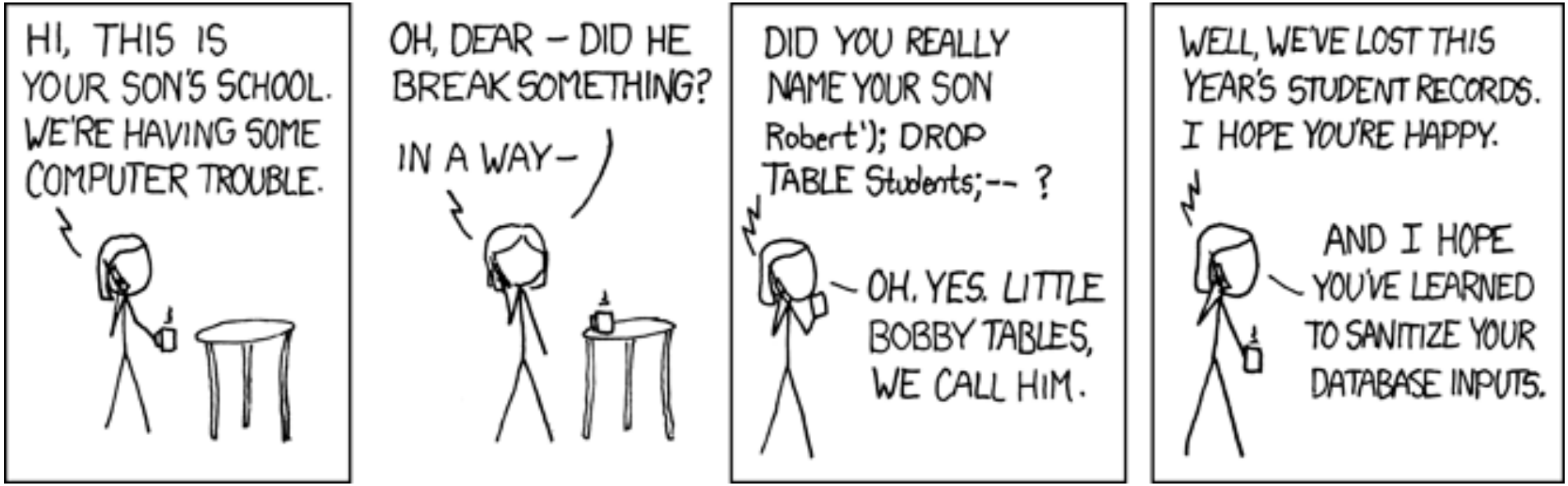
CS 55:
Security and Privacy

SQL injection attacks



https://xkcd.com/237/

Demo on VM

Start Firefox browser

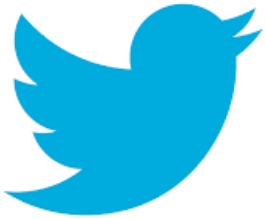
Go to: http://www.seedlabsqlinjection.com

- Log in as Alice
  - Username: alice
  - Password: seedalice
  - See Alice's info
- Go to Edit Profile (at top)
  - Change details

- Log in as admin
  - Username: admin
  - Password: seedadmin
  - See all employee data

You use databases every day, but may not think them about very much

The New York Times



facebook

Virtually all non-trivial applications have a database component

Databases are a set of programs used to Create, Read, Update, or Delete (CRUD) data through operations called queries

Queries typically use SQL to carry out queries

What characteristics would you like in a database?

Database skills are in high demand

TOP TECH SKILLS

2019 Rank	Occupation	Change in Rank from 2018
1	SQL	—
2	Java	—
3	JavaScript	▲ 1
4	Project Management	▼ 1
5	Python	▲ 1
6	Linux	▼ 1
7	Oracle	—
8	Microsoft C#	—
9	Scrum	▲ 1
10	Quality Assurance and Control	▼ 1

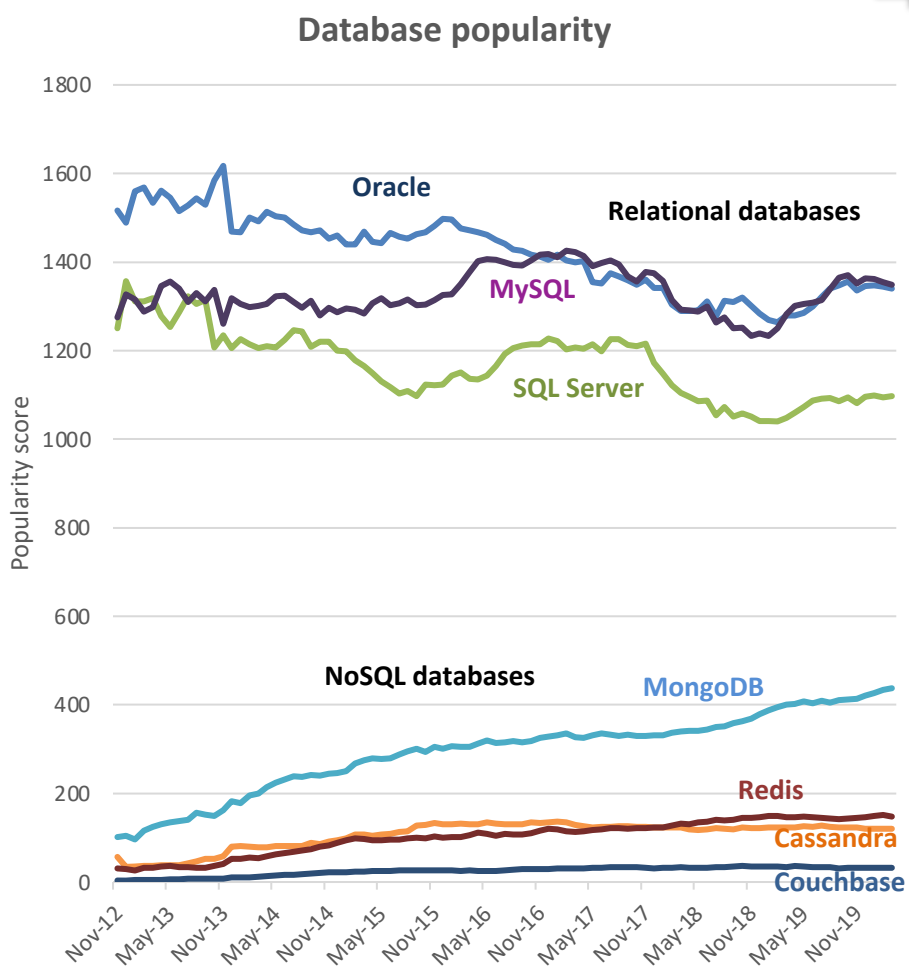
Job placement firm Dice analyzed 6 million job postings for most frequently sought tech skills in 2020

