CS3226

WEB PROGRAMMING AND APPLICATIONS

LECTURE 08 - SQL

DR STEVEN HALIM

STEVENHALIM@GMAIL.COM

For 'not newbies': Try this SQL quiz @ W3Schools first

OUTLINE

WHY DATABASE? (1)

So far, we have been augmenting the *stateless* HTTP with either HTML5 `localStorage` and/or PHP superglobals variable `$_SESSION` to make a dynamic web application that remembers user's interactions with the application

But how to *permanently* store the data that we have due to interaction with the users *across sessions* in *our server*?

WHY DATABASE? (2)

One clear solution: Database

We do not use simple file as our web application will likely be accessed *by many users*, **possibly at similar time**

We need the database system's Atomicity, Consistency, Isolation, Durability (ACID)* properties to handle concurrent access and yet still maintain data integrity

Note: The first half of this lecture is CS2102 quick revision using example that is relevant for the next security lecture

RELATIONAL DATABASE

- First published by Edgar Frank Codd in 1970
- A relational database consists of a collection of tables
- A table consists of rows and columns
- Each row represents a record
- Each column represents an attribute of the records contained in the table

EXAMPLE: USER ACCOUNT

Imagine we have a simple database with three tables: USER, ACCESS, and SCOREBOARD

1. USER has attributes: USERID, EMAIL, PASSWORD
2. ACCESS has attributes: USERID and ROLE
3. SCOREBOARD has attributes: USERID and SCORE

SOME CONTENT

```
Table USER
USERID   | EMAIL                      | PASSWORD
---------+----------------------------+-------------
steven   | stevenha@comp.nus.edu.sg | hashed1
student1 | axxxxxx@comp.nus.edu.sg  | hashed2
student2 | bxxxxxx@comp.nus.edu.sg  | hashed3
student3 | cxxxxxx@comp.nus.edu.sg  | hashed4
...
Table ACCESS                  Table SCOREBOARD
USERID   | ROLE               USERID   | SCORE
---------+--------            ---------+--------
steven   | 1                  student2 | 90
student1 | 2                  student3 | 70
student2 | 3                  ...
student3 | 3
...
```

UNSTRUCTURED DATA

Compare the previous tables with this unstructured one:

Steven has one student as his TA (student1) and a few other students. He has listed the score of his students in a table and he wants to let his TA to update the scores weekly. His students can only see the scores without ability to do update....

Same information, but it is very hard to perform *meaningful queries* on this data

ENTITIES AND RELATIONS

- An *entity* represents something real
- A *relation* represents a connection between entities
- The tables USER and SCOREBOARD may be regarded as *entities*
- Table ACCESS may be regarded as a *relationship* between a particular user in table USER and what he/she can do with table SCOREBOARD

SOME FORMALITIES

- A **domain** is a set of values
  The set of integers is a domain, so is the set of all strings
- The **cartesian product** of domains D_1 x D_2 x … x D_k is the set of all k-tuples (v_1, v_2,…, v_k)
  where v_1 is in D_1, v_2 is in D_2, …, and v_k is in D_k
  Example: If k = 2 and D_1 = {0, 1} and D_2 = {a, b, c} then $D_1 \times D_2$ is {(0, a), (0, b), (0, c), (1, a), (1, b), (1, c)}

SQL BASICS

STRUCTURED QUERY LANGUAGE (SQL)

Pronounciation: Either S Q L (one character at a time) or 'sequel', your choice

SQL is used on a Relational Database Management System (RDBMS) to create, search, and modify tables

MYSQL

In CS3226, we use MySQL as the default RDBMS

We will concentrate on important MySQL Create Read Update Delete (CRUD) operations

You are free to explore other SQL RDBMS like PostgreSQL or even NoSQL system like MongoDB, but they will not be officially supported

ACCESSING MYSQL AT CS3226-1 VM

1. Login to CS3226-1 VM using your CS3226-1 account
2. At command prompt (terminal), type in '`mysql -u your_username -h localhost -p`'
3. Type in `your_CS3226_password` (the one for Lab0)
4. (If this is your first login): At MySQL prompt, type '`SET PASSWORD FOR 'your_username'@'localhost' = PASSWORD('typeinanotherpwdhere');`'
5. At MySQL prompt, type in '`USE your_username;`'
   This is the only database that you can access other than the default read-only `information_schema` (can be checked using 'SHOW databases')

