID</td>
                  <td><a href="editbooking.cshtml?bookingID=
                    @row.bookingID">edit</a></td>
             </tr>
            }
        </tbody>
    </table>
  </body>
</html>
```

editbooking.cshtml

```
@{
    var id = Request["bookingID"];
    var adults = Request["adults"];
    var children = Request["children"];

    var db = Database.Open("gcutourswm");
    var selectQueryString =
            @"SELECT * FROM Bookings, Users
            WHERE Bookings.username = Users.username
            AND bookingID = @0";
    var result = db.Query(selectQueryString, id).FirstOrDefault();

    if (IsPost) {
        var updateQuery = @"UPDATE Bookings
            SET adults = @0, children = @1
            WHERE bookingID = @2";
        db.Execute(updateQuery, adults, children, id);

        Response.Redirect(@Href("~/bookings.cshtml"));
    }
 }

<!DOCTYPE html>

<html lang="en">
    <head>
        <meta charset="utf-8" />
        <title>GCUTours edit booking</title>
    </head>
    <body>
     <h1>GCUTours edit booking</h1>
        <form method="post" action="">
            <input type="hidden" name="bbookingID" value="@id" />
            <p>Booking ID: @result.bookingID</p>
            <p>User: @result.lastname, @result.firstname</p>
            <p>TourID: @result.tourID</p>
            <p>Adults: <input name="adults" type="text" size="5"
                    value="@result.adults" /></p>
            <p>Children: <input name="children" type="text" size="5"
                    value="@result.children" /></p>
            <p><input type="submit" value="Update" /></p>
        </form>
    </body>
</html>
```