Employers are looking to hire people with database skills

https://techhub.dice.com/Dice-2020-Tech-Job-Report.html

There are number of popular DBMS's in use today; we will use MySQL

Popular Database Management Systems



https://db-engines.com/en/ranking_trend

Relational Our focus

- *MySQL/MariaDB (open source, but owned by Oracle, MariaDB is fork)*
- Oracle (king of the hill, but expensive)
- Microsoft SQL Server (also Access, easy to use compared with Oracle)

NoSQL

- *Mongo (most popular NoSQL, has security concerns?)*
- Redis (in-memory data structure store, used as a database, cache and message broker)
- Cassandra (hybrid key-value and column-oriented DB)
- Couchbase (key/value store)

Agenda

1. SQL tutorial
2. SQL injection attacks
3. Countermeasures

Big picture of relational database design

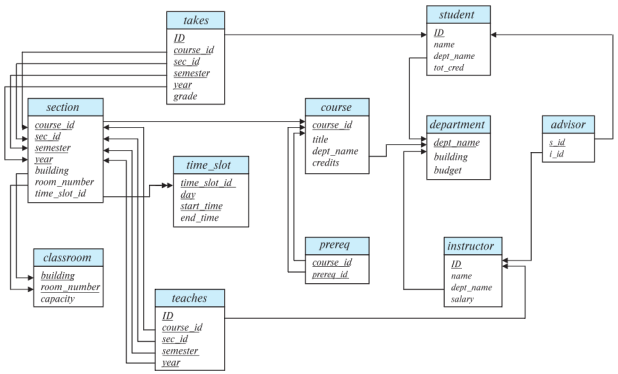
Relational Database Management System (RDBMS)

- Normally represented graphically as a cylinder
- Holds data in relations (tables)

ID	*name*	*dept_name*	*salary*
22222	Einstein	Physics	95000
12121	Wu	Finance	90000
32343	El Said	History	60000
45565	Katz	Comp. Sci.	75000
98345	Kim	Elec. Eng.	80000
76766	Crick	Biology	72000
10101	Srinivasan	Comp. Sci.	65000
58583	Califieri	History	62000
83821	Brandt	Comp. Sci.	92000

Relations (tables)

- Each relation holds data about people, places, things or events (nouns)
- Tables consist of rows and columns
- Each row (tuple or relation instance) represents one person, place, thing, or event
- Each column (field or attribute) represents one aspect of a person, place, thing, or event (e.g., last name)
- A column (FK) can refer to a column (PK) in another table, creating a relationship between tables



Database schema

- Logical collection of tables and relationships
- Minimizes storing multiple copies of data
- Look up additional data in another table if needed using key

Silberschatz, Abraham, Henry F. Korth, and Shashank Sudarshan. *Database system concepts*. 7th Edition. New York: McGraw-Hill, 2019.

Relational database systems store data in relations (aka tables) made up of attributes

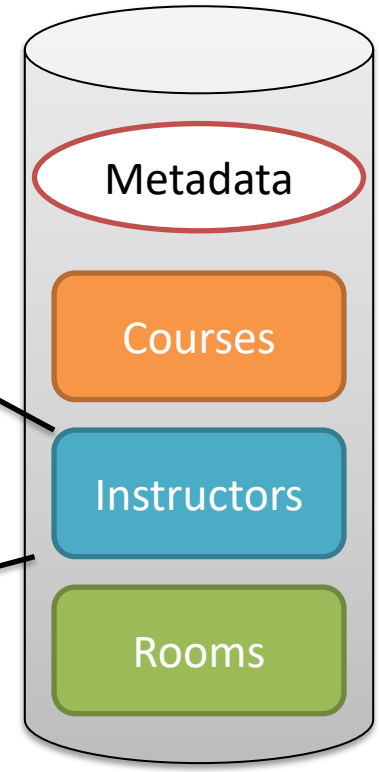
Relational database table

attributes

Instructors relation

Relation instances

ID	*name*	*dept_name*	*salary*
22222	Einstein	Physics	95000
12121	Wu	Finance	90000
32343	El Said	History	60000
45565	Katz	Comp. Sci.	75000
98345	Kim	Elec. Eng.	80000
76766	Crick	Biology	72000
10101	Srinivasan	Comp. Sci.	65000
58583	Califieri	History	62000
83821	Brandt	Comp. Sci.	92000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
76543	Singh	Finance	80000



College database schema

Data in a relational database

- Data stored in **relations** (tables)
- Relations are made up of **relation instances** (rows or tuples)
- Relation instances are made up of a fixed number of **attributes** (fields or columns) of fixed type
- Related relations are contained in a **schema** (aka database)
- Database may store multiple schemas

Silberschatz, Abraham, Henry F. Korth, and Shashank Sudarshan. *Database system concepts*. 7th Edition. New York: McGraw-Hill, 2019.