CREATE TABLE

The basic syntax is '`CREATE TABLE TABLE_NAME (list of columns and their definitions);`' (details)

Usually we use UPPERCASE for most SQL commands and table/attribute names

Example: Executing the following SQL commands causes the creation of the USER, ROLE, and SCOREBOARD database tables shown earlier (all initially empty)

```
CREATE TABLE USER (USERID VARCHAR(20) PRIMARY KEY, EMAIL VARCHAR(30),
CREATE TABLE ACCESS (USERID VARCHAR(20) PRIMARY KEY, ROLE INTEGER);
CREATE TABLE SCOREBOARD (USERID VARCHAR(20) PRIMARY KEY, SCORE INTEGE
```

SQL DATA TYPES

Some basic SQL data types that you may need in your database tables (details):

- INTEGER: signed fullword binary integer
- FLOAT: signed doubleword floating point number
- VARCHAR(n): varying length character string of maximum length n
- DATE: A date attribute in a DBMS-specific format

For basic web applications, these basic data types are usually enough

A NOTE ABOUT NULL

SQL allows the **NULL** value to appear in tuples (table rows) and it indicates a non-initialized attribute in a row

This can be disallowed by adding a **NOT NULL** constraint during table creation

```
CREATE TABLE USER (
  USERID VARCHAR(20) PRIMARY KEY,
  EMAIL VARCHAR(30) NOT NULL,
  PASSWORD VARCHAR(20) NOT NULL
);
```

If the USER table is declared this way, then when we add a row to the USER table, then EMAIL and PASSWORD must be specified (on top of USERID)

INSERT INTO TABLE

To do some work with our tables, we need to first populate it (using the `INSERT` command)

The syntax is '`INSERT INTO TABLE_NAME VALUES(list of values)`' (details)

Example:

```
INSERT INTO USER VALUES('steven', 'stevenha@comp.nus.edu.sg', 'hashed
INSERT INTO USER VALUES('student1', 'axxxxxx@comp.nus.edu.sg', 'hashe
INSERT INTO USER VALUES('student2', 'bxxxxxx@comp.nus.edu.sg', 'hashe
INSERT INTO USER VALUES('student3', 'cxxxxxx@comp.nus.edu.sg', 'hashe
INSERT INTO USER VALUES('student4', null, 'hashed5'); -- will not wo
```

EXECUTING AN SQL SOURCE SCRIPT

The SQL commands used so far have been recorded as populate.sql script

To ensure that we have the same database table contents as with the one shown in this slide, type in '`source populate.sql;`' at MySQL prompt

The SQL commands that will be used in the next few slides until before the PHP+MySQL part are recorded as examples.sql script

SELECT - BASIC

The result of a SQL search is a table (soon, we will learn how to format the result as an HTML table)

The most basic syntax is '`SELECT EXPRESSION FROM TABLE_NAME OPTIONAL_COMMANDS;`' (details)

```
SELECT * FROM ACCESS;
```

The result is *the whole* ACCESS table

```
+----------+-------+
| USERID   | ROLE  |
+----------+-------+
| steven   |     1 |
| student1 |     2 |
| student2 |     3 |
| student3 |     3 |
+----------+-------+
```

SELECT - WHERE (1)

Example on using the "WHERE" clause

```
SELECT * FROM ACCESS WHERE ROLE < 3;
```

The result is *part* of the ACCESS table

```
+----------+-------+
| USERID   | ROLE |
+----------+-------+
| steven   |    1 |
| student1 |    2 |
+----------+-------+
```

SELECT - WHERE (2)

Example on using the more complex "WHERE" clause

```
SELECT * FROM ACCESS WHERE ROLE = 1 OR ROLE >= 3;
```

The result is *part* of the ACCESS table

```
+----------+------+
| USERID   | ROLE |
+----------+------+
| steven   |    1 |
| student2 |    3 |
| student3 |    3 |
+----------+------+
```

SELECT - GET COLUMN(S)

