Django 1-day Analysis

STEALIEN 윤석찬 연구원

제 26회 해킹캠프, 2023.02.12.

POC SECURITY

Presenter

Django 1-day Analysis

윤석찬

- STEALIEN 선제대응팀 연구원
- 경희대학교 컴퓨터공학과 재학
- 웹해킹
- http://fb.com/ch4n3.yoon

Agenda

Django 1-day Analysis

- Overview of Django and its ORM
- Overview of released 1-day vulnerabilities
- How Django makes SQL Query?
- CVE-2022-28346
- How was it patched?
- How to report security issue?
- Conclusion

Announcement

Django 1-day Analysis

- 시간 관계상 SQL Query에 대해 안다고 가정
- 모르는게 있다면 나중에 언제든지 연락주세요!

Overview of
Django and its ORM

1-1. Why we use Django?

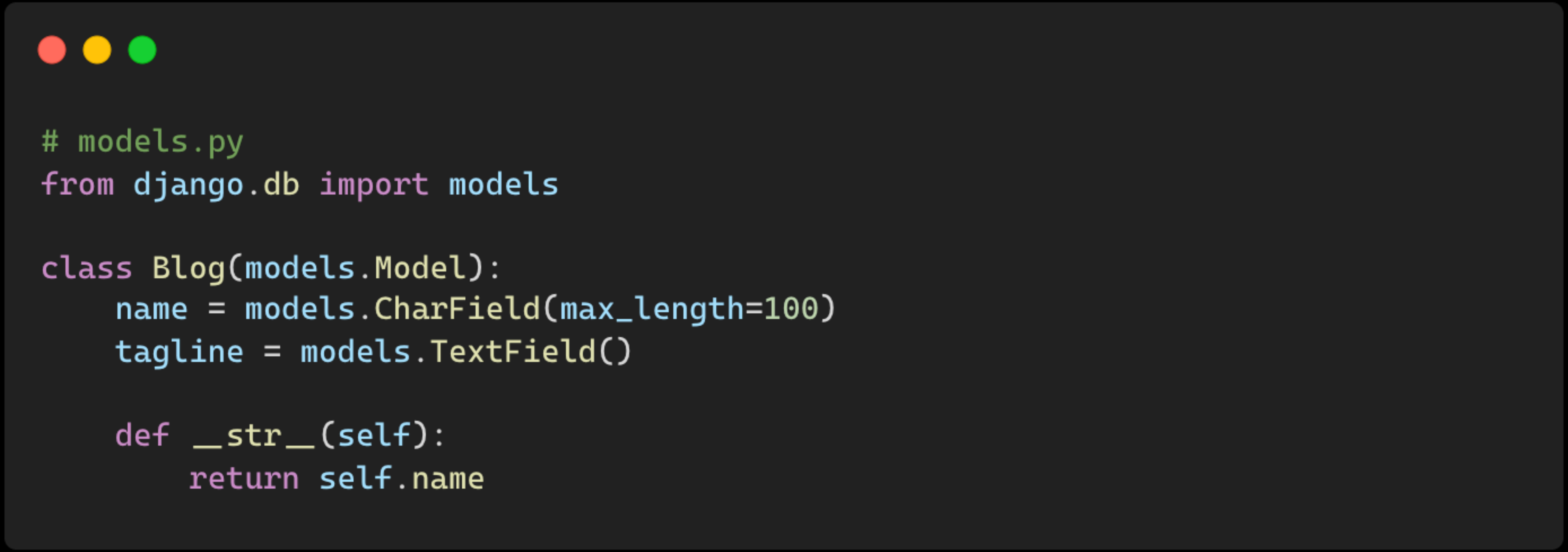
Overview of Django and its ORM

- Powered by Python
- Provides Default Admin Panel
- **Highly Secure**
- Provides **Powerful ORM**
- Offers Rapid Development

1-2. ORM?

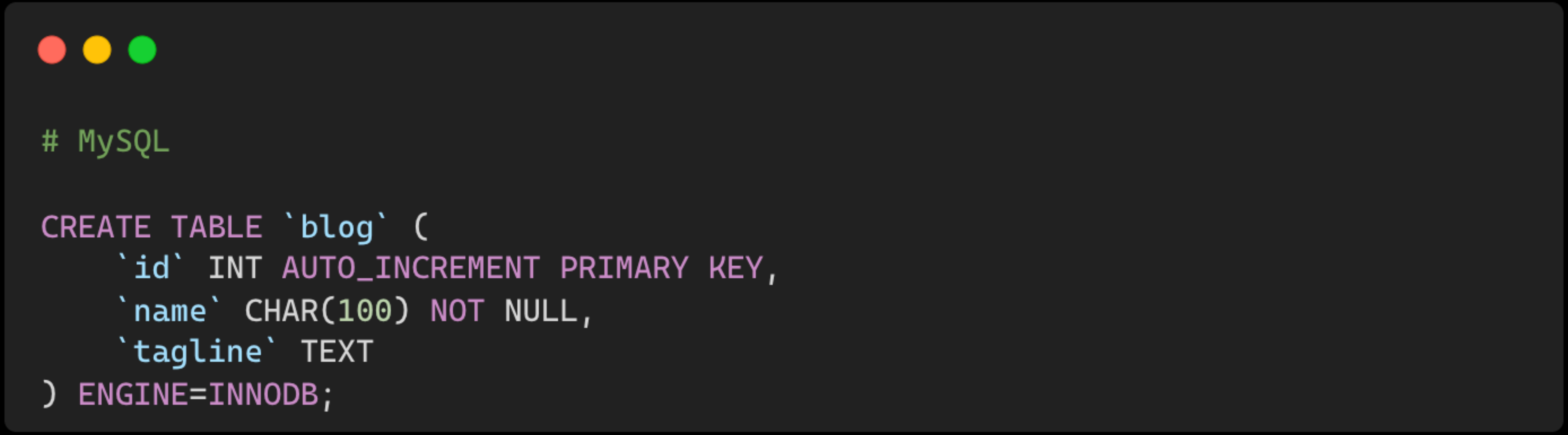
Overview of Django and its ORM

- Object Relational Mapping (ORM)
- Creates “**Bridge**” between Object-Oriented Program and DBMS
- Can access to database with ONLY Python code



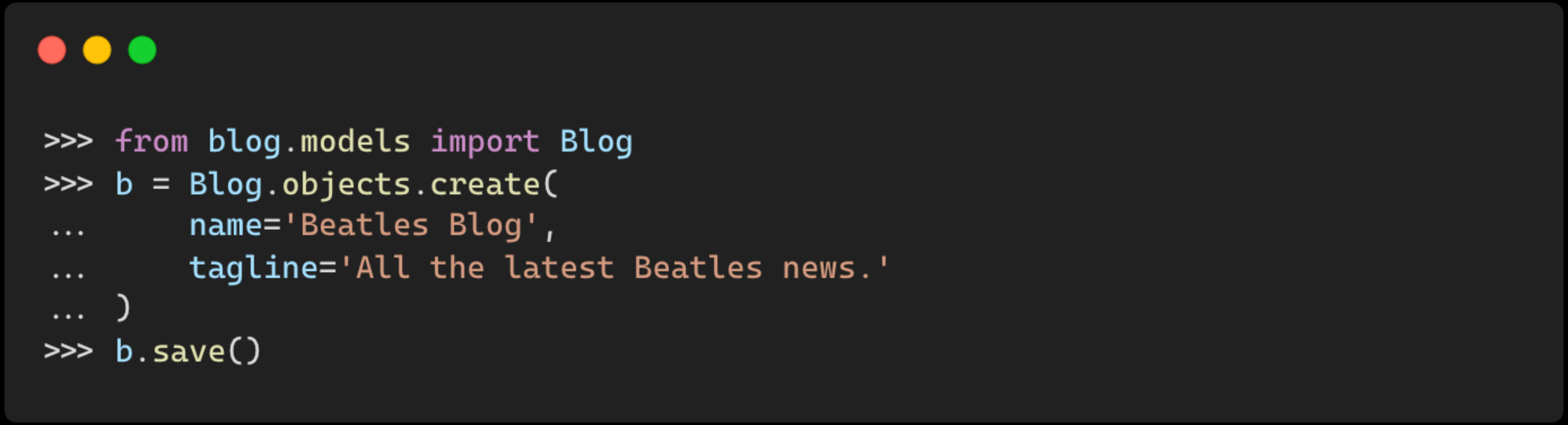
```python
# models.py
from django.db import models

class Blog(models.Model):
    name = models.CharField(max_length=100)
    tagline = models.TextField()

    def __str__(self):
        return self.name
```