We will use the popular MySQL relational database management system (RDBMS)

Log in to MySQL

```
$ mysql –uroot –pseedubuntu
Welcome to the MySQL monitor.
mysql>
```

Most relational databases use Structured Query Language (aka S-Q-L, aka Sequel) to issue commands

If you use MySQL often, consider a tool such as MySQL Workbench

See databases “SHOW DATABASES;” create a new with “CREATE DATABASE <name>;”

See what databases already exist

```
mysql> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| Users              |
| dbtest             |
| elgg_csrf          |
| elgg_xss           |
| mysql              |
| performance_schema |
| phpmyadmin         |
| sys                |
+--------------------+
9 rows in set (0.00 sec)9 rows in set (0.00 sec)
```

Note: SQL commands end with semicolon

By convention we capitalize SQL keywords, but it is not required

Create new database

```
mysql> CREATE DATABASE cs55;
```

Data is stored in tables; each table represents a collection of related entities

Tell MySQL to use our new database and create an Employees table

```
mysql> USE cs55;
Database changed

mysql> CREATE TABLE `Employees` (
  `ID` int(11) NOT NULL AUTO_INCREMENT,
  `Name` varchar(30) NOT NULL,
  `EID` varchar(7) NOT NULL,
  `Password` varchar(60) DEFAULT NULL,
  `Salary` int(11) DEFAULT NULL,
  `SSN` varchar(11) DEFAULT NULL,
  PRIMARY KEY (`ID`)
);
Query OK, 0 row affected (0.002 sec)
```

Create table named Employees

Add attributes (fields) about employees, give name and data type

Table will have one row (record) per employee

Auto_increment means give each entry a number one greater than the previous entry, used for Key attribute

NOT NULL means the attribute must have a value other than NULL

DEFAULT NULL means assign NULL unless given another value

Database will look up employees by their primary key (ID here)

Use “describe <table>” see the structure of a database table

See the table’s structure with describe command

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

Show tables lists all tables in a database schema

See the tables in a database schema

```
mysql> show tables;
+----------------+
| Tables_in_cs55 |
+----------------+
| Employees      |
| Restaurants    |
| Users          |
+----------------+
3 rows in set (0.00 sec)
```

Structured Query Language (SQL) performs CRUD operations on data

CRUD

Create

```
INSERT INTO table (field1, field2, …)
VALUES (val1, val2, …)
```

Read

```
SELECT *|field list
FROM table
WHERE conditions
```

Update

```
UPDATE table
SET field1=val1, field2=val2
WHERE conditions
```

Delete

```
DELETE FROM table
WHERE conditions
```

Use INSERT to add rows to the table; use SELECT to see table rows

Add employee named Alice into table with INSERT command

```
mysql> INSERT INTO Employees (Name, EID, Password, Salary, SSN)
VALUES ('Alice', 'EID5001', 'passwd123',80000, '111-11-1111');
Query OK, 1 row affected (0.01 sec)
```

See table rows with SELECT command

SELECT * means return all attributes

```
mysql> SELECT * FROM Employees;
+----+------------+---------+-----------+--------+-------------+
| ID | Name       | EID     | Password  | Salary | SSN         |
+----+------------+---------+-----------+--------+-------------+
|  1 | Alice      | EID5001 | passwd123 |  80000 | 111-11-1111 |
+----+------------+---------+-----------+--------+-------------+
1 row in set (0.00 sec)
```

Can specify what attributes to return

```
mysql> SELECT Name, Salary FROM Employees;
+------------+--------+
| Name       | Salary |
+------------+--------+
| Alice      |  80000 |
+------------+--------+
```

SQL commands can span multiple lines, end with semicolon

SELECT Name, Salary
FROM Employees;

WHERE clauses allow us to limit which rows are returned

WHERE clause

WHERE clause is used to set conditions for several types of SQL statements including SELECT, UPDATE, or DELETE

```
mysql>  SQL command
        WHERE predicate;
```

Affects rows for which the predicate in the WHERE clause is TRUE

The predicate is a logical expression; multiple predicates can be combined using keywords AND and OR

WHERE clauses can use AND and OR

WHERE clause

Added some more employees before this command

```
mysql> SELECT * FROM Employees;
+----+---------+---------+-----------+--------+-------------+
| ID | Name    | EID     | Password  | Salary | SSN         |
+----+---------+---------+-----------+--------+-------------+
|  1 | Alice   | EID5001 | passwd123 |  80000 | 111-11-1111 |
|  2 | Bob     | EID5002 | passwd123 |  80000 | 222-22-2222 |
|  3 | Charlie | EID5003 | passwd123 |  85000 | 333-33-3333 |
|  4 | Denise  | EID5004 | passwd123 |  90000 | 444-44-4444 |
+----+---------+---------+-----------+--------+-------------+

mysql> SELECT * FROM Employees WHERE Name = 'Alice' OR EID='EID5003';
+----+---------+---------+-----------+--------+-------------+
| ID | Name    | EID     | Password  | Salary | SSN         |
+----+---------+---------+-----------+--------+-------------+
|  1 | Alice   | EID5001 | passwd123 |  80000 | 111-11-1111 |
|  3 | Charlie | EID5003 | passwd123 |  85000 | 333-33-3333 |
+----+---------+---------+-----------+--------+-------------+
2 rows in set (0.00 sec)
```

If WHERE clause always evaluates to true, then all rows are affected

WHERE clause

1 always equals 1, so true for all rows
Returns all rows

```
mysql> SELECT * FROM Employees WHERE 1=1;
+----+---------+---------+-----------+--------+-------------+
| ID | Name    | EID     | Password  | Salary | SSN         |
+----+---------+---------+-----------+--------+-------------+
|  1 | Alice   | EID5001 | passwd123 |  80000 | 111-11-1111 |
|  2 | Bob     | EID5002 | passwd123 |  80000 | 222-22-2222 |
|  3 | Charlie | EID5003 | passwd123 |  85000 | 333-33-3333 |
|  4 | Denise  | EID5004 | passwd123 |  90000 | 444-44-4444 |
+----+---------+---------+-----------+--------+-------------+
```

Seems like an odd thing to point out, stand by…

UPDATE modified records, normally using WHERE clause

UPDATE command

Give Alice a raise

```
mysql> UPDATE Employees SET Salary=82000 WHERE ID=1;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

Alice now has a new salary

```
mysql> SELECT * FROM Employees;
+----+---------+---------+-----------+--------+-------------+
| ID | Name    | EID     | Password  | Salary | SSN         |
+----+---------+---------+-----------+--------+-------------+
|  1 | Alice   | EID5001 | passwd123 |  82000 | 111-11-1111 |
|  2 | Bob     | EID5002 | passwd123 |  80000 | 222-22-2222 |
|  3 | Charlie | EID5003 | passwd123 |  85000 | 333-33-3333 |
|  4 | Denise  | EID5004 | passwd123 |  90000 | 444-44-4444 |
+----+---------+---------+-----------+--------+-------------+
4 rows in set (0.00 sec)
```

SQL commands can have comments, MySQL supports three different types