Selecting particular columns

```
SELECT USERID FROM ACCESS WHERE ROLE < 3;
```

The result is *part* of the ACCESS table, *column USERID*

```
+----------+
| USERID   |
+----------+
| steven   |
| student1 |
+----------+
```

SELECT - ORDER BY

We can sort the output

```
SELECT * FROM ACCESS ORDER BY ROLE DESC;
```

The result is the ACCESS table, but in descending order of ROLE values

```
+----------+------+
| USERID   | ROLE |
+----------+------+
| student2 |    3 |
| student3 |    3 |
| student1 |    2 |
| steven   |    1 |
+----------+------+
```

SELECT - CARTESIAN PRODUCT

We can combine information from two tables

```
SELECT * FROM USER, ACCESS;
```

The result is the table USER x ACCESS with rows of the form (u, a) where u is a row in USER and a is a row in ACCESS; In total, we have 4*4 = 16 rows

```
+----------+--------------------------+----------+----------+------+
| USERID   | EMAIL                    | PASSWORD | USERID   | ROLE |
+----------+--------------------------+----------+----------+------+
| steven   | stevenha@comp.nus.edu.sg | hashed1  | steven   |    1 |
| student1 | axxxxxx@comp.nus.edu.sg  | hashed2  | steven   |    1 |
| student2 | bxxxxxx@comp.nus.edu.sg  | hashed3  | steven   |    1 |
| student3 | cxxxxxx@comp.nus.edu.sg  | hashed4  | steven   |    1 |
| steven   | stevenha@comp.nus.edu.sg | hashed1  | student1 |    2 |
... 10 other rows ...
| student3 | cxxxxxx@comp.nus.edu.sg  | hashed4  | student3 |    3 |
+----------+--------------------------+----------+----------+------+
```

SELECT - JOINS

We can match up the rows in multiple tables based on the same (primary key) value for columns, e.g. compare the previous cartesian product of USER x ACCESS with this one

```
SELECT U.EMAIL, A.ROLE FROM USER U, ACCESS A WHERE U.USERID=A.USERID;
```

Note: The tables have been aliased in the above query: USER as U, ACCESS as A with this result:

```
+--------------------------+------+
| EMAIL                    | ROLE |
+--------------------------+------+
| stevenha@comp.nus.edu.sg |    1 |
| axxxxxx@comp.nus.edu.sg  |    2 |
| bxxxxxx@comp.nus.edu.sg  |    3 |
| cxxxxxx@comp.nus.edu.sg  |    3 |
+--------------------------+------+
```

OTHER COMMON SQL QUERIES

Update row(s) in a table USER

```
UPDATE USER SET email='stevenhalim@gmail.com' WHERE USERID='steven';
SELECT * FROM USER; -- see that steven's email has been updated
```

Delete row(s) from a table SUPPLIER

```
DELETE FROM ACCESS WHERE USERID = 'student3';
SELECT * FROM ACCESS; -- see that 'student3' is no longer listed
```

Drop the entire table ACCESS

```
DROP TABLE ACCESS;
SELECT * FROM ACCESS; -- error, that table no longer exist
```

NOTES

A relational database is a powerful tool and we have just scratched the surface as we do not cover:

- Transaction processing
- Concurrent access
- Aggregate queries
- Stored procedures (PL/SQL, embedded Java)
- Integrity constraints
- Design
- Indexes
- Fault tolerence
- Online backups
- Database distribution, etc...

Take CS2102 (if you have not done so) to learn more about these...

PHP + MYSQL

LINKING THE TWO (1)

In the first half of this lecture, we digressed to database (CS2102) and used direct mysql command line interface

Now, we will write PHP scripts to access our MySQL database so that we can Create/Read/Update/Delete our database *from our web application*

LINKING THE TWO (2)

Basically, the PHP commands that you will need to know to access MySQL database are:

```
$db = new mysqli($db_host, $db_uid, $db_pwd, $db_name); // open conne
$res = $db->query($query); // THAT SQL query that we covered so far
$sanitizedinput = $db->escape_string($inputfromuser); // increase sec
$db->close(); // close connection
```

More details are given in the next few slides

MAKING DATABASE CONNECTION (1)