ORM code to create table



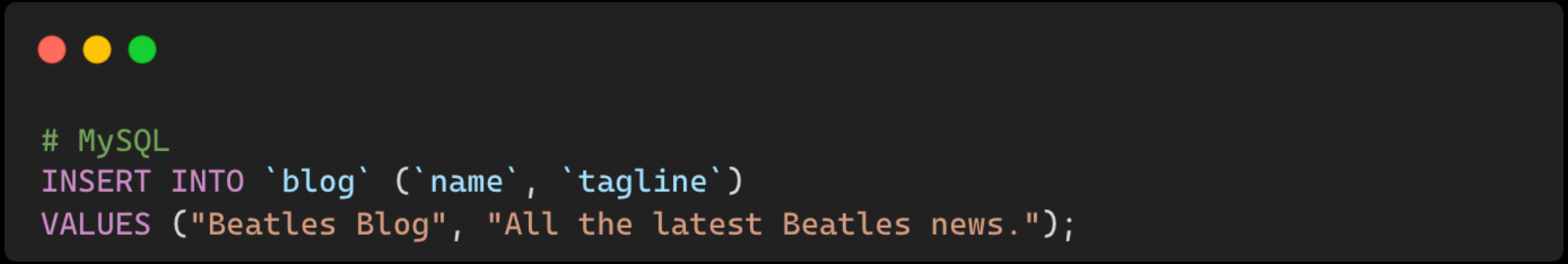
```
# MySQL

CREATE TABLE `blog` (
    `id` INT AUTO_INCREMENT PRIMARY KEY,
    `name` CHAR(100) NOT NULL,
    `tagline` TEXT
) ENGINE=INNODB;
```

SQL Query to create table



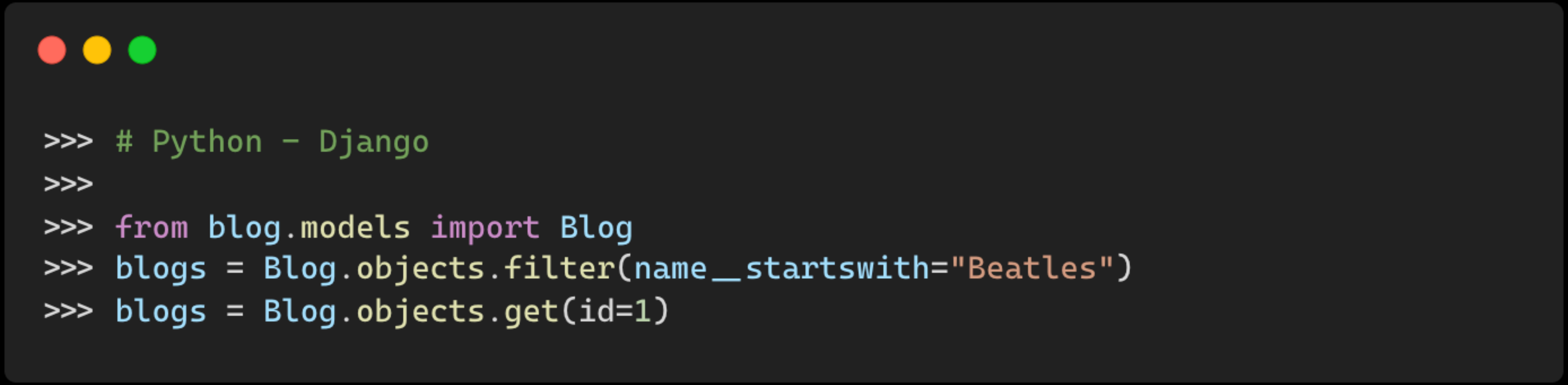
```
>>> from blog.models import Blog
>>> b = Blog.objects.create(
...     name='Beatles Blog',
...     tagline='All the latest Beatles news.'
... )
>>> b.save()
```

ORM code to create an entity



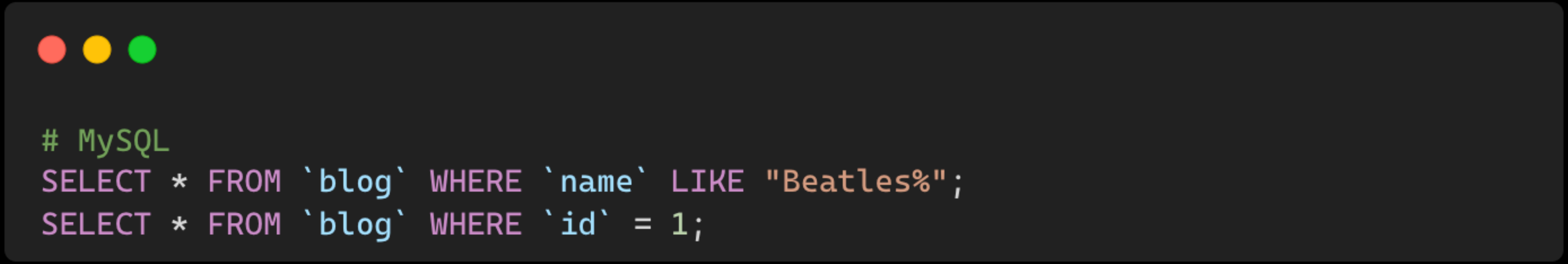
```
# MySQL
INSERT INTO `blog` (`name`, `tagline`)
VALUES ("Beatles Blog", "All the latest Beatles news.");
```

SQL Query to create an entity



```
>>> # Python - Django
>>>
>>> from blog.models import Blog
>>> blogs = Blog.objects.filter(name__startswith="Beatles")
>>> blogs = Blog.objects.get(id=1)
```

ORM code to search entities



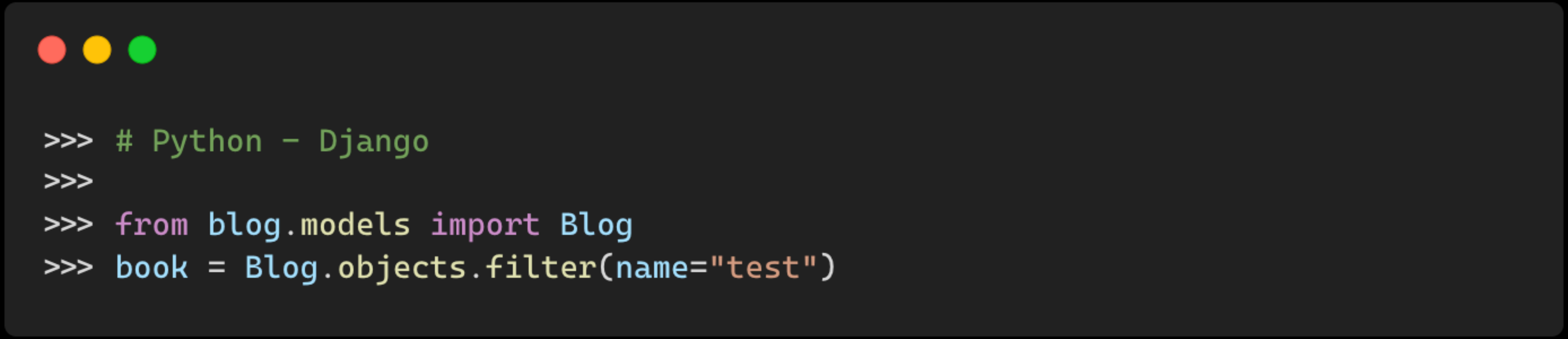
```sql
# MySQL
SELECT * FROM `blog` WHERE `name` LIKE "Beatles%";
SELECT * FROM `blog` WHERE `id` = 1;
```