```
mysql> SELECT * FROM Employees; # comment to end of line
mysql> SELECT * FROM Employees;  -- comment to end of line
mysql> SELECT * FROM /* In-line comment */ Employees;
```

Agenda

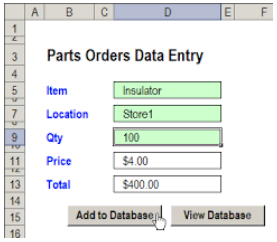
1. SQL tutorial
2. SQL injection attacks
3. Countermeasures

Today applications (and users) normally access a database through an API

Three-tiered architecture

Smart phone apps

Web browser



"Thick client" apps

Network

API

Database Management System (DBMS)

Metadata

Course

Instructor

Room

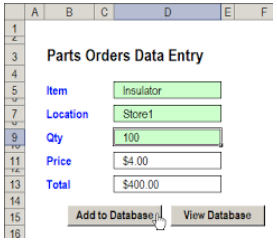
College database schema

Today applications (and users) normally access a database through an API

Three-tiered architecture

Smart phone apps

Web browser

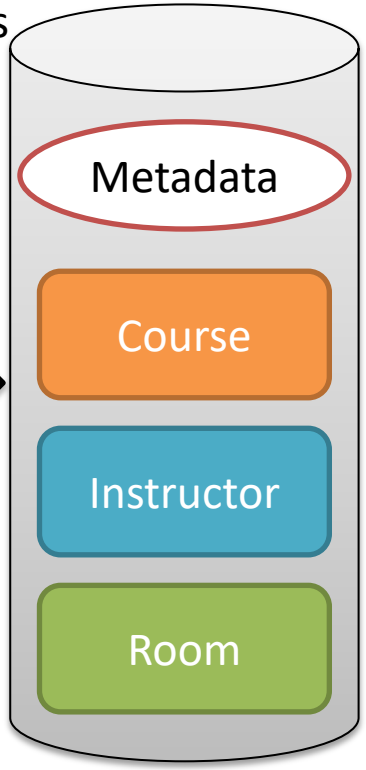


"Thick client" apps

Network

API

Database Management System (DBMS)



College database schema

Advantages

1. Allows data to be shared between multiple applications
2. Creates a single repository of knowledge
3. Manages security

Tier 1: DBMS

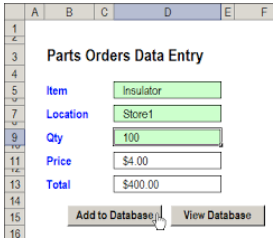
- Manages database structure
- Controls access to database
- Allows data to be shared

Today applications (and users) normally access a database through an API

Three-tiered architecture

Smart phone apps

Web browser



"Thick client" apps

Network

Advantages

1. Abstracts data access
2. Data storage can be changed without changing all user applications

API

Tier 2: API

- Provides access to database via web services
- May also be web server for web pages

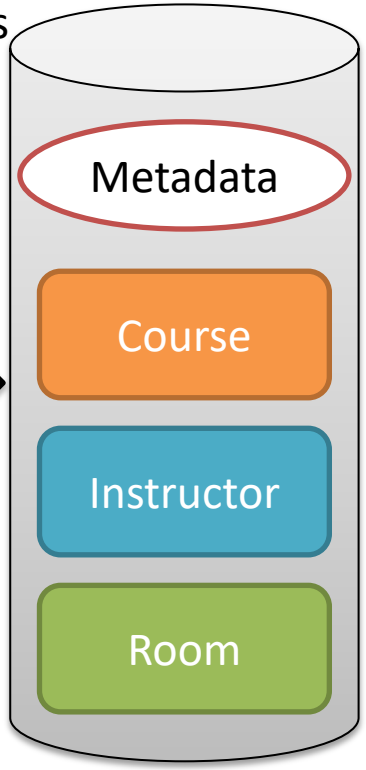
Advantages

1. Allows data to be shared between multiple applications
2. Creates a single repository of knowledge
3. Manages security

Database Management System (DBMS)

Tier 1: DBMS

- Manages database structure
- Controls access to database
- Allows data to be shared



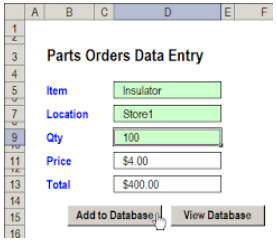
College database schema

Today applications (and users) normally access a database through an API

Three-tiered architecture

Smart phone apps

Web browser



"Thick client" apps

Tier 3: Applications

Network

Advantages

1. Abstracts data access
2. Data storage can be changed without changing all user applications

API

Tier 2: API

- Provides access to database via web services
- May also be web server for web pages

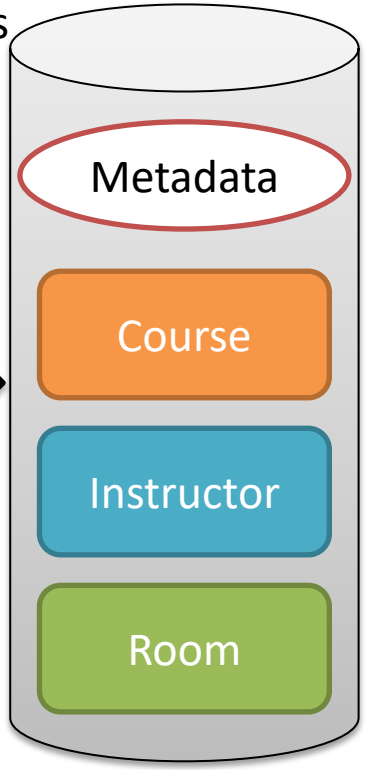
Advantages

1. Allows data to be shared between multiple applications
2. Creates a single repository of knowledge
3. Manages security

Database Management System (DBMS)

Tier 1: DBMS

- Manages database structure
- Controls access to database
- Allows data to be shared



College database schema

Users do not typically issue their own SQL commands

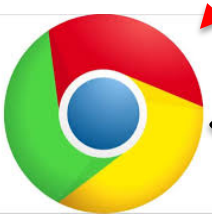
Three-tiered architecture

Users interact with applications Applications call API (e.g., do not issue SQL commands)

API interacts with DBMS (e.g., API issues SQL commands)

DBMS runs query returns results

Smart phone apps

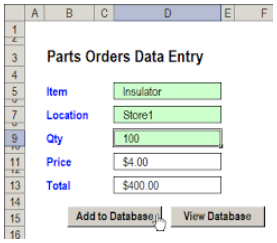


Web browser

Network

API

Database Management System (DBMS)



"Thick client" apps

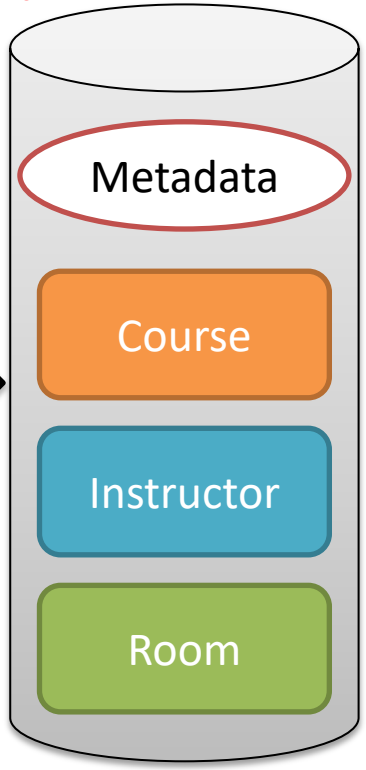
Tier 3: Applications

Tier 2: API

- Provides access to database via web services
- May also be web server for web pages

Tier 1: DBMS

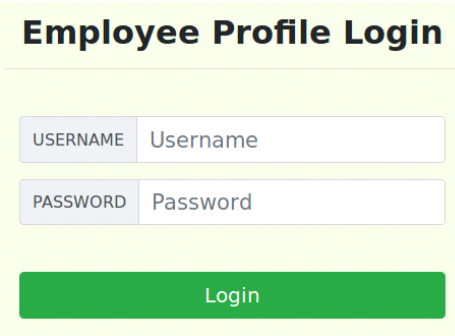
- Manages database structure
- Controls access to database
- Allows data to be shared



College database schema

Most applications will need to get a user's input at some point, often from a browser

Login screen where the user enters their username and password

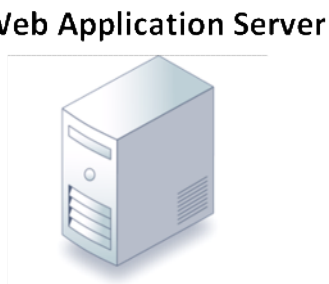


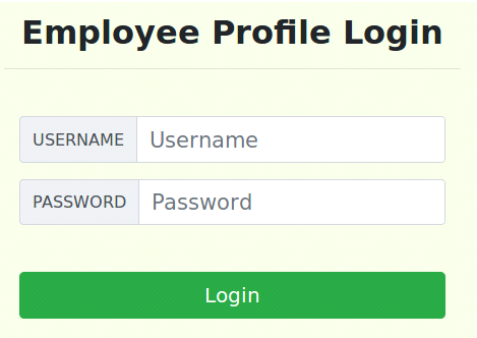

