Build a Subfile with PHP

Module 1: Simple Database Listing, with CSS

Contents

Customer File Format

In all of this workshop's exercises, we will use a sample customer table supplied by Zend, which is found in the Zend Server product library: ZENDSVR6, file name is SP_CUST. Below is the record format for this table.

Record Format

ZENDSVR6/SP_CUST

Name		Type	Len	Prec	Scale	Nulls?	Description
CUST_ID	1	Numeric		5	0	N	CUSTOMER ID
COMPANY	2	Char	30			N	COMPANY NAME
FIRSTNAME	3	Char	20			N	CONTACT FIRST NAME
LASTNAME	4	Char	20			N	CONTACT LAST NAME
CIVIL	5	Char	1			N	
ADDRESS	6	Char	30			N	ADDRESS
ADDR2	7	Char	30			N	ADDRESS 2
CITY	8	Char	15			N	CITY
STATE	9	Char	20			N	STATE
ZIP	#	Char	10			N	ZIP
COUNTRY	#	Char	20			N	COUNTRY
PHONE	#	Char	15			N	PHONE
FAX	#	Char	15			N	FAX

Basic customer listing using paragraphs

Note: In this exercise, all source files you will be working with are contained in the `sfl_01` folder.

The purpose of this exercise is to learn how to access DB2 data from PHP, using SQL, and display a list of records in the browser, with minimal formatting.

In this exercise you will do the following things:

1. Connect from PHP to DB2
2. Prepare an SQL query to retrieve all records from the customer table (SP_CUST)
3. Loop through the query results to retrieve each record from the query result set.
4. Format certain pieces of data from the record for display (name, address)
5. Output each customer's information in a paragraph format.

After finishing this part, in the next exercise you will add CSS to make the display format more interesting.

A skeleton for this exercise is provided in file `/slf_01/sfl_01.php`. This contains most of the HTML, so you will only need to write the PHP code. Comments are already supplied, but you need to write the PHP code that does what the comment says.

⇨ Open file `sfl_01.php`

It should look like this:

```
<!DOCTYPE html>
<html>
      <head>
            <title>Customer Listing</title>
      </head>

      <body>
            <h1>Customer Listing</h1>

      <?php
      // Connect to db2

      // Prepare and run query

      // Loop through query results, format output variables, and
      // display an HTML paragraph for each customer record.
```

```
       ?>

             <!-- The following paragraph (<p></p>) will be repeated
                   for each DB row retrieved -->
             <p>
                   <b>Cust ID</b>: <!-- php to output cust id here -->
                   <br>
                   <b>Name</b>: <!-- php to output name here -->
                   <br>
                   <b>Address</b>:
                   <br>
                   <!-- php to output address here -->
             </p>

       <?php
       // Close the while loop
       ?>

       </body>

</html>
```

The first several lines contain pure HTML, including

- The open <html> tag, which defines the beginning of the html document structure
- The <head> section, with a <title> that will show up in the title bar of the browser. The rest of the <head> section, up to </head> is information for about the document for the browser, and does not show up within the page contents. Later you will add a CSS style sheet in the <head> section.
- The open <body> tag, which defines the beginning of the visible content of the page
- A level-1 heading (<h1>Customer Listing </h1>) which will display a large-text title within the page.

After this comes a PHP block, started by `<?php`, and terminated by `?>`. Within this block, you will write PHP code to connect to DB2, run an SQL query, and loop through the results. There are two PHP comments, denoted by lines starting with `//`. You will add your code below these comments.

Below this, the ?> ends the PHP code block, and we are back to HTML code. There is an HTML comment, delimited by <!-- and -->, telling you that the following paragraph (delimited by <p> and </p>), will be output by PHP for each customer record. Within this paragraph are

1. Text labels for the values that will be output, surrounded by <b></b> (bold) tags.
2. HTML comments telling you where you will add PHP snippets to output customer variables.
3. Line break (<br>) tags after each output line. Without these, all the data would appear on one line (because browsers will render all repeating white space into a single visible space on the screen).

Below the paragraph, there is one last PHP block where you will close the while loop which fetches the customer records. The last items are the closing tags for the body section (</body>) and the html document (</html>).

Connecting to DB2

⇨ In the first PHP block, add the lines below.

