Tutorial 08
Django, SQL Injection

Big Data Engineering

Prof. Dr. Jens Dittrich

bigdata.uni-saarland.de

June 18, 2025

Repetition - Question 1

Question

What is an Object-Relational Mapper? What are its advantages?

Repetition - Question 2

Question

What are ‘through’ tables? How can you add additional attributes to them?

Repetition - Question 3

Question

How can you model uniques in Django?

Repetition - Question 4

Question

What are the different scenarios in which GET and POST requests are used?

Repetition - Question 5

Question

What is a SQL Injection?

Repetition - Question 6

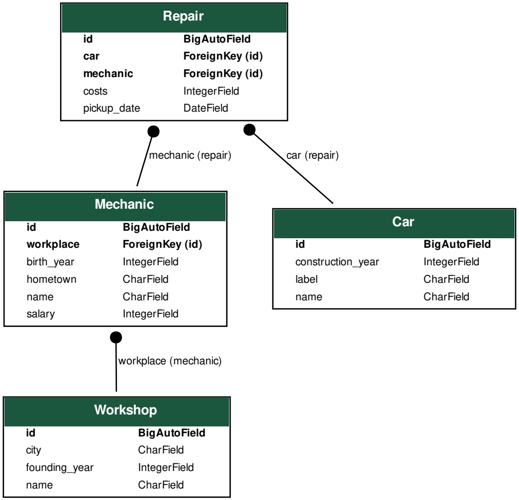
Question

How can a SQL Injection be prevented?

Exercise 1

Exercise

Consider the Django model below.



Repair	
id	**BigAutoField**
car	**ForeignKey (id)**
mechanic	**ForeignKey (id)**
costs	IntegerField
pickup_date	DateField

Mechanic	
id	**BigAutoField**
workplace	**ForeignKey (id)**
birth_year	IntegerField
hometown	CharField
name	CharField
salary	IntegerField

Car	
id	**BigAutoField**
construction_year	IntegerField
label	CharField
name	CharField

Workshop	
id	**BigAutoField**
city	CharField
founding_year	IntegerField
name	CharField

Exercise 1.1

Question

Provide QuerySets for the following natural language statements.

- The unique birth years of mechanics that live in Neunkirchen.
- The construction years of cars that have been repaired more than 6 times by a mechanic who works for a workshop that has the name 'CarRepair'. The output should only contain the first 5 tuples and should be sorted in descending order according to the label of the car.

Exercise 1.2

Question

Translate the following SQL queries into QuerySets.

-
  ```
  SELECT name
  FROM   workshop
  WHERE  founding_year <> 2000;
  ```

-
  ```
  SELECT DISTINCT w.city
  FROM     mechanic AS m
           JOIN workshop AS w
                ON m.workplace = w.id
  WHERE    m.salary < 500
  GROUP BY w.id, w.city
  HAVING   COUNT(*) = 4
  ORDER BY w.city
  LIMIT 10;
  ```

Exercise 2.1

Question

How can you manipulate the following SQL statement to get the `secret` of the user `admin`? You can assume that the user and the corresponding entry exist.

```
statement = f"""
SELECT secret
FROM   users
WHERE  (user = '{enteredUser}')
       AND (pw = '{enteredPW}');
"""
```

Exercise 2.2

Question

How can you manipulate the following SQL statement to get the secret of the user `admin` and also change the password? You can assume that this user and the corresponding entries exist.

```
statement = f"""
SELECT non_secret_data
FROM   users
WHERE  user = '{enteredUser}';
"""
```

Exercise 3

Question

Below is a code snippet from a database of a social media website that allows its users to provide a short biography. How can you manipulate this code to delete the entire `users` table? You can assume that the table `users` exists.

Exercise 3

Question

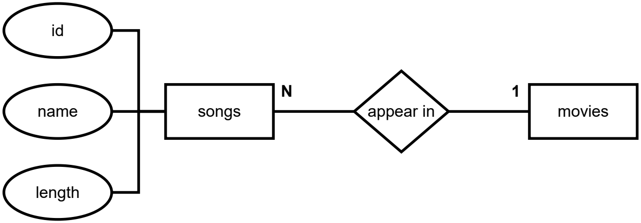
```
def update_bio(username):
    new_bio = input("Please enter a short biography: ")
    # sanitize input to prevent SQL Injection
    sanitized_bio = sanitize(new_bio)
    cur = conn.cursor()
    cur.execute(f"UPDATE users SET bio='{sanitized_bio}'
                WHERE username=%s;", (username,))

def sanitize(string):
    keywords = ["SELECT", "FROM", "WHERE", "UPDATE",
                "SET", "DROP", "TABLE"]
      for word in keywords:
          string = string.replace(word, "")
      string = string.replace("--", "-")
      return string
```

Exercise 4

Question

We now want to want to extend the Django model presented in the lecture with songs and add associated functionalities to it.
The songs should be connected to the existing model according to the ER-model below. Note that the `length` represents the time in seconds.
We also want to offer a webpage, that shows songs sorted in ascending order according to the rank of the movie they appear in. Further, only 3 songs should be shown.
You can either implement the desired extension in the attached Django project directly or describe in words the steps you would do in order to implement it.



Exercise 4

Hints

- You are mostly free on how you want to approach this exercise. You only need to ensure that the table of the underlying database that models the songs is called `songs`.
- We already provide a sample HTML-template `list_songs`, where you only need to add the functionality to read the provided songs from the given context and show them on the webpage.
- We provide some sample songs in the file `imdb_data.json` that you can use to test your implementation.