```
<form action="getdata.php" method="get">
Username: <input type="text" name="username"><br>
Password: <input type="text" name="password"><br>
          <input type="submit" value="Login>
</form>
```


Du, Wenliang. *Computer & Internet Security: A Hands-on Approach*. 2019.

User input passed to web service API on the server after the Login button clicked

www.example.com/getdata.php?Username=Alice&Password=seedalice






```
<?php
  $name = $_GET["Username"];
  $pwd = $_GET["Password"];
  $sql = "SELECT id, name, salary
            FROM credential
            WHERE name='$name' AND
            Password='$pwd'";
  $result = $conn->query($sql);
?>
```

```
<form action="getdata.php" method="get">
Username: <input type="text" name="username"><br>
Password: <input type="text" name="password"><br>
          <input type="submit" value="Login>
</form>
```

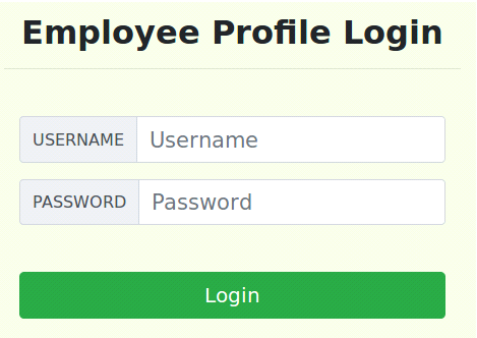
Clicking submit button calls PHP file on server named getdata.php and passes Username and Password parameters in query string

getdata.php issues SQL command to get data from the database using parameters passed in query string

Du, Wenliang. *Computer & Internet Security: A Hands-on Approach*. 2019.

The database fetches and returns the data requested by the user

www.example.com/getdata.php?Username=Alice&Password=seedalice




```
<?php
  $name = $_GET["Username"];
  $pwd = $_GET["Password"];
  $sql = "SELECT id, name, salary
          FROM credential
          WHERE name='$name' AND
          Password='$pwd'";
  $result = $conn->query($sql);
?>
```

```
SELECT id, name, salary
FROM credential
WHERE name='Alice' AND
Password='seedalice'
```

Database is queried and returns requested data

```
<form action="getdata.php" method="get">
Username: <input type="text" name="username"><br>
Password: <input type="text" name="password"><br>
          <input type="submit" value="Login>
</form>
```

Key point: the user never issues a SQL command directly to the database, the API does

Du, Wenliang. *Computer & Internet Security: A Hands-on Approach*. 2019.

PHP can connect to a MySQL database and parse user parameters passed by GET

Server side

```
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;
}
```

Server side: create a connection to the database