⇨ Be careful to use the exact syntax specified, including case and punctuation. Remember to put a semi-colon at the end of each PHP statement.

```
<?php
// Connect to db2
$options = array('i5_naming'=> DB2_I5_NAMING_ON,
                    'i5_libl' => 'ZENDSVR6');

$conn = db2_connect("*LOCAL","jvlabsXX","jvlabsXXpw", $options)
      or die("Connection failed! ". db2_conn_errormsg());
```

⇨ On the db2_connect() parameters, for " jvlabsXX " and " jvlabsXXpw", substitute your team's user ID and password

This code consists of two PHP statements. Each statement here spans two lines. PHP statements are terminated by a semi-colon.

- The first PHP statement creates an array variable named $options, which is used to define a list of connection options to be passed as a parameter to the db2_connect() function.

- The next statement creates a variable named $conn, which will contain a reference to the db2 connection returned by the db2_connect() function. We will need this for all subsequent db2 operations.

Variable $options is defined as an associative array (an array with alpha keys), having two elements:

1. **`'i5_naming'`**: Setting this option to the constant value `DB2_I5_NAMING_ON` tells db2 to use the IBM i native naming convention for qualified object names, with a slash (/) between the library and object name (i.e. MYLIB/MYFILE), versus the standard SQL naming convention which uses a dot (i.e. MYLIB.MYFILE). More importantly, when setting this option ON, unqualified object names will be resolved using the connection's library list. Which brings us to the second option…
2. **`'i5_libl'`**: This defines the library list during this connection. In this workshop, we will be using a single table for database access, named SP_CUST, in the library ZENDSVR6. Hence, there is only one library in the library list. To add more libraries, separate them with commas or spaces. You couls also not include this option, in which case the library list would be determined by the user profile's job description.

Here is some background on PHP variables and arrays:

PHP variables

- All variables in PHP start with a dollar sign ($). In the code above, $options and $conn are both variables.

PHP arrays

- Arrays are a very important and powerful feature of PHP, and are very different than RPG arrays. They are more like data structures because each element can have a different data type.
- Arrays can be indexed both by numbers and character strings, which makes them very handy for storing lists of name/value pairs (like a LOOKUP table in RPG).
- You can define an array using the array() function, and specify an initial list of name/value pairs as parameters separated by commas.
- Names and values are separated by => (the array element assignment operator). For example:
```
array('name1' => 'value1',
      'name2' => 'value2')
```
- To retrieve an array element from an array, specify the array name followed by the key value surrounded by square brackets:

```
        echo $options['i5_libl']   // displays 'ZENDSVR6'
```

- Any number of elements can be specified on the array() function. Arrays are open-ended – i.e. you do not specify how many elements an array can hold - you can always add more elements to an array. To add a new element, specify the key and use the assignment operator:

```
        $options['new_element'] = 'some value';
```

Now back to the task at hand…

The db2_connect () function

Let's look again at the code that calls the db2_connect() function:

```
$conn = db2_connect("*LOCAL", " jvlabsXX", " jvlabsXXpw", $options)
          or die("Connection failed! ". db2_conn_errormsg() );
```

⇨ Change both instances of the string jvlabsXX so that XX is replaced with your team number.
  - This is your user ID and password for the IBM i. Any valid user/pswd can be used, provided they have authority to perform the needed operations on the database tables being accessed.

php.net

The full reference for the `db2_connect()` function can be found at php.net: http://us1.php.net/manual/en/function.db2-connect.php . This site is the official reference for all aspects of the php language. You should bookmark the home page of this site – as a PHP programmer you will need to visit www.php.net quite often. PHP has many hundreds of commonly used built-in functions. You can search for a function's reference using the search box on the home page.

This function has 3 required parameters, and 1 optional parameter:

```
  db2_connect(<host>, <user-id>, <password>, [ <options array>] )
```

where:

- <host> can be specified by an IP address, or if the database is on the same host as the web server, you can specify '*LOCAL' or 'localhost'
- <user-id> and <password> are valid IBM i user and password. Use your team's user id and password.
- [<options>] is the optional array of connection settings that we looked at earlier.

The second line (`or die()`…) says that if the connection cannot be established, then stop processing and display the message specified as a parameter to the `die()` function. (look up 'die' on php.net). The parameter passed to die() is a single character string. In this case, we pass it a literal string "Connection failed!", plus another string concatenated on the end of this. The dot operator (.) in PHP means concatenation, and the value being concatenated is the message returned by the db2_conn_errormsg() function. This will retrieve the text associated with the CPF error message – for example: ' Authorization failure on distributed database connection attempt. ' if an invalid user or password is supplied.