Create a PHP file: `config.php` and save it *outside* the `public_html` folder, e.g. `../public_html`

```
<?php // config.php basically contains these 4 constants
define("db_host", "localhost"); // a constant doesn't need a $
define("db_uid", "your_cs3226_username"); // change this to yours
define("db_pwd", "your_cs3226_password"); // change this to yours
define("db_name", "your_cs3226_username"); // default for this class
?>
```

MAKING DATABASE CONNECTION (2)

To connect with our database, we use `connect.php`:

```
<?php // connect.php basically contains these commands
require_once '../config.php'; // your PHP script(s) can access this f
$db = new mysqli(db_host, db_uid, db_pwd, db_name); // it is built-ir
if ($db->connect_errno) // are we connected properly?
  exit("Failed to connect to MySQL, exiting this script");
?>
```

MAKING SQL QUERIES

After we are connected, we can perform MySQL queries from our web application :), e.g. displaying the table in our database as HTML table, try db.php

```php
<?php require_once 'connect.php';
function DisplayTableInHTML($table_name) {
  global $db; // refer to the global variable 'db'
  $query = "SELECT * FROM " . $table_name;
  $res = $db->query($query); // yes, just like this
  if (!$res) exit("There is a MySQL error, exiting this script");
  echo "<p>Table " . $table_name . "<br></p>"; // dynamic HTML table
  echo "<table border=\"1px\" style=\"border-collapse: collapse\">";
  while ($r = mysqli_fetch_row($res)) { // important command
    echo "<tr><td>" . $r[0] . "</td>"; // echo first column
    for ($i = 1; $i < count($r); $i++) echo "<td>" . $r[$i] . "</td>"
    echo "</tr>";
  }
  echo "</table>";
}
DisplayTableInHTML("USER"); $db->close(); ?>
```

UPDATING OUR DATABASE

Obviously, we can update our database using our PHP script

However, to prevent security issues like SQL injection, I will limit what you can do to changing email of table USER only

Example update_db.php?userid=steven&email=a@b.com and then see db.php again

```
<?php require_once 'connect.php';
$uid = $_REQUEST['userid']; // get userid (dangerous)
$eml = $_REQUEST['email']; // get the new email (dangerous)
echo "userid = <font color="red">" . $uid . "</font><br>\n";
$q = "UPDATE USER SET EMAIL='" . $eml . "' WHERE USERID='" . $uid . '
echo "SQL query = <font color="red">" . $q . "</font><br>\n";
$res = $db->query($query); // you can form SQL query from URL paramet
if (!$res) exit("MySQL reports " . $db->error);
else echo $db->affected_rows . " rows are updated, db.php.<br>";
?>
```

ALWAYS SANITIZE THE INPUT! (1)

We can still inject something update_db.php?userid=steven&email=a@b.com', PASSWORD='somethingelse

Check db.php again

```
+----------+---------------------------------+-----------------+
| USERID   | EMAIL                           | PASSWORD        |
+----------+---------------------------------+-----------------+
| steven   | a@b.com                         | somethingelse   |
| student1 | axxxxxx@comp.nus.edu.sg         | hashed2         |
| student2 | bxxxxxx@comp.nus.edu.sg         | hashed3         |
| student3 | cxxxxxx@comp.nus.edu.sg         | hashed4         |
+----------+---------------------------------+-----------------+
```

ALWAYS SANITIZE THE INPUT! (2)

But if we sanitize the inputs like this:

```php
<?php require_once 'connect.php';
$uid = $db->escape_string($_REQUEST['userid']); // better
$eml = $db->escape_string($_REQUEST['email']); // better
// same as before
?>
```

Now you should not be able to do that SQL injection anymore (details in security lecture)

NOTES (1)

This entire lecture note is just one simple working example on how to use database (in this case, MySQL) and manipulate it via web-based interface (in this case, via PHP script called from a web browser)

The process of updating/showing/modifying database in a real web application is much more sophisticated and fancier than this example

NOTES (2)

There are many more SQL commands (take CS2102/other database module in SoC if you have not done so or self-study) and PHP commands involving MySQL (self-study)

Let's get our hands dirty with Lab5

THE END

Try this SQL quiz @ W3Schools if you have not done so