```
   $eid = $_GET['EID'];
   $pwd = $_GET['Password'];
   echo "EID: $eid --- Password: $pwd\n";
```

$_GET['param name') returns value in query string

Use $_POST['param name'] if HTML POST command issued

Du, Wenliang. *Computer & Internet Security: A Hands-on Approach*. 2019.

Using user's input in SQL command creates an attack vector

Construct SQL command from user input, then send command to database for execution

Get user's input from HTML GET

```
/* getdata.php */
<?php
   $eid = $_GET['EID'];
   $pwd = $_GET['Password'];

   $conn = new mysqli("localhost", "root", "seedubuntu", "dbtest");
   $sql = "SELECT Name, Salary, SSN
           FROM employee
           WHERE eid= '$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();
?>
```

Constructing SQL statement

Run command

Put user's input into SQL command

Print results

Key point: what the user types becomes part of the database query

What could do wrong?

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


Du, Wenliang. *Computer & Internet Security: A Hands-on Approach*. 2019.

Do not trust user input

A malicious user could bypass user/password checks

The intention of the web app developer by the following is for the user to provide some data for the blank areas.

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

Assume that a user types "EID5002'#" in the eid entry. The SQL statement will become the following

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

Everything after the # is treated as a comment, so the command is

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

Get information without knowing the password because password commented out!

Du, Wenliang. *Computer & Internet Security: A Hands-on Approach*. 2019.

A malicious user could bypass user/password checks

- Let's see if a user can get all the records from the database assuming we don't know all the EID's in the database.
- We need to create a predicate for WHERE clause so that it is true for all records.

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

User types a' or 1=1 #

Now all employees are listed because 1=1 is always true!

Du, Wenliang. *Computer & Internet Security: A Hands-on Approach*. 2019.

curl can launch attacks from the command line

Using cURL, we can send out a form from a command-line, instead of from a web page

```
$ curl 'www.SeedLabSQLInjection.com/unsafe_home.php?username=alice&Password=seedalice'
```

Special characters need to be UTF-8 encoded for curl to work:

Character	UTF-8
'	%27
#	%23
space	%20

Sometimes an adversary can change the contents of the database

If the statement is UPDATE or INSERT, we will have chance to change the database

Password change form asks for EID, old password, new password

When Submit button clicked, an HTTP POST sent to server-side script changepasswd.php, which uses an UPDATE statement to change the user's password

Note: adversary usually doesn't know what SQL looks like on server, has to guess

EID	EID50000
Old Password	paswd123
New Password	paswd456

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

    $conn = new mysqli("localhost:,"root","pwd","db");
    $sql = "UPDATE Employees
            SET password='$newpwd'
            WHERE eid='$eid' and password='$oldpwd'";
    $result = $conn->query($sql);
    $conn->close();
?>
```

Get user's input from HTTP POST

Use user's input in SQL UPDATE command

Du, Wenliang. *Computer & Internet Security: A Hands-on Approach*. 2019.

Alice wants to give herself a raise

Password change web form

EID	EID50000
Old Password	paswd123
New Password	paswd456

SQL command

```
UPDATE Employees
SET password='$newpwd'
WHERE eid='$eid' AND password='$oldpwd'
```

SQL statement sets two attributes: password and salary

```
UPDATE Employees
SET password='paswd456'
WHERE eid='EID5000' AND password='paswd123'
```


Du, Wenliang. *Computer & Internet Security: A Hands-on Approach*. 2019.

Alice wants to give herself a raise

Password change web form

EID	EID50000
Old Password	paswd123
New Password	paswd456', salary=100000 #

SQL command

```
UPDATE Employees
SET password='$newpwd'
WHERE eid='$eid' AND password='$oldpwd'
```

SQL statement sets two attributes: password and salary

```
UPDATE Employees
SET password='paswd456', salary=100000 #'
WHERE eid='EID5000' AND password='paswd123'
```


Du, Wenliang. *Computer & Internet Security: A Hands-on Approach*. 2019.

Alice doesn't like Bob…

Password change web form

EID	EID50001' #
Old Password	anything
New Password	paswd456', salary=0 #

SQL command

```
UPDATE Employees
SET password='$newpwd'
WHERE eid='$eid' AND password='$oldpwd'
```

SQL statement changes Bob's password and sets salary to zero

```
UPDATE Employees
SET password='paswd456', salary=0 #'
WHERE eid='EID5001' # AND password='anything'
```

Assume Alice knows Bob's EmployeeID but does not know Bob's password

Du, Wenliang. *Computer & Internet Security: A Hands-on Approach*. 2019.

Sometimes an adversary can execute multiple commands

Login web form

EID	a'; DROP DATABASE db; #
Password	anything

SQL command

```
SELECT Name, Salary, SSN
FROM Employees
WHERE eid='$eid'
```

SQL statement becomes

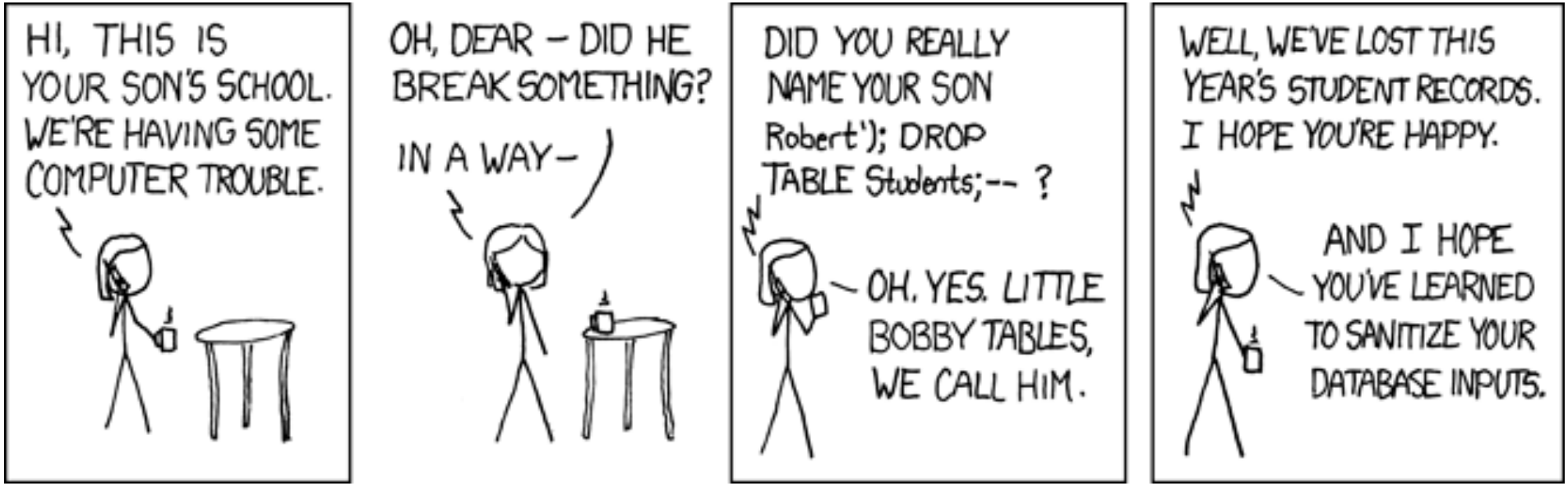
```
SELECT Name, Salary, SSN
FROM Employees
WHERE eid='a'; DROP DATABASE db; #'
```