Prepare and Execute the SQL query

⇨ Below these lines, add the following 3 statements:

```
            // Prepare and run query
            $sql = 'SELECT * FROM SP_CUST';
            $stmt = db2_prepare($conn, $sql);
            db2_execute($stmt);
```

- The first statement creates a string variable named $sql containing the SQL select statement to retrieve all rows and columns from SP_CUST.
- The second statement calls the db2_prepare() function, which passes the sql string ($sql) to the db2 connection we created earlier ($conn). The db2_prepare function tells db2 to create an access plan for the query, and returns a reference to a prepared statement ready for execution.
- The third statement calls db2_execute() to run the query. The database rows returned by the query are placed in holding area called a "result set", associated with the prepared statement ($stmt). Our next step will be to fetch each result set row using a loop construct.

Fetching the Rows of the Result Set

We will use the db2_fetch_assoc() function to retrieve each row as an associative array (an array indexed by alpha values) . This is a handy function, since the array elements returned will be keyed by the field names for each value.

⇨ Add the following lines.

⇨ Be careful to follow the syntax exactly. Use double quotes or single quotes exactly as specified below. Note the use of **square brackets** for the array indices after $row – those are **NOT parentheses**. i.e. use $row[ ], *not* $row( ).

```
        // Loop through query results, format output variables, and
        // display an HTML paragraph for each customer record.
        while ($row = db2_fetch_assoc($stmt)) {
              // Store formatted values in PHP variables
              $custId = $row['CUST_ID'];
              $name = $row['FIRSTNAME'] . ' ' . $row['LASTNAME'];
              $address = "{$row['ADDRESS']}    {$row['ADDR2']}<br>
                           {$row['CITY']}, {$row['STATE']} {$row['ZIP']} {$row['COUNTRY']}";
```

The first statement here defines the beginning of a while loop. While the condition in the parentheses is true, the loop continues. But what is inside the parentheses? It's an assignment statement! In this case, the statement is executed, and whatever value gets assigned to the variable is tested as either true or false. The db2_fetch_assoc() function will either return the next row of the query result, or, if no more rows are found, it will return false. In PHP, anything except a false value is considered true, so if an array is returned, this is considered true and keeps the loop running.

Note the brace { at the end of this line – it defines the end of the while statement and the beginning of the loop body. Later, we will terminate the loop body with a closing brace }. But first we will format some of the data retrieved into variables, and output the paragraph of customer information for each row.

Below the `while` statement are 3 statements to create variables for the formatted output values. The variable $row is an array containing all the fields from one customer record. It is an associative array, with keys equal to the field names in SP_CUST. So, to get the CUST_ID field value from the current row, use `$row['CUST_ID']`.

For $name, we will concatenate the FIRSTNAME and LASTNAME fields, with one space between them.

For $address, we are concatenating all of the address fields, rendered by the browser as 3 lines by using the <br> (line break) tags. In this case, concatenation is accomplished using a different PHP technique known as variable interpolation.

Strings and Variable Interpolation

PHP character string literals come in two flavors: `'surrounded by single quotes'` and `"surrounded by double quotes"`. Generally you can use either single or double quotes for string literals. But double-quoted strings have a very handy feature that makes concatenation very simple and easy to read. Double-quoted strings can have variables embedded within them. For example:

```
$name = 'Peter Smith';
echo "Hello, my name is $name. What is your name?";
```

Running this will display the following:

```
Hello, my name is Peter Smith. What is your name?
```

The current value of $name gets inserted within the string literal before the echo happens. The $ in front of the variable name allows the PHP interpreter to recognize this as a variable whose value is retrieved and inserted into the string. Remember, this only works with double-quoted strings. If you used single quotes, you'll get a message like

```
Hello, my name is $name. What is your name?
```

This is a simple technique for use with scalar variables, but what about array elements? For example, what if we want to insert the value of `$row['ADDRESS']` into the string? Because of the single quotes and brackets, this will confuse the parser. To make this work, surround the entire variable with curly braces `{ }` like this:

```
echo "I live in {$row['CITY']}, in the state of {$row['STATE']}.";
```

That's exactly what we're doing on the lines that format the address:

```
$address = "{$row['ADDRESS']}  {$row['ADDR2']}<br>
             {$row['CITY']}, {$row['STATE']} {$row['ZIP']} {$row['COUNTRY']}";
```

Outputting the Customer Record

Now that we've fetched a row and formatted some data for display, let's write it out in the paragraph.