SQL Query to search entities

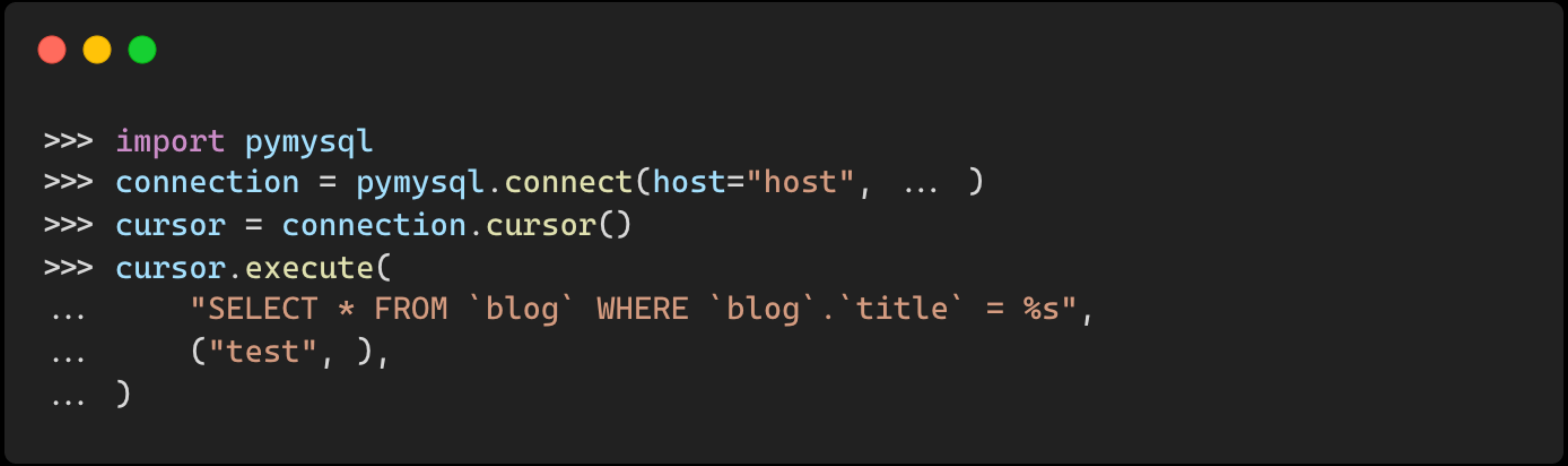
1-3. How secure is ORM?

Overview of Django and its ORM

- **FULLY SAFE** to SQL Injection yet 🔐

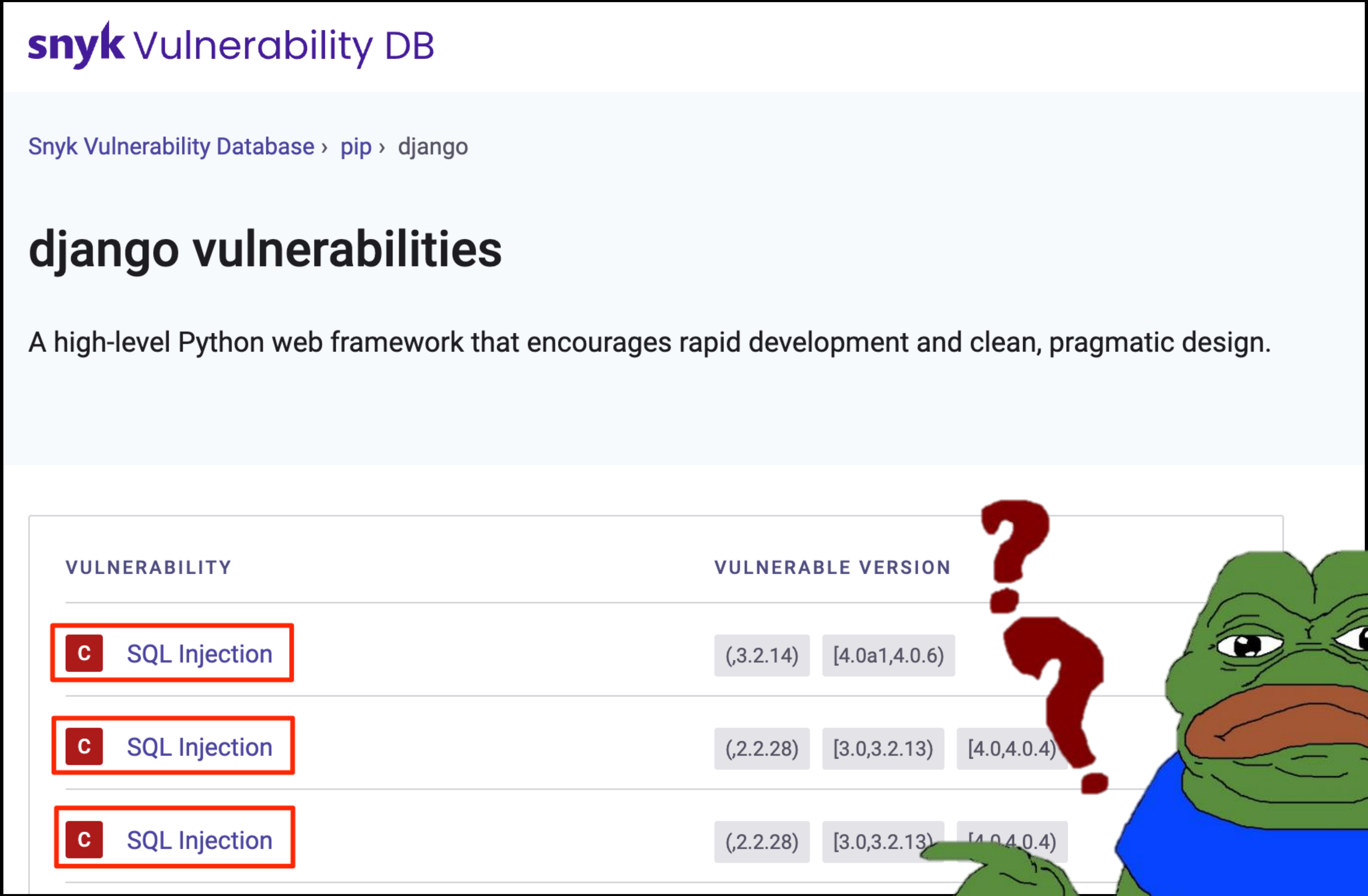


```
>>> # Python - Django
>>>
>>> from blog.models import Blog
>>> book = Blog.objects.filter(name="test")
```



```
>>> import pymysql
>>> connection = pymysql.connect(host="host", ... )
>>> cursor = connection.cursor()
>>> cursor.execute(
...     "SELECT * FROM `blog` WHERE `blog`.`title` = %s",
...     ("test", ),
... )
```

But.. Why.. Whyrano..