Semicolon ends statements then second DROP DATABASE command follows

Note: does not work against MySQL because mysqli does not allow multiple queries

Du, Wenliang. *Computer & Internet Security: A Hands-on Approach*. 2019.

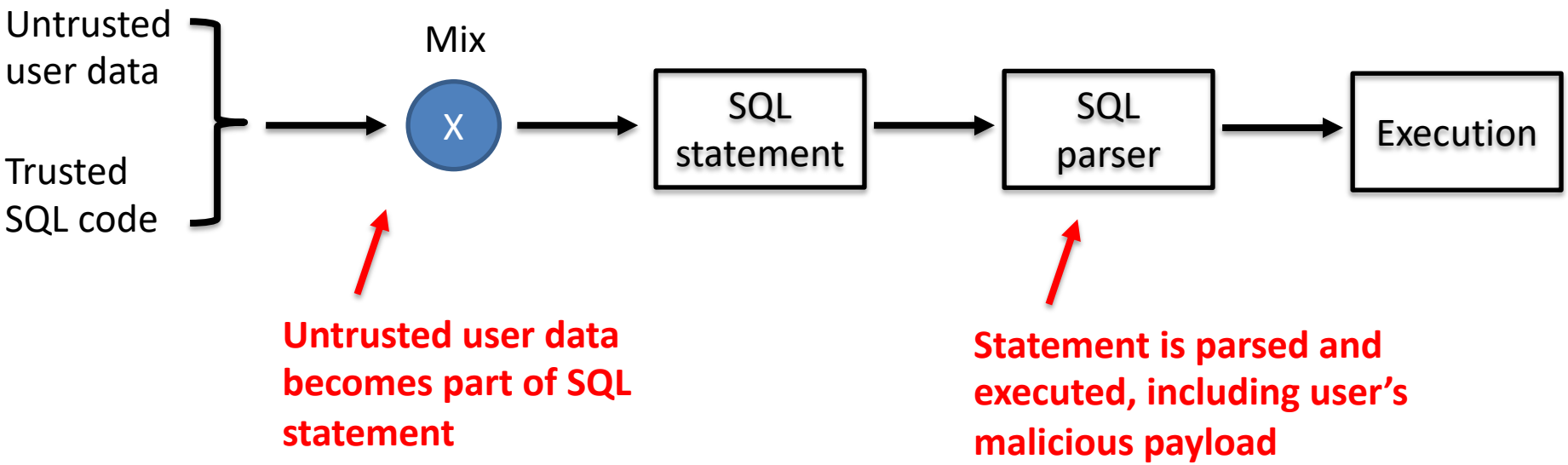
Now we know why this is funny



https://xkcd.com/237/

The fundamental problem is mixing untrustworthy user data and code

Simplified SQL execution flow



The problem of mixing user data and code is not limited to just SQL!

Du, Wenliang. *Computer & Internet Security: A Hands-on Approach*. 2019.

Agenda

1. SQL tutorial
2. SQL injection attacks
3. Countermeasures

Approach 1: filter out dangerous data that might be interpreted as code

Encode special characters tells parser to treat the encoded character as data and not as code

```
Before encoding:   aaa' OR 1=1 #
After encoding:    aaa\' OR 1=1 #
```

Use MySQL real_escape_string to encode user data

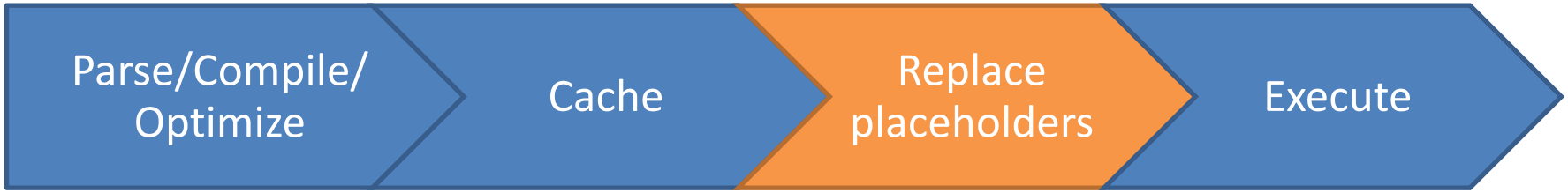
```
/* getdata_encoding.php */
<?php
   $conn = new mysqli("localhost", "root", "seedubuntu", "dbtest");
   $eid = $mysqli->real_escape_string($_GET['EID']);
   $pwd = $mysqli->real_escape_string($_GET['Password'];
   $sql = "SELECT Name, Salary, SSN
           FROM employee
           WHERE eid= '$eid' and password='$pwd'";
?>
```

Not recommended
Still mixes code and data

Approach 2: use prepared statements to separate code and data

High-level overview of SQL execution process

UPDATE Users SET UserName = ? AND Password = ?



