THE UNIVERSITY of ADELAIDE

CRICOS PROVIDER 00123M

Lecture 26 – Integrating Databases in Web Applications

Web & Database Computing

adelaide.edu.au

seek LIGHT

Databases for application data management

- Databases can help manage data requests.
- Task
  - *For a given user, we should be able to display their blog posts on a page.*
- Design Plan
  - Server
    - Store Users and Blog Entries in database
    - When GET /friends.json request received with username parameter, return dates and contents of all posts from user
  - Client
    - Use Text Box and Button to send request for posts by a user.
    - Dynamically fill page with blog posts

Database

Consider the following database:

```
MariaDB [travel_blog]> describe users;
+----------+-------------+------+-----+---------+-------+
| Field    | Type        | Null | Key | Default | Extra |
+----------+-------------+------+-----+---------+-------+
| user_id  | char(10)    | NO   | PRI | NULL    |       |
| username | varchar(20) | YES  |     | NULL    |       |
+----------+-------------+------+-----+---------+-------+

MariaDB [travel_blog]> describe blog_entries;
+-----------+----------+------+-----+---------+-------+
| Field     | Type     | Null | Key | Default | Extra |
+-----------+----------+------+-----+---------+-------+
| post_id   | char(20) | NO   | PRI | NULL    |       |
| author_id | char(10) | YES  |     | NULL    |       |
| date      | datetime | YES  |     | NULL    |       |
| contents  | longtext | YES  |     | NULL    |       |
+-----------+----------+------+-----+---------+-------+
```

What data would we need to display blog posts for a given user?

Database

Consider the following database:

```
MariaDB [travel_blog]> describe users;
+----------+-------------+------+-----+---------+-------+
| Field    | Type        | Null | Key | Default | Extra |
+----------+-------------+------+-----+---------+-------+
| user_id  | char(10)    | NO   | PRI | NULL    |       |
| username | varchar(20) | YES  |     | NULL    |       |
+----------+-------------+------+-----+---------+-------+

MariaDB [travel_blog]> describe blog_entries;
+-----------+----------+------+-----+---------+-------+
| Field     | Type     | Null | Key | Default | Extra |
+-----------+----------+------+-----+---------+-------+
| post_id   | char(20) | NO   | PRI | NULL    |       |
| author_id | char(10) | YES  |     | NULL    |       |
| date      | datetime | YES  |     | NULL    |       |
| contents  | longtext | YES  |     | NULL    |       |
+-----------+----------+------+-----+---------+-------+
```

Now how do we combine that data?

Database

Consider the following database:

```
MariaDB [travel_blog]> describe users;
+----------+-------------+------+-----+---------+-------+
| Field    | Type        | Null | Key | Default | Extra |
+----------+-------------+------+-----+---------+-------+
| user_id  | char(10)    | NO   | PRI | NULL    |       |
| username | varchar(20) | YES  |     | NULL    |       |
+----------+-------------+------+-----+---------+-------+

MariaDB [travel_blog]> describe blog_entries;
+-----------+----------+------+-----+---------+-------+
| Field     | Type     | Null | Key | Default | Extra |
+-----------+----------+------+-----+---------+-------+
| post_id   | char(20) | NO   | PRI | NULL    |       |
| author_id | char(10) | YES  |     | NULL    |       |
| date      | datetime | YES  |     | NULL    |       |
| contents  | longtext | YES  |     | NULL    |       |
+-----------+----------+------+-----+---------+-------+
```

Let's write a query for this.

Exercise

Setting up the server

- First we need to install the mysql module for our server
  - Run `npm install --save mysql`
- Next we need to add code to our app.js to connect with the SQL server:

```
/* use mysql in this app */
var mysql = require('mysql');

/* create a 'pool' (group) of connections that can be used for interacting
with the database, and adding the connection to our SQL server */
var dbConnectionPool = mysql.createPool({ host: 'localhost',
                                          database: 'travel_blog'});

var app = express();

/* middleware for accessing database.  We need access to the database to be
available *before* we process routes in index.js, so this code needs to be
*before* the app.use('/', routes);  Express will run this function on every
request and then continue with the next module, index.js. So for all
requests that we handle in index.js, we'll be able to access the pool using
req.pool */
app.use(function(req, res, next) {
    req.pool = dbConnectionPool;
    next();
});
```

Server routes

- Setup is done. Now for the processing.
- Our server needs to provide a route for:
  - Getting a list of posts for the given user
- New route in index.js
  - The user will specifying a username to show get posts for and sending this information to the server to get posts from the database. Do we need a POST or GET?
- Steps when request is received:
  - Connect to the database
  - Query the database for blog posts for that user
  - Release the connection to the database
  - Send Response

Connecting to the Database

```
router.get('/posts.json', function(req, res) {
    //Connect to the database
    req.pool.getConnection(function(err,connection) {
        if (err) throw err;
    });
});
```

- Recall that our middleware sets req.pool to our database pool that we created in the app.js file.
- getConnection() is a function in the mysql module that connects to the mysql database.
- The callback function will have err and connection as parameters. *connection* is the connection to the database.