⇨ In the lines below, remove the HTML comments that instruct you to output php data, as highlighted below:

```
            <!-- The following paragraph (<p></p>) will be repeated
                 for each DB row retrieved -->
            <p>
                 <b>Cust ID</b>: <!-- php to output cust id here --> <br>
                 <b>Name</b>: <!-- php to output name here --> <br>
                 <b>Address</b>: <br>
                 <!-- php to output address here -->
            </p>
```

⇨ Replace these comments with the following PHP snippets, which echo the variables that were populated above:

```
            <!-- The following paragraph (<p></p>) will be repeated
                 for each DB row retrieved -->
            <p class="box">
                <b>Cust ID</b>: <?php echo $custId ?> <br>
                <b>Name</b>: <?php echo $name ?> <br>
                <b>Address</b>: <br>
                <?php echo $address ?>
            </p>
```

One last thing to do, and then you can run this PHP script and, if no errors, see the results.

End the While Loop

⇨ Add the closing brace }, as shown highlighted, below.

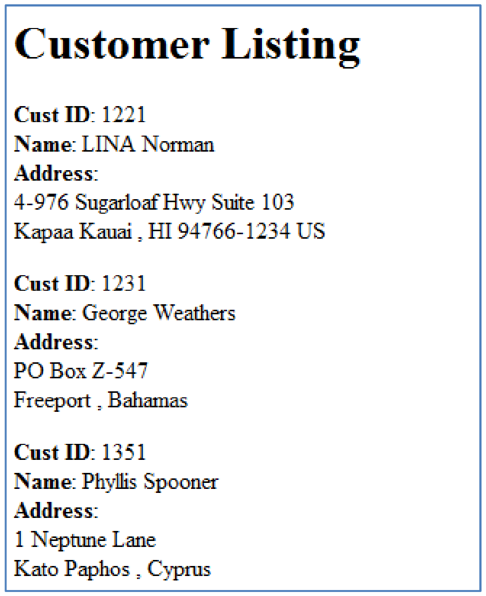
```
        <?php
        // Close the while loop
        }
        ?>
        </body>
</html>
```

You are finally ready to run the first script.

Run the Script

⇨ ***Save your changes*** by pressing Ctrl+S, or by clicking the Save icon in the toolbar, in the upper left.
⇨ Open your browser of choice, and type in the following URL:
`http://common1.frankeni.com:10080/jvlabs/teamXX/phpsf/sfl_01/sfl_01.php`
where the XX in phpsfXX should be ***replaced with your team number***

You should see something like this in the browser:



Customer Listing

Cust ID: 1221
Name: LINA Norman
Address:
4-976 Sugarloaf Hwy Suite 103
Kapaa Kauai , HI 94766-1234 US

Cust ID: 1231
Name: George Weathers
Address:
PO Box Z-547
Freeport , Bahamas

Cust ID: 1351
Name: Phyllis Spooner
Address:
1 Neptune Lane
Kato Paphos , Cyprus

There are a total of 54 records that will be displayed. If you do not see the data, check for source code syntax errors in the editor (they should be highlighted in red), and if you're still having problems, ask the instructor or an assistant to help troubleshoot your script.

If you do see the customer records, congratulations! You've successfully written a PHP script that connects to DB2, runs an SQL query, fetches all of the result set rows, and displays database information in the browser!

Styling with CSS

The listing above is aesthetically pretty bland, especially for a web page. In this part of the exercise we will introduce Cascading Style Sheets (CSS), and apply some simple styling rules to this page to make it look much nicer, and hopefully more intuitive for our users.

CSS is a language that allows us to define sets of styling rules and apply them to parts of the HTML document. Styling rules specify how document elements should be displayed in the browser, such as fonts, sizes, colors, borders, and other visual aspects of the document. Once a style rule is defined, we can apply it to various parts of a web page or an entire web site, by associating the rule with a *CSS selector*. Here's an example of a selector and some CSS rules:

```
p {
   font-size: 12pt;
   font-family: arial;
   color: navy
}
```

This rule states that all <p> tag's contents should be rendered with a font of 12 points, Arial, with navy (dark blue) text color. In this case, `p` is the selector (selecting all paragraphs in the document), and everything between the braces being the styling rule that is applied to the selected elements. A group of CSS rules is referred to as a ***style sheet***.

