Web and Mobile App Development

Relational Database Security and Performance Considerations

Material created by:
David Augenblick, Bill Mongan, Dan Ziegler, Samantha Bewley, and Matt Burlick

SQL Injections

- A SQL injection is an attack upon a relational database in which an SQL statement is altered to modify the original intent of the query.
- It usually takes place in web applications in which the user is requested to input a parameter to be used in the dynamic construction of a query by the server (kind of like some of the apps we are developing this term).
- At this point, there exists an opportunity for an unscrupulous person to enter input to damage the intended query in order to obtain sensitive data, or modify the contents of the database along with other unwanted consequences.

SQL Injections

Some of the most common SQL injection consequences include:

- Obtaining of sensitive data
- Change or delete data
- Bypass authentication (enter a password protected system without using a password)
- Obtain or modify authorization information stored in the database

SQL Injections

How can this happen?

- SQL Injections can happen pretty easily, and in a number of ways
- For instance, by inputting text to alter the construction of the query
- Example: Original (intended) statement:

```
select * from someTable
    where owner = (some user inputted text)
     and someField = (another user input);
```

SQL Injections

How can this happen?

- What if the user typed an identity statement that will always calculate to “true”, i.e

```
'someValue' OR 'a'='a'
```

- Then the dynamically constructed query becomes:

```
select * from someTable
    where owner = (some user inputted text)
    and someField = 'someValue' OR 'a'='a' ;
```

- This will always calculate to true and thus will display the entire table (as if “`select * from someTable`” was the query

SQL Injections

How can this happen?

- Another possibility is if someone added a statement to the back end of the query, say to delete a table
- This is possible for servers where it is possible to enter multiple SQL queries, separated by a ;
- Example: Original (intended) statement:

```
select * from someTable
    where owner = (some user inputted text)
    and someField = (another user input);
```

SQL Injections

How can this happen?

- If the attacker enters this into the input box:

```
'someValue' ; drop table sometable;
```

- Then the dynamically constructed query becomes:

```
select * from someTable
    where owner = (some user inputted text)
    and someField = 'someValue' ; drop table someTable;
```

- Which will drop (delete) “someTable” in addition to generating a response to the original, intended query.

SQL Injections

How can we stop this!!!???

- A variety of other techniques can be used to launch an SQL injection
- A literature search will reveal some of these methods
- Most involve placement of a single ' mark in the data you want to include in your SQL

SQL Injections

How can we prevent this from happening?

- We will now discuss a common technique for preventing SQL Injections, called *escaping characters*
- We will focus on a node.js / MySQL environment to demonstrate this type of remediation
- By adding special “escape characters” to text input by a user we can indicate to the database that the characters between these special escape characters are to be treated as text and not SQL commands.
- Note that a variety of other injection and remediation strategies are possible; we are limiting our discussions to those that are defined here!

SQL Injections

How can we prevent this from happening?

- Example – escaping characters in node.js / MySQL using the MySQL connection's “escape” method:

```
var userID = 'user provided ID'; // may contain an injection
var sql = 'select * from userTable
    where ID='  + connection.escape(userID);

connection.query(sql, function(err, res)  {
   //stuff--------
   /stuff---------
}
```

Connection Pooling in MySQL

- By default, NodeJS is single threaded, however most SQL databases are designed to handle several queries at the same time.
- Therefore if multiple calls come in to your nodejs server, they are queued and for the next SQL request to be sent to the database, the prior one must have been completed.
- One solution to this issue is *pooling.*

Connection Pooling in MySQL

- You can pool connections to a relational (MySQL in this case) database to alleviate sharing of one single connection, or managing multiple connections.
- Connection pooling maintains a set of database connections so that a connection can be re-used after releasing it.
- Instead of creating a single connection (`mysql.createConnection`), a pool of connections is created
- The code for this operation using node.js and MySQL is listed on the next slide

Connection Pooling in MySQL

- Connection pooling example code using node.js and MySQL

```
// Setup of pool
var mysql = require('mysql');
var pool = mysql.createPool( {
       connectionLimit  : 20,
       host : localhost,
       user : 'your user id',
       password : 'your password',
       database : ' your database name‘
});
```

Connection Pooling in MySQL

- Connection pooling example code using node.js and MySQL

```
// Get a connection from the pool and query the database
pool.getConnection(function(err, connection)  {
   connection.query('select * from pet', function(err,rows) {
        connection.release();  // release the connection to the pool
        if (err)
            throw err;
        else
            console.log(rows);
   });
 });
```

Sharing Databases

- We all know that using a versioning system (git, etc..) is a good idea when collaborating on a project.
- But what about the database portion?
- Maybe there's a few options….
  - Use some remote SQL server
  - Add the ability to export/import dataset info.