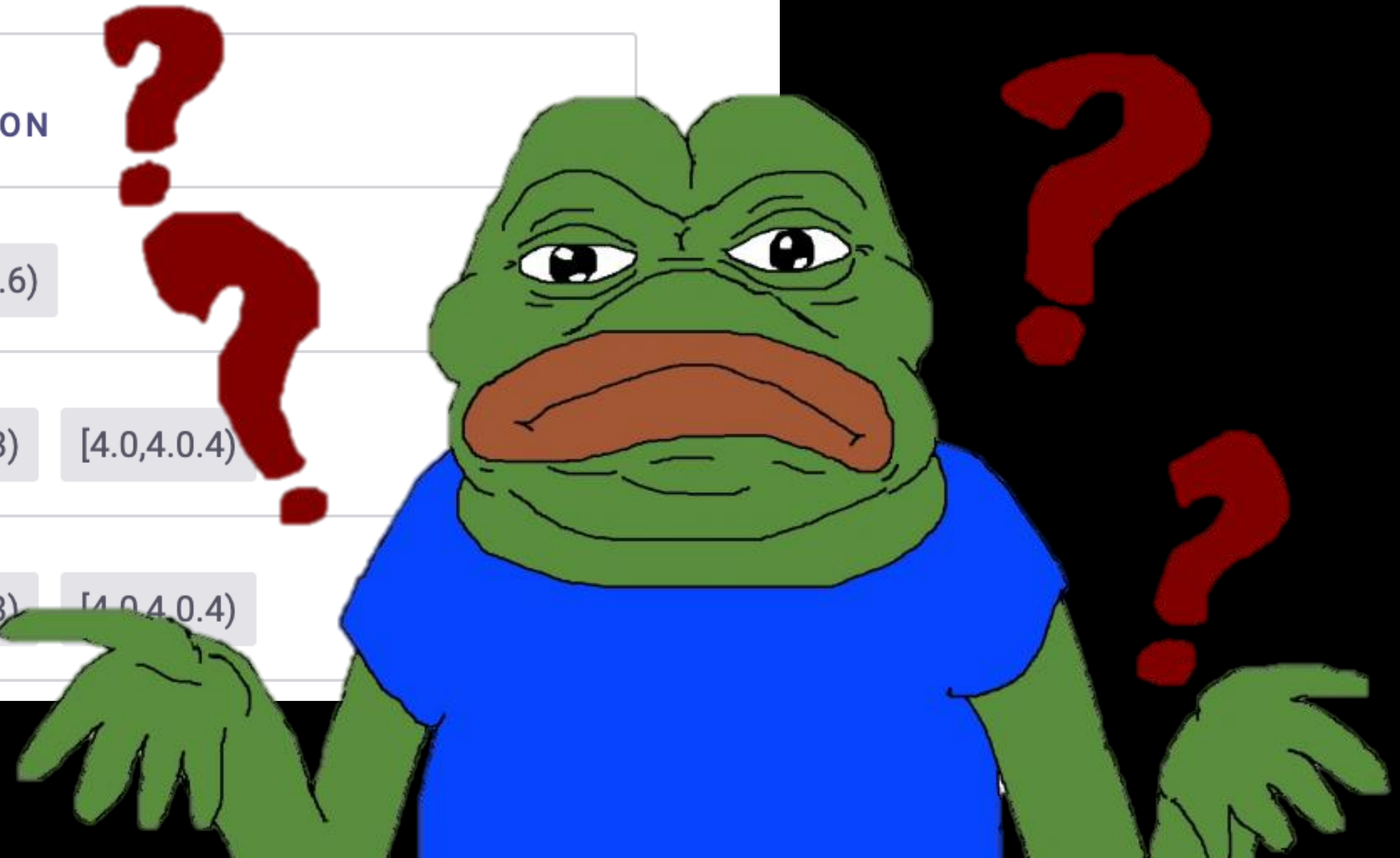
snyk Vulnerability DB

Snyk Vulnerability Database › pip › django

django vulnerabilities

A high-level Python web framework that encourages rapid development and clean, pragmatic design.

VULNERABILITY	VULNERABLE VERSION
C SQL Injection	(,3.2.14) [4.0a1,4.0.6)
C SQL Injection	(,2.2.28) [3.0,3.2.13) [4.0,4.0.4)
C SQL Injection	(,2.2.28) [3.0,3.2.13) [4.0,4.0.4)

