Prepared Statements and SQL Injections: Takeaways

Syntax

- Using dictionary for placeholders in the `cursor.execute()` method:

```
cur.execute("INSERT INTO users VALUES (%(id)s, %(email)s, %(name)s, %(address)s)", {
    'address': '124, Fake Street'
    'name': 'John',
    'id': 1000,
    'email': 'hello@dataquest.io',
})
```

- Preparing a statement:

```
cur.execute("""
    PREPARE insert_user(integer, text, text, text) AS
    INSERT INTO users VALUES ($1, $2, $3, $4)
""")
```

- Executing a prepared statement:

```
cur.execute("EXECUTE insert_user(1001, 'bob@dataquest.io', 'Bob', '101 Fake Street')")
```

Concepts

- When queries are designed for user input, it is not safe to use string formatting to build up the query string from the user inputs as this is prone to SQL injections.
- SQL injections are the ability of a user to user badly parsed queries to replace an input value by SQL code in a way that it will be execute on the database server.
- Postgres offers two mechanisms to prevent SQL injections. One is to pass the arguments as the second argument of the `cursor.execute()` method. The other one is to use prepared statements.
- A prepared statement is a named query where the only missing information are the query values. These are parsed and planned at creation and so replacing the values later on will never lead to them being executed.
- A query goes thought four steps before it is executed:
    - The query is parsed for correct syntax.
    - If there are no syntax errors, the query is transformed into something that the SQL engine can understand and execute.
    - Using the result from step 2 a query plan is created that tries to find the most efficient way to execute the query.

        - The plan produced on step 3 is executed on the database.
    - Prepared statements are often more efficient as they need not to be planned and are executed directly.

Resources

- Prepared statements in Postgres
- SQL injections