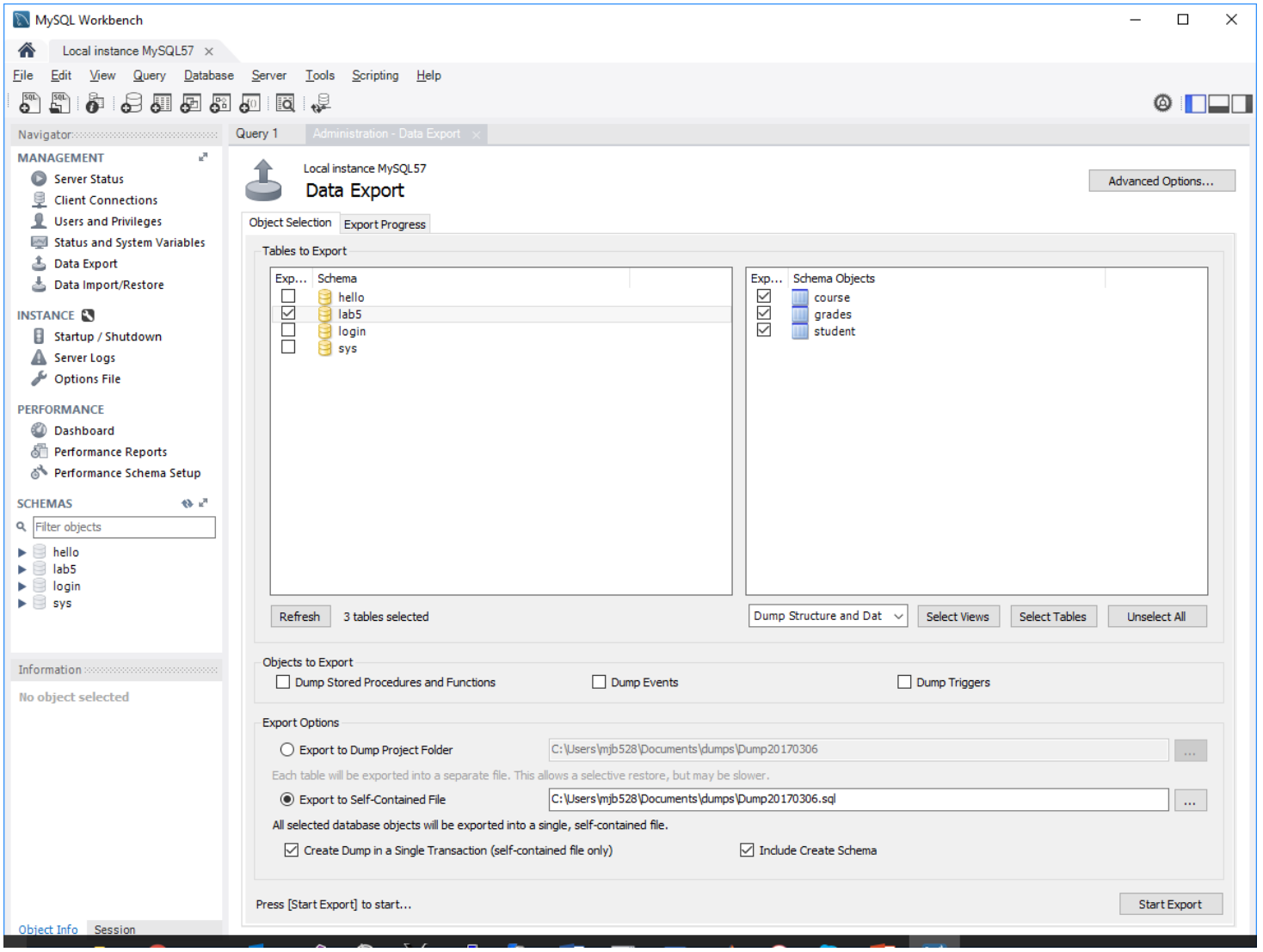
Exporting Databases

- Using a remote SQL database is attractive because then any changes to the data is seen by all immediately.
- But what if the structure changes?
- Another idea would be to dump your DB to your repository when you'd made critical changes and then others can import it when they're ready.

Exporting Databases

- Most MySQL distributions come with something called MySQL Workbench
- Here you can using a GUI to play around with your DB
- In addition, once you're in your DB you can export the data by:
  - Server Tab→Data Export
  - Then choose your database, the tables, and if you want structure and/or data to be exported
  - You can also choose to export to a self-contained file and to include the create schema.

Exporting Databases



Importing Databases

- Now you can put this in your repository.
- To import it you can:
  - Delete the current database
    - Maybe via the MYSQL command prompt?
  - Import the new one
    - Several ways. From the MySQL command line you can just do

```
use <dbname>;
source <filename.sql>;
```

    - Typically the expected root directory is wherever you installed MySQL's Server's Binaries.