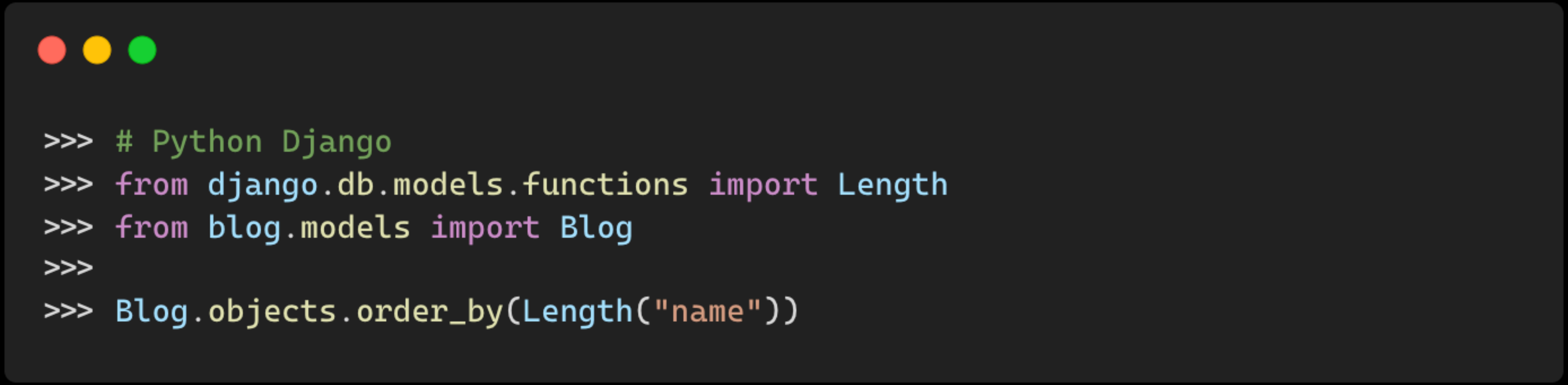


Overview of released 1-day vulnerabilities

2-1. SQL Functions in ORM

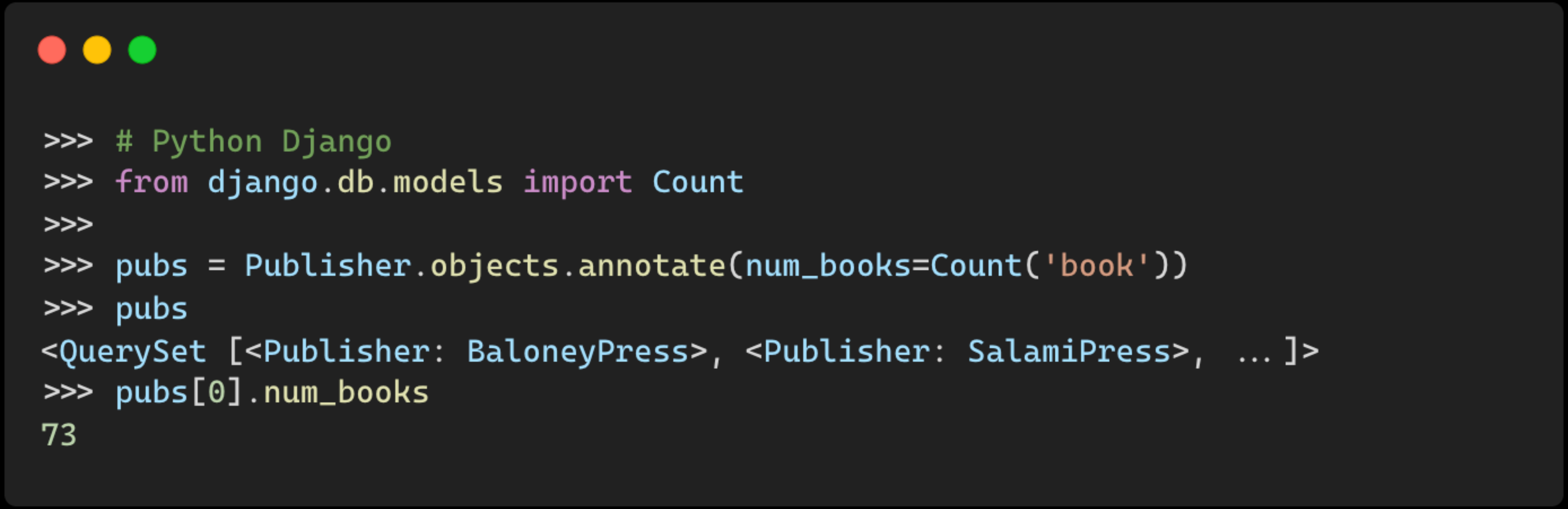
Overview of released 1-day vulnerabilities

- All Databases have many **built-in functions**
- SQL have many **aggregate functions**
    - e.g. `GROUP BY`, `HAVING`
- ORM should implement all SQL functions

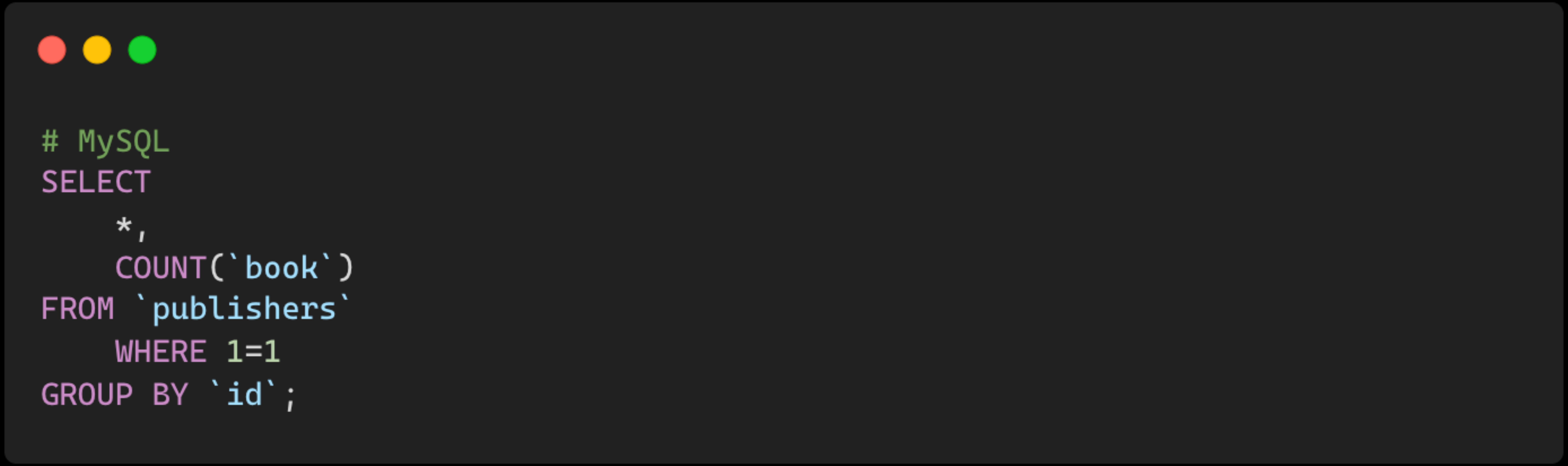


```
>>> # Python Django
>>> from django.db.models.functions import Length
>>> from blog.models import Blog
>>>
>>> Blog.objects.order_by(Length("name"))
```

Using SQL `LENGTH()` function in Django ORM


```
>>> # Python Django
>>> from django.db.models import Count
>>>
>>> pubs = Publisher.objects.annotate(num_books=Count('book'))
>>> pubs
<QuerySet [<Publisher: BaloneyPress>, <Publisher: SalamiPress>, ...]>
>>> pubs[0].num_books
73
```

Using aggregate function **COUNT()** in Django ORM

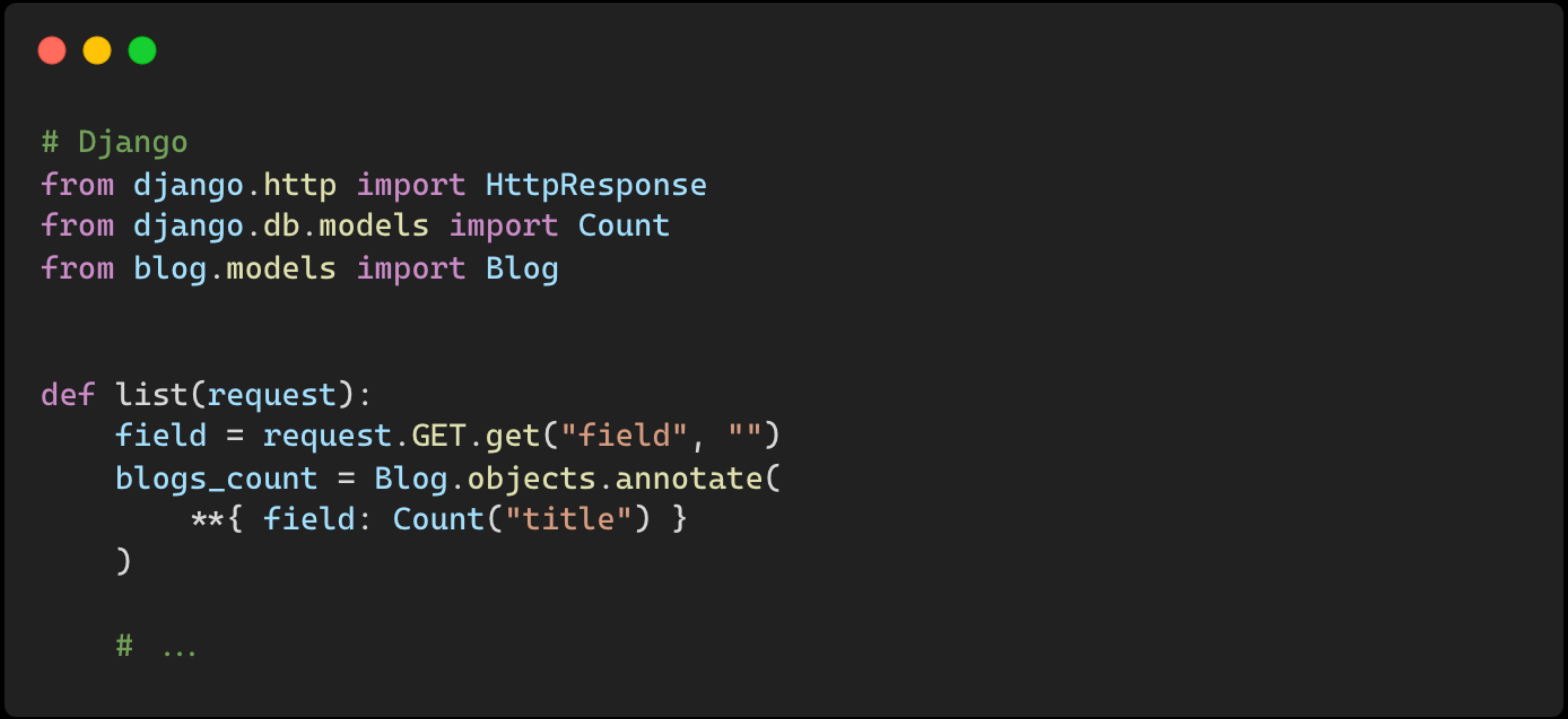


```sql
# MySQL
SELECT
    *,
    COUNT(`book`)
FROM `publishers`
    WHERE 1=1
GROUP BY `id`;
```