CSS rules have a fairly straight-forward syntax:

```
<selector> {
   <property1> : <value1>;
   <property2> : <value2>;
   etc…
}
```

Note the use of colons `(:)` between properties and values, and semi-colons `(;)` between each `property:value` pair.

To see a list of all CSS properties and their possible values, go to http://www.w3schools.com/cssref/.

Selectors can be any as the following:

- Any valid HTML tag name (e.g.: `body, table, tr, p, input`, etc…)
- A user-defined ***class*** name, which can be used to either broaden or narrow the elements that are selected
- A user-defined, unique ***id*** name, which can be used to select exactly one element in the HTML document.

CSS Class selectors

Using class selectors is a very common and useful technique. The term class in CSS refers to a grouping of elements under a name of our own devising, for the purpose of styling (the term class in this context has nothing to do with classes in object oriented programming). To specify a class selector, decide on an appropriately descriptive name for your class, and precede it with a period in your style sheet. You can specify a class selector in conjunction with HTML tag names or as a stand-alone class selector. Here is an example:

```
p.blue-arial { font-family: arial;  color: blue;  background-color: silver;  }
```

The above rule states that all <p> tags with a class of "blue-arial" will have these styling properties. But what does it mean, *all <p> tags with a class of "blue-arial"*? In order to make use of a class selector, the html tag must specify an attribute named ***class***, with a value equal to the class name defined in the style sheet. For example, to use the above style rule, specify a <p> tag like this:

```
<p class="blue-arial">Paragraph content… </p>
```

In the above example, the CSS rule was defined with an html tag name (p) and a class name (blue-arial). In this case, this class can only be specified on <p> tags. It won't have any affect if we specify `class="blue-arial"` on any other html tags. But another way to use class selectors is to NOT specify what tag it applies to, just start with a period, like this:

```
.blue-arial { font-family: arial;  color: blue;  background-color: silver;  }
```

Now we can apply this class to *any* html tag. We could use this as follows, and all would render with the properties specified:

```
<p class="blue-arial">
<td class="blue-arial>
<input class="blue-arial>
<h3 class="blue-arial>
```

So, if we use a tag name before the class name (p.blue-arial), this will narrow the scope of the selection. And if we do not specify a tag name (.blue-arial), this broadens the scope of the selection.

Applying CSS to the Customer Listing

Let's apply what we've learned so far about CSS to our customer listing application. In order to define a style sheet for our web page, we must surround the style rules with a `<style> </style>` tag pair. (In the next exercise, we will see how to define our CSS externally, i.e. in a separate .css file, which is a more reusable way of defining style rules).

⇨ At the top of your php file, within the <head> section, add the <style> </style> tags, and add the CSS rules as specified below.

```
      <head>
            <title>Customer Listing</title>

            <style>
                  body { font-family: verdana; color: brown }
                  b {  color: #4545FF; font-family: calibri; font-weight: normal}
                  h1 { text-align: center; }
                  .ul {text-decoration: underline }
                  p.custRec { border: 2px solid #4545FF; background-color: #FFF0BF;
                              width: 40%; margin: 9px; padding: 3px }
            </style>
      </head>
```

Note the following:

- We specified two class-based selectors: `.ul`, which can be used on any tag, and `p.custRec`, which we will apply to our customer record paragraphs. For the class-based rules, we will need to add class="…" to the appropriate html tags.
- We used hex color codes on some of the rules (color: #4545FF). For an excellent explanation of CSS colors, start here: http://www.w3schools.com/cssref/css_colors.asp and look at all the color-related lessons that follow.

- The <b> tag (i.e. b for bold) by default provides a `font-weight: bold` styling, with no CSS involved. But the rule we specified for bold completely overrides the default styling, rendering <b> content as normal weight, with a blue color.
- Note that the border property has 3 values specified (2px, solid, and #4545FF), which correspond to the border-width, border-style, and border-color properties, respectively. Each of these properties could be specified separately, if desired.

Three of these rules apply to html tags globally (body, p, and h1). So they will be applied automatically with no other code changes. But for the class-based selectors, we now need to add class attributes to the appropriate html tags.