Parse
- Check syntax
- Check table and columns exist

Compile
- Convert query to machine code

Optimize
- Choose optimal execution plan

Cache
- Store optimized query plan in cache
- If command submitted again, skip prior steps (already done)

Replace placeholders
- Prepared statement are not complete statements
- Have placeholders for some values
- But, format of command is set now
- Placeholders filled with literal values
- Place holder data doesn't change command format

Execute
- Query is executed
- Data is returned
- Malicious command are stored in table as text, not executed

Adapted from: http://javabypatel.blogspot.com/2015/09/how-prepared-statement-in-java-prevents-sql-injection.html

Approach 2: use prepared statements to separate code and data

```
$conn = new mysqli("localhost", "root", "seedubuntu", "dbtest");
$sql = "SELECT Name, Salary, SSN
        FROM employee
        WHERE eid= '$eid' and password='$pwd'";
$result = $conn->query($sql);
```

Original approach mixes code and user data

Instead, separate code and data with prepared statement

```
$conn = new mysqli("localhost", "root", "seedubuntu", "dbtest");
$sql = "SELECT Name, Salary, SSN
        FROM employee
        WHERE eid= ? and password=?";

if ($stmt = $conn->prepare($sql)) {
   $stmt->bind_param("ss", $eid, $pwd);
   $stmt->execute();

   $stmt->bind_result($name, $salary, $ssn);
   while ($stmt->fetch()) {
      printf ("%s %s %s\n", $name, $salary, $ssn);
   }
}
```

Question marks are placeholders for user data

Fill placeholders after statement is compiled

Added benefit: possible improved query performance!

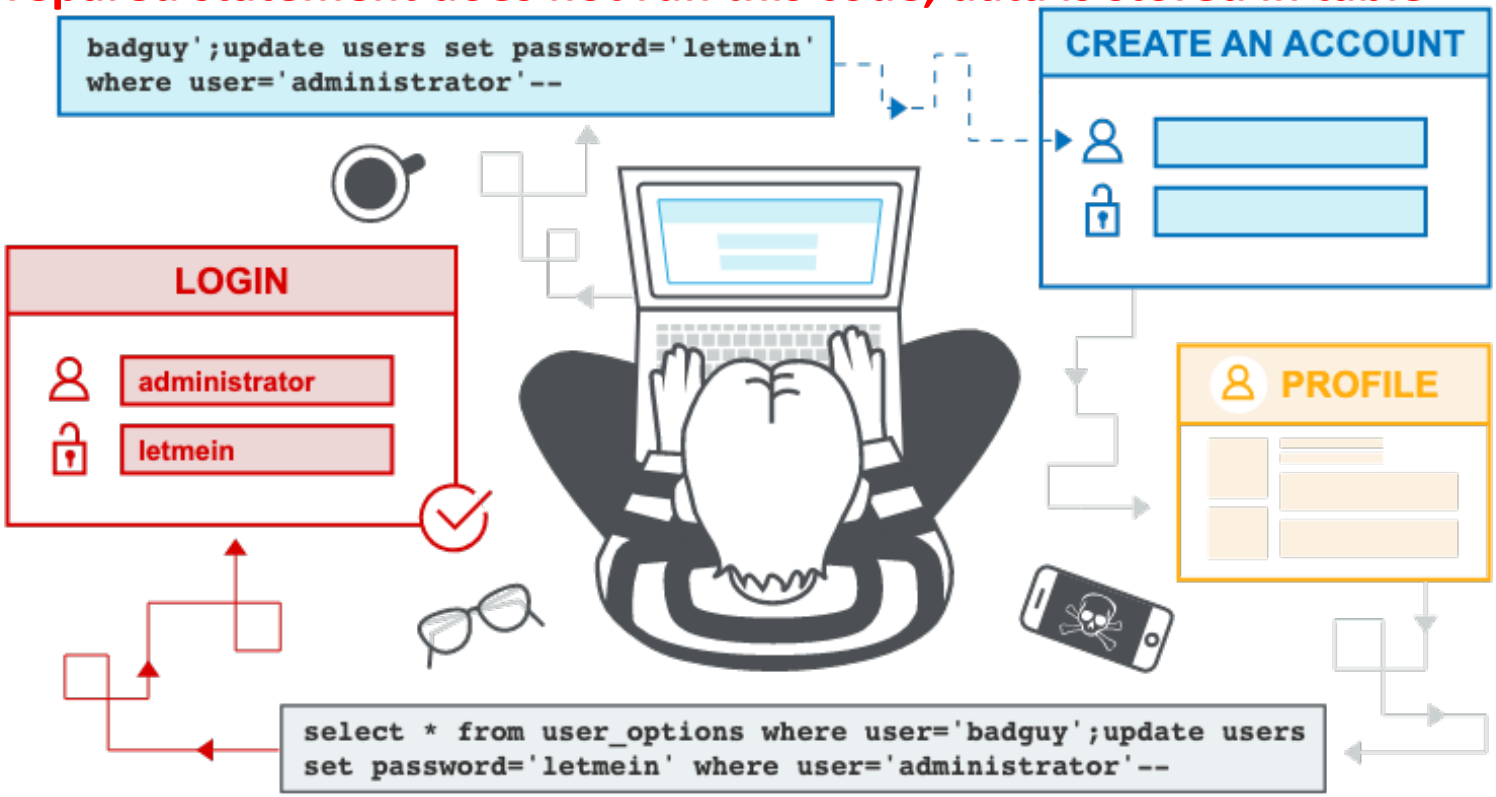
Du, Wenliang. *Computer & Internet Security: A Hands-on Approach*. 2019.

Even if you use prepared statements, be wary of data in your database!

Second-order attack:

User enters data with SQL embedded

Prepared statement does not run this code, data is stored in table



Can't trust data in database either!

Later someone runs a command where user = 'badguy'

Command executes; here resets admin password

Source: https://portswigger.net/web-security/sql-injection

SQL injection attacks are still being found!

August 2020

Freepik data breach: Hackers stole 8.3M records via SQL injection

By Sergiu Gatlan August 21, 2020 06:37 PM 0

8.3M usernames and passwords stolen via SQL injection attack

SQL injection attacks have been around for a long time

We should know better!

https://www.bleepingcomputer.com/news/security/freepik-data-breach-hackers-stole-83m-records-via-sql-injection/