2-2. CVE-2022-28346

Overview of released 1-day vulnerabilities

- Potential SQL Injection in **`QuerySet.annotate()`** method
- SQL Injection **in column aliases**
  via crafted dictionary and **dictionary expansion**
- If **user input affects column aliases**,
  it is vulnerable to SQL Injection

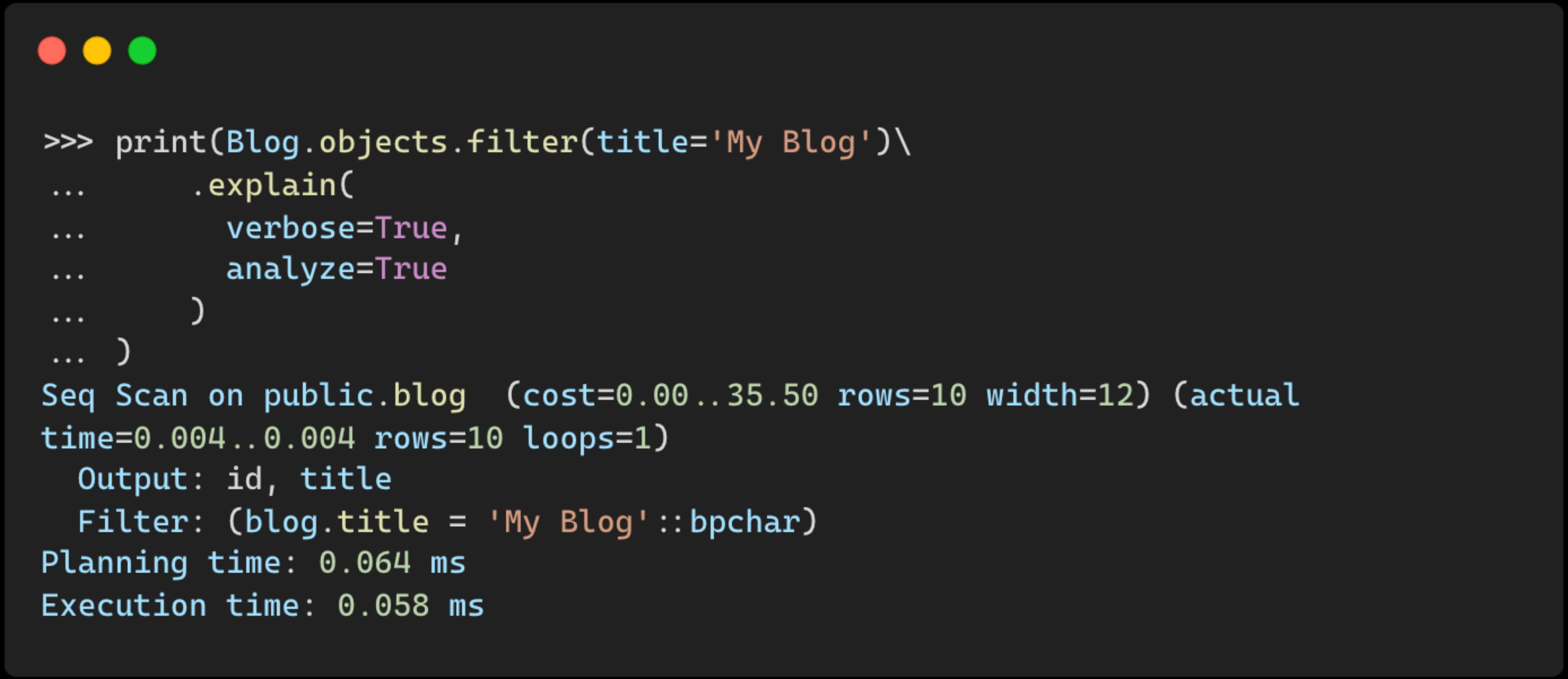


```python
# Django
from django.http import HttpResponse
from django.db.models import Count
from blog.models import Blog


def list(request):
    field = request.GET.get("field", "")
    blogs_count = Blog.objects.annotate(
        **{ field: Count("title") }
    )

    # ...
```

2-3. CVE-2022-28347

Overview of released 1-day vulnerabilities

- Potential SQL injection
  via `QuerySet.explain(**options)` on **PostgreSQL**
- `EXPLAIN` Query



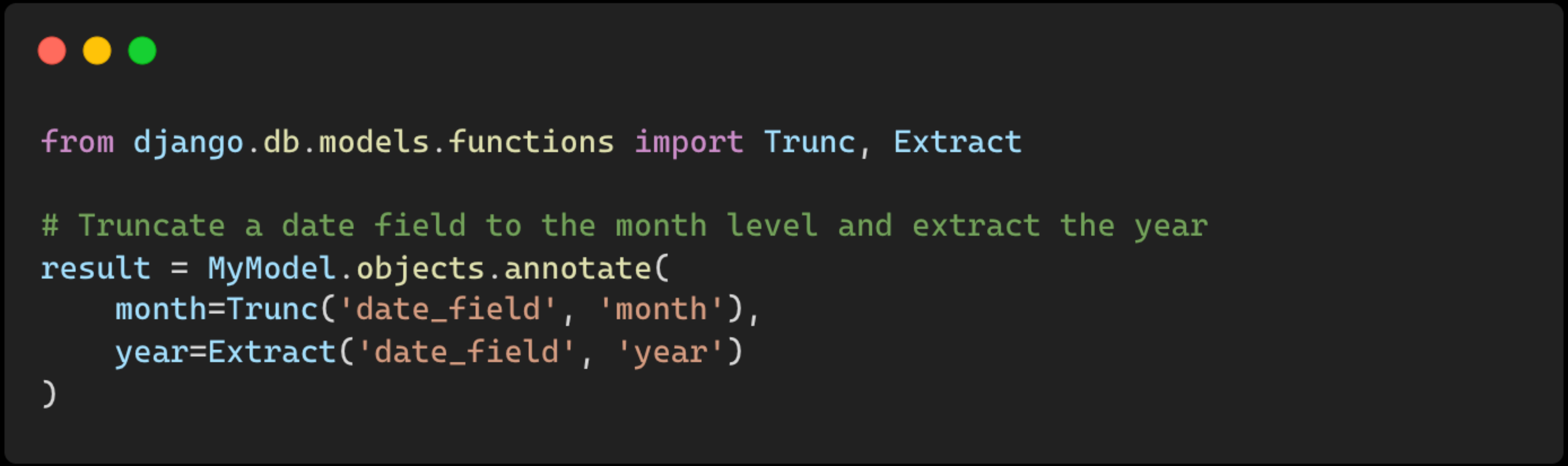
```
>>> print(Blog.objects.filter(title='My Blog')\
...       .explain(
...         verbose=True,
...         analyze=True
...       )
... )
Seq Scan on public.blog  (cost=0.00..35.50 rows=10 width=12) (actual
time=0.004..0.004 rows=10 loops=1)
  Output: id, title
  Filter: (blog.title = 'My Blog'::bpchar)
Planning time: 0.064 ms
Execution time: 0.058 ms
```

Example for **EXPLAIN** Query

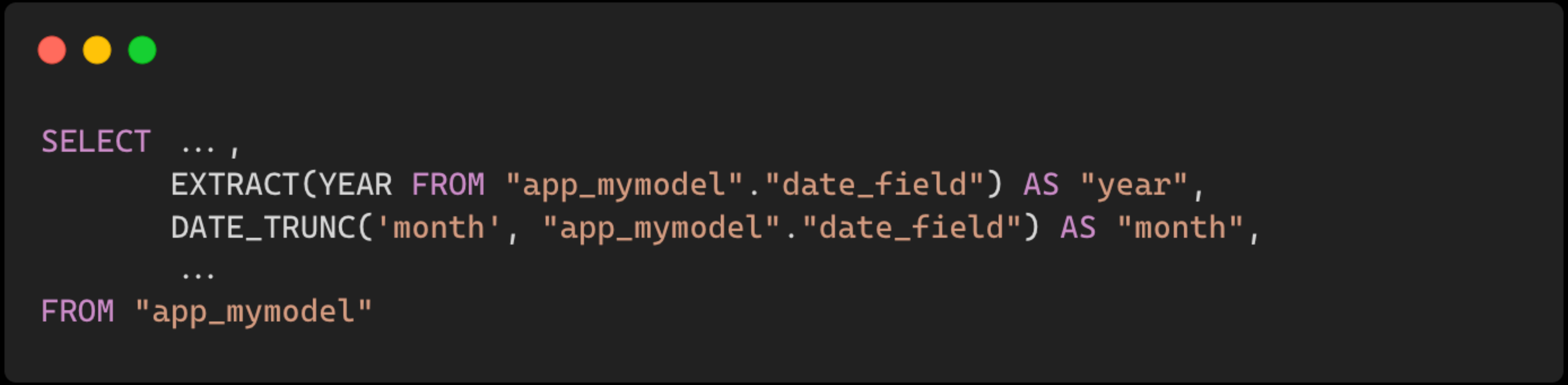
2-4. CVE-2022-34265

Overview of released 1-day vulnerabilities

- Potential SQL injection
  via `Trunc(kind)` and `Extract(lookup_name)` arguments.
- These implement SQL Functions in Python
  - TRUNC: **truncate a date or a string** to a specific level of precision
  - Extract


```python
from django.db.models.functions import Trunc, Extract

# Truncate a date field to the month level and extract the year
result = MyModel.objects.annotate(
    month=Trunc('date_field', 'month'),
    year=Extract('date_field', 'year')
)
```

django.db.models.functions.Trunc, .Extract



```sql
SELECT ...,
       EXTRACT(YEAR FROM "app_mymodel"."date_field") AS "year",
       DATE_TRUNC('month', "app_mymodel"."date_field") AS "month",
       ...
FROM "app_mymodel"
```

SQL Query which will be crafted by Django ORM

How Django makes REAL SQL Query?

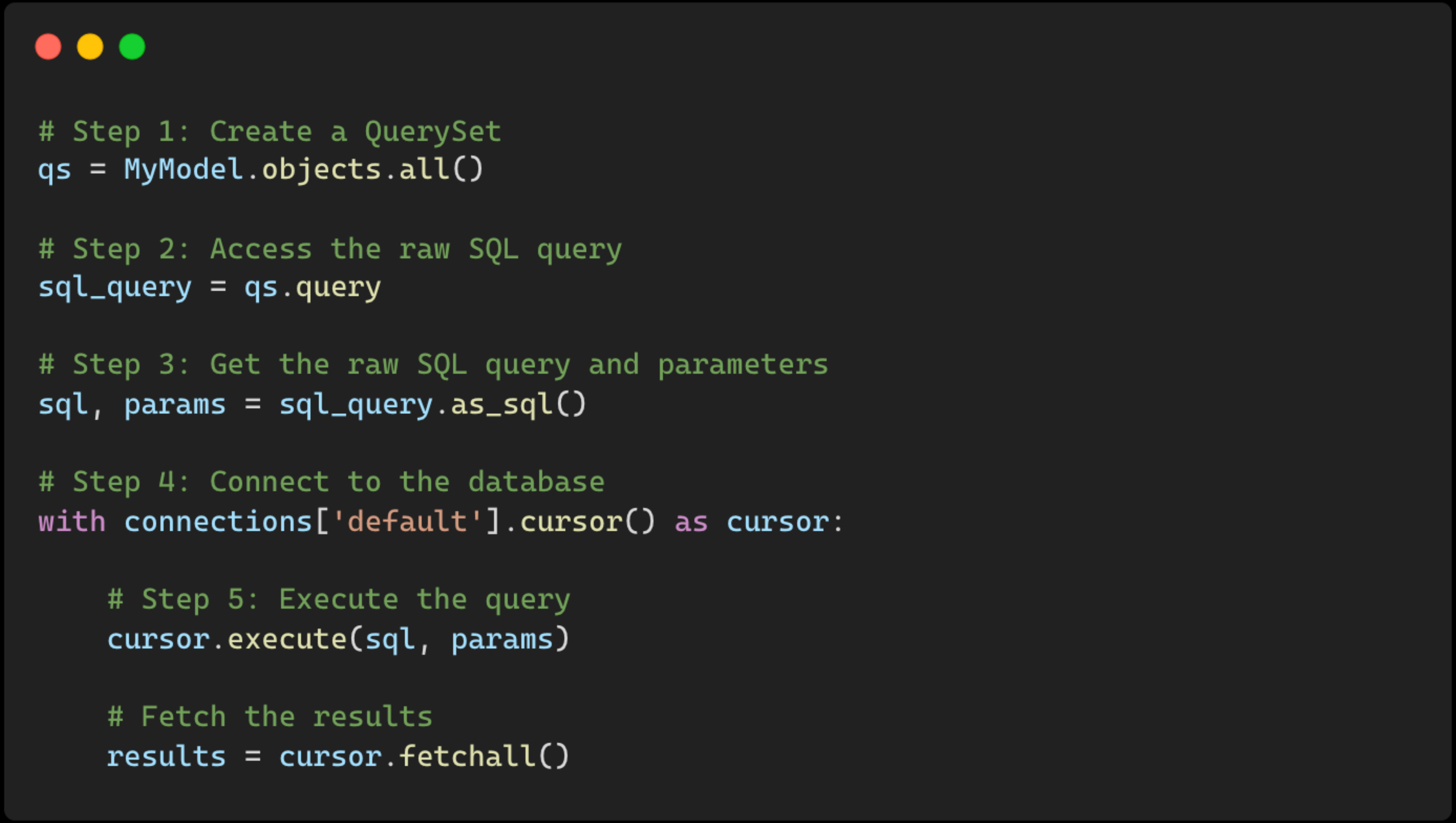
3-1. `QuerySet`
How Django makes REAL SQL Query?

- It creates a bridge
  - **SELECT**
    - `all()`, `filter()`, `get()`, …
  - **INSERT**
    - `create()`
  - **ALIAS**
    - `annotate()`, `aggregate()`, …

3-2. Process to make raw SQL query

How Django makes REAL SQL Query?