Adding Class Attributes to HTML Tags

⇨ In the while loop, add the class attributes as highlighted below:

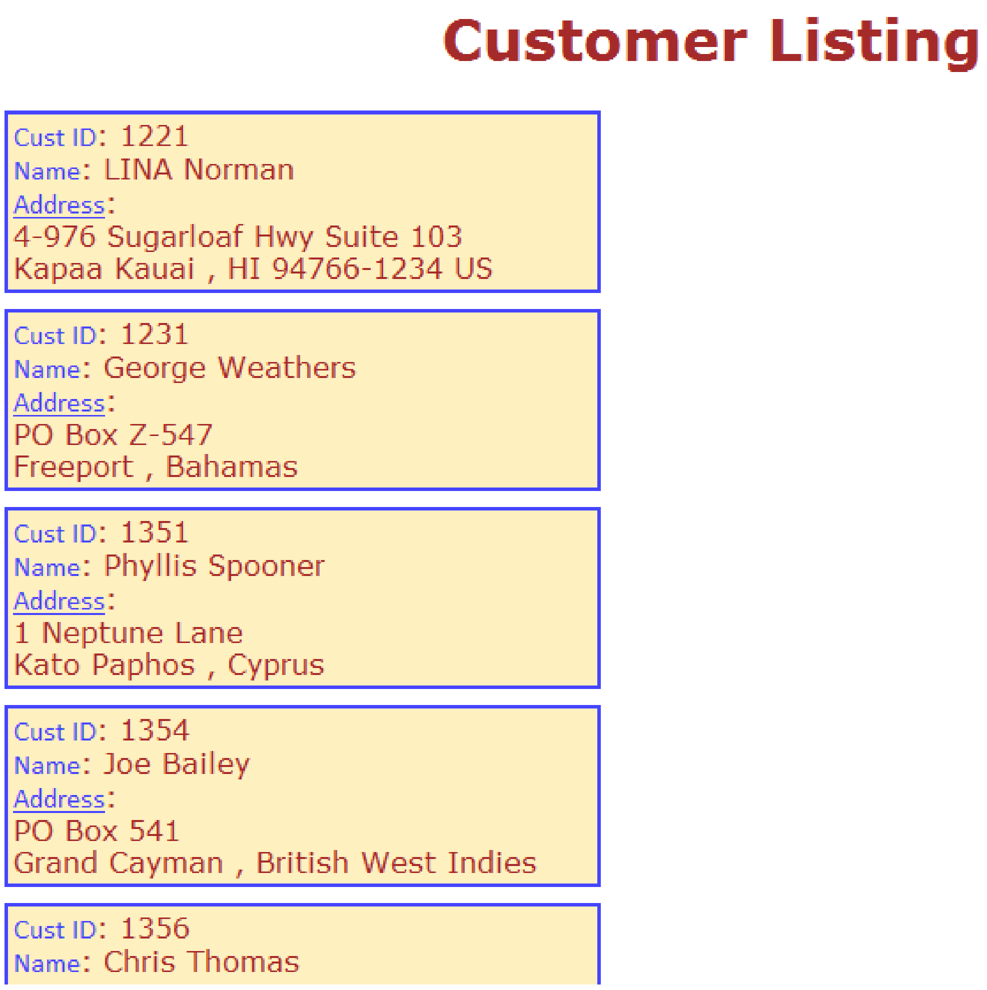
```
            <!-- The following paragraph (<p></p>) will be repeated
                 for each DB row retrieved -->
            <p class="custRec">
                 <b>Cust ID</b>: <?php echo $custId ?><br>
                 <b>Name</b>: <?php echo $name ?><br>
                 <b class="ul">Address</b>: <br>
                 <?php echo $address ?>
            </p>
       <?php
       endwhile;
       ?>
```

⇨ Save your changes

⇨ Refresh the display in your browser. You can either click the refresh icon in the address bar, or press F5 to refresh.

Note: Always remember to save your source code changes, and refresh your browser in order to see your changes applied.

You should see something like the following now:



Customer Listing

Cust ID: 1221
Name: LINA Norman
Address:
4-976 Sugarloaf Hwy Suite 103
Kapaa Kauai , HI 94766-1234 US

Cust ID: 1231
Name: George Weathers
Address:
PO Box Z-547
Freeport , Bahamas

Cust ID: 1351
Name: Phyllis Spooner
Address:
1 Neptune Lane
Kato Paphos , Cyprus

Cust ID: 1354
Name: Joe Bailey
Address:
PO Box 541
Grand Cayman , British West Indies

Cust ID: 1356
Name: Chris Thomas