Querying the database

Use the query from earlier to get the blog post contents and their dates for a given user

```
router.get('/posts.json', function(req, res, next) {

    //Connect to the database
    req.pool.getConnection( function(err,connection) {
        if (err) {
            throw err;
            res.json({});
        }
        var username = req.query.username;
        var query = "SELECT date,contents "+
                    "FROM users INNER JOIN blog_entries "+
                    "ON users.user_id=blog_entries.author_id "+
                    "WHERE users.username LIKE ?";
        connection.query(query, [username], function(err, rows, fields) {
            connection.release(); // release connection
            res.json(rows); //send response
        });
    });
});
```

Querying the database

- The query() method takes
  - an SQL query string,
  - an array of parameters,
  - and the function to run when the query returns results.
- If the query requires data from the client to run (or other external sources), ? placeholders are added in their place.
- The ? placeholders are replaced with the values in the parameters array.
- Using placeholders with data like this keeps our SQL secure.

```
var username = req.query.username;
var query = "SELECT date,contents "+
            "FROM users INNER JOIN blog_entries "+
            "ON users.user_id=blog_entries.author_id "+
            "WHERE users.username LIKE ?";
connection.query(query, [username], function(err, rows, fields) {
    connection.release(); // release connection
    res.json(rows); //send response
});
```

Releasing the database & sending the response

Release the connection to the database for other queries to use and
send the response

```
router.get('/posts.json', function(req, res, next) {

    //Connect to the database
    req.pool.getConnection( function(err,connection) {
        if (err) {
            throw err;
            res.json({});
        }
        var username = req.query.username;
        var query = "SELECT date,contents "+
                    "FROM users INNER JOIN blog_entries "+
                    "ON users.user_id=blog_entries.author_id "+
                    "WHERE users.username LIKE '?'";
        connection.query(query, [usernsme], function(err, rows, fields) {
            connection.release(); // release connection
            res.json(rows); //send response
        });
    });
});
```

Query response format

- In the callback function on the previous slide, we've sent the rows selected by the query directly on to the client:

```
connection.query(query, function(err, rows, fields) {
    connection.release(); // release connection
    res.json(rows); //send response
});
```

- What does this data look like and how can we use it on our client?
  - An array of RowDataPacket objects that we can send as JSON:

```
[ RowDataPacket { date: 2016-01-23T10:45:48.000Z,
                  contents: 'Lorem ipsum dolor' },
  RowDataPacket { date: 2016-02-15T02:51:32.000Z,
                  contents: 'Phasellus aliquam, leo' } ]
```

- How do we use this data in code?
  - Same way we access other values in objects, for example
  - `rows[0].contents` will give us the string `'Lorem ipsum dolor'`
  - `rows[1].date` will give us the date `2016-02-15T02:51:32.000Z`

Client HTML

We'll need:

- A Text Box and Button to send the request, and a div to fill with the responses.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Travel Blog</title>
    <meta charset="utf-8">
  </head>
  <body>
    <input id="username" type="text" maxlength="20" length="20" />
    <button onClick="getPosts()">Show Posts</button>
    <hr>
    <div id="posts"></div>
    <script  src="javascripts/main.js"></script>
  </body>
</html>
```

- We can run our AJAX when getPosts() called by button onClick event

Client AJAX

```
function getPosts() {

    // Get username input
    var input = document.getElementById('username');

    // Create AJAX request
    var blogRequest = new XMLHttpRequest();

    // What to do when AJAX response comes
    blogRequest.onreadystatechange = function() {
    if (blogRequest.readyState==4 &&
        blogRequest.status==200) {

        // get posts div and clear it
        var posts_div = document.getElementById('posts');
        while (posts_div.hasChildNodes()) {
            posts_div.removeChild(posts_div.childNodes[0]);
        }

        // although we sent JSON, XMLHttpRequest only
        // handles text and xml so we need to parse it
        var result = JSON.parse(blogRequest.response);
```

Client AJAX cont.

```
        // create new blogpost entries
        for (var i = 0; i < result.length; i++) {
            var current_post = document.createElement("DIV");
            current_post.innerHTML = '<p>'+new Date(result[i].date)+
                                     '</p>\n<p>'+result[i].contents+
                                     '</p>\n<hr>';
            posts_div.appendChild(current_post);
        }

    }
    }

    // Open connection
    blogRequest.open("GET",
                     'http://localhost:3000/posts.json?username='+
                     input.value,
                     true);

    // Send request
    blogRequest.send();

}
```

THE UNIVERSITY of ADELAIDE

Demo

Review

- Setting up express app to connect to a database
  - Creating a connection pool (allows multiple concurrent database connections)
  - Node mysql for interacting with database
    - getConnection()
    - query()
- Using the response on the client


THE UNIVERSITY
of ADELAIDE

What's happening?

Due

- Prac Exercise 8 Available
- **Group Project Handin Extended to Monday**
  - Allow for reviwing of marking rubrik
  - Allow for reviwing of E-R Feedback

Next Week

- Schemas & Conversion
- Further SQL