1. Create a `QuerySet`, and **run methods of `QuerySet`**
2. Use the `` `.query` `` attribute
   - This returns a `` `django.db.models.sql.query.Query` `` object
3. Run the **`` `.as_sql()` `` method** of `` `Query` `` object
4. **Connect** to the database: **`` `django.db.connection.cursor()` ``**
5. **Execute** the SQL query: `` `cursor().execute()` ``



```python
# Step 1: Create a QuerySet
qs = MyModel.objects.all()

# Step 2: Access the raw SQL query
sql_query = qs.query

# Step 3: Get the raw SQL query and parameters
sql, params = sql_query.as_sql()

# Step 4: Connect to the database
with connections['default'].cursor() as cursor:

    # Step 5: Execute the query
    cursor.execute(sql, params)

    # Fetch the results
    results = cursor.fetchall()
```

How Django crafts raw SQL query and executes it

3-3. Then.. we should analyze.. what?

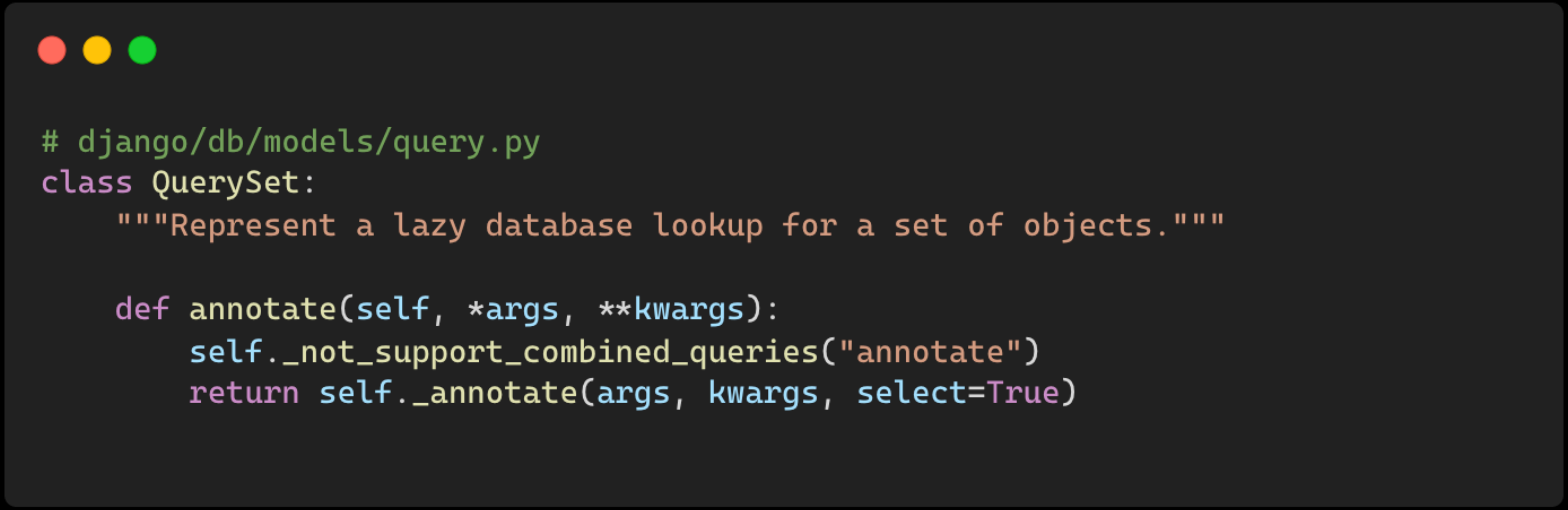
How Django makes REAL SQL Query?

CVE-2022-28346

Step by step!

4-1. QuerySet.annotate()

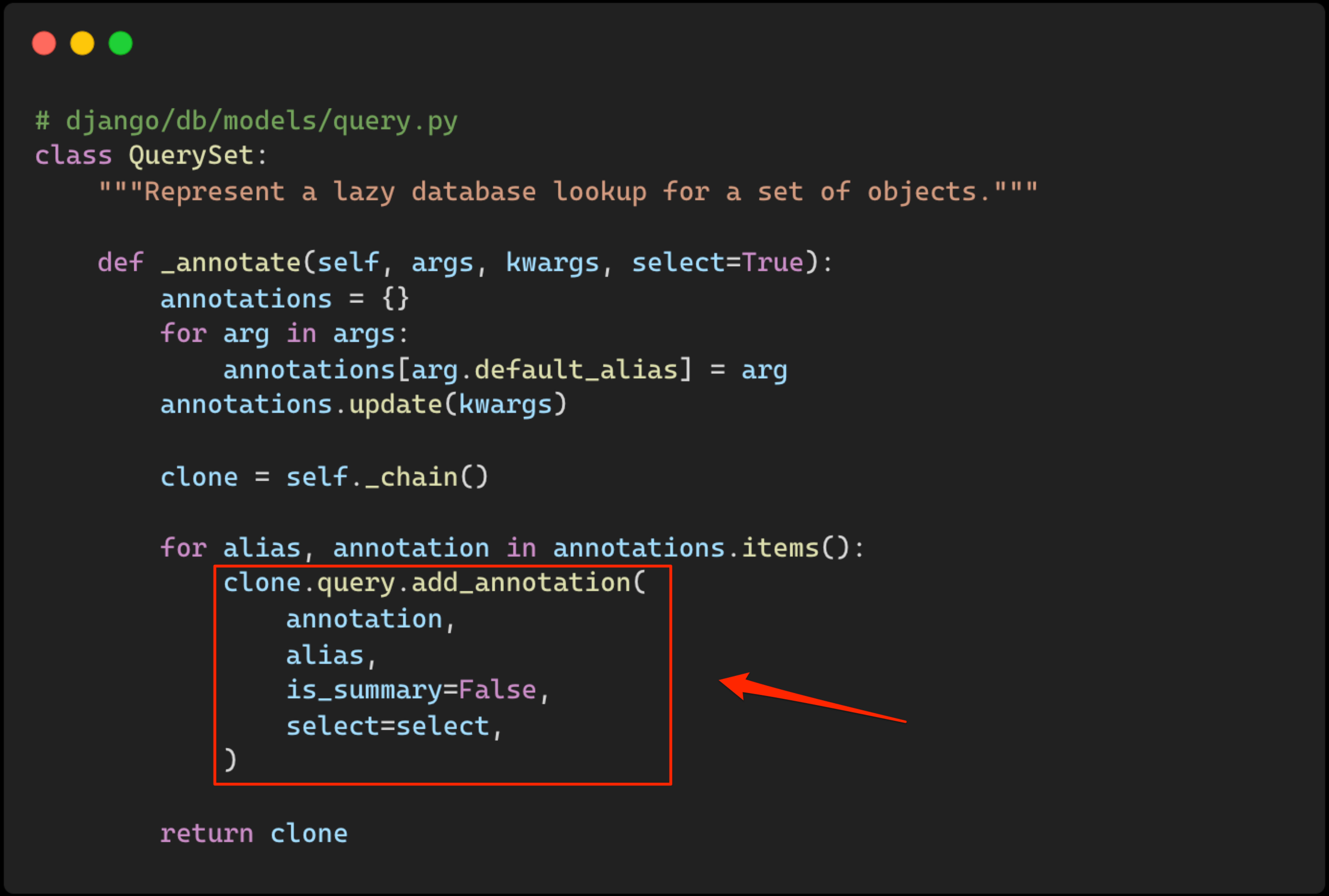
CVE-2022-28346 Step by step!



```python
# django/db/models/query.py
class QuerySet:
    """Represent a lazy database lookup for a set of objects."""

    def annotate(self, *args, **kwargs):
        self._not_support_combined_queries("annotate")
        return self._annotate(args, kwargs, select=True)
```

4-2. `QuerySet._annotate()`

