Embedded SQL Demo - Cont'd

We continued wit the demo from last time. Just like before, I had two windows open, both connected to dbsrv1:

- Python window: where I run the python scripts.
- Psql window: where I run SQL commands in the psql shell - in the same way we have been doing so far.

SQL injection attack

We started by looking at `dynamic_danger.py` again - I encourage you to open the file and take a look at its contents.

Make sure you understand what the code does. As the file name suggests this way of using users' inputs is danger - we saw an example of that before reading week, but let's revisit that example again now.

Psql window

```
csc343h-marinat=> -- First, I repopulated my db since we make too many
csc343h-marinat=> -- modifications to our data last time.
csc343h-marinat=> \i ./343-26s/csc343db.sql
    ... output removed for the sake of space ...
csc343h-marinat=>
csc343h-marinat=> \i ./343-26s/csc343db_data.sql
    ... output removed for the sake of space ...
csc343h-marinat=>
csc343h-marinat=> -- Let's remember what is in our Course table before we run
csc343h-marinat=> -- the script.
csc343h-marinat=> SELECT * FROM Course;
 cnum |            name            | dept | breadth
------+----------------------------+------+---------
  343 | Intro to Databases         | CSC  | f
  207 | Software Design            | CSC  | f
  148 | Intro to Comp Sci          | CSC  | f
  263 | Data Struct & Anal         | CSC  | f
  320 | Intro to Visual Computing  | CSC  | f
  200 | Intro Archaeology          | ANT  | t
  203 | Human Biol & Evol          | ANT  | f
  150 | Organisms in Environ       | EEB  | f
  216 | Marine Mammal Bio          | EEB  | f
  263 | Compar Vert Anatomy        | EEB  | f
  110 | Narrative                  | ENG  | t
  205 | Rhetoric                   | ENG  | t
  235 | The Graphic Novel          | ENG  | t
  200 | Environmental Change       | ENV  | f
  320 | Natl & Intl Env Policy     | ENV  | f
  220 | Mediaeval Society          | HIS  | t
  296 | Black Freedom              | HIS  | t
  222 | COBOL programming          | CSC  | f
(18 rows)
```

Python window

```
dbsrv1:~/343-26s/afternoon$ python3 dynamic_danger.py
We are going to add a new course!
Course number: 108
Course name: Intro to CS
Department: CSC
```

Psql window

Let's check our course was added correctly.

```
csc343h-marinat=> SELECT * FROM Course;
 cnum |           name            | dept | breadth
------+---------------------------+------+---------
  343 | Intro to Databases        | CSC  | f
  207 | Software Design           | CSC  | f
  148 | Intro to Comp Sci         | CSC  | f
  263 | Data Struct & Anal        | CSC  | f
  320 | Intro to Visual Computing | CSC  | f
  200 | Intro Archaeology         | ANT  | t
  203 | Human Biol & Evol         | ANT  | f
  150 | Organisms in Environ      | EEB  | f
  216 | Marine Mammal Bio         | EEB  | f
  263 | Compar Vert Anatomy       | EEB  | f
  110 | Narrative                 | ENG  | t
  205 | Rhetoric                  | ENG  | t
  235 | The Graphic Novel         | ENG  | t
  200 | Environmental Change      | ENV  | f
  320 | Natl & Intl Env Policy    | ENV  | f
  220 | Mediaeval Society         | HIS  | t
  296 | Black Freedom             | HIS  | t
  222 | COBOL programming         | CSC  | f
  108 | Intro to CS               | CSC  |
(19 rows)
```

Yes! We can see a new row added for CSC108 - Great!

But what can go wrong? We talked at length before reading week at what a malicious user could do to cause a SQL injection attack. In particular, we saw how using the users' inputs as is without sanitization can cause issues. This is the problemmatic line:

```
cur.execute(f"INSERT INTO COURSE VALUES ({cnum}, '{name}', '{dept}');")
```

Thinking as a malicious user, what values for cnum, name and dept can we provide that will allow us to do things we really shouldn't be allowed to do? Make sure to see the `devious_inputs.txt` from last week that goes over why that line is particularaly problemmatic.

Let's say we are the student with sid `157` - can I inject an update that gets me a 100 on all my courses?

Here are this student's original grades:

```
csc343h-marinat=> SELECT * FROM Took WHERE sid = 157;
 sid | oid | grade
-----+-----+-------
 157 |   1 |    99
 157 |  14 |    98
 157 |  31 |    82
 157 |  21 |    71
 157 |  11 |    39
 157 |  34 |    62
 157 |  35 |    75
 157 |   3 |    82
 157 |   5 |    59
 157 |   6 |    72
 157 |   7 |    89
 157 |  28 |    91
 157 |  13 |    90
 157 |  26 |    71
 157 |  17 |    59
(15 rows)
```

Python window

```
dbsrv1:~/343-26s/afternoon$ python3 dynamic_danger.py
We are going to add a new course!
Course number: 111
Course name: Malicious user
Department: CSC'); UPDATE Took SET grade = 100 WHERE sid = 157;--
```

Psql window

And now, the student got a 100 on all his courses.

```
csc343h-marinat=> SELECT * FROM Took WHERE sid = 157;
 sid | oid | grade
-----+-----+-------
 157 |   1 |   100
 157 |  14 |   100
 157 |  31 |   100
 157 |  21 |   100
 157 |  11 |   100
 157 |  34 |   100
 157 |  35 |   100
 157 |   3 |   100
 157 |   5 |   100
 157 |   6 |   100
 157 |   7 |   100
 157 |  28 |   100
 157 |  13 |   100
 157 |  26 |   100
 157 |  17 |   100
(15 rows)
```

Moral of the story

NEVER, NEVER, NEVER use string concatenation (that includes f-strings and the other unsafe formats outlined in the file `dynamic_safe.py` as comments) to embed user's inputs into a query string. To prevent injection attacks, pass two arguments to `execute`:

1. the query with placeholders where needed (`"%s"`, and note that this is the placeholder regardless of the attribute type), and
2. a list containing a value for each placeholder, in order.

We also talked about an alternative that allows you to use a dictionary as a second argument instead. See the file `dynamic_safe.py` for detail.

The `execute` method "sanitizes" our input to ensure that the values are treated as their correct type e.g., a string and not embedded as SQL commands. Let's run our `dynamic_safe.py` that uses this approach. This time we will try to update the grades of the student with sid 99132.

Python window

```
dbsrv1:~/343-26s/afternoon$ python3 dynamic_safe.py
We are going to add a new course!
Course number: 112
Course name: Malicious but will fail
Department: CSC'); UPDATE Took SET grade = 100 WHERE sid = 99132; --
An Error occurred!
value too long for type character varying(20)
```

Yeay! the injection attack was prevented. Make sure to never use string concatenation, python formatted strings, … etc for dynamic queries i.e., queries that use users' inputs. Always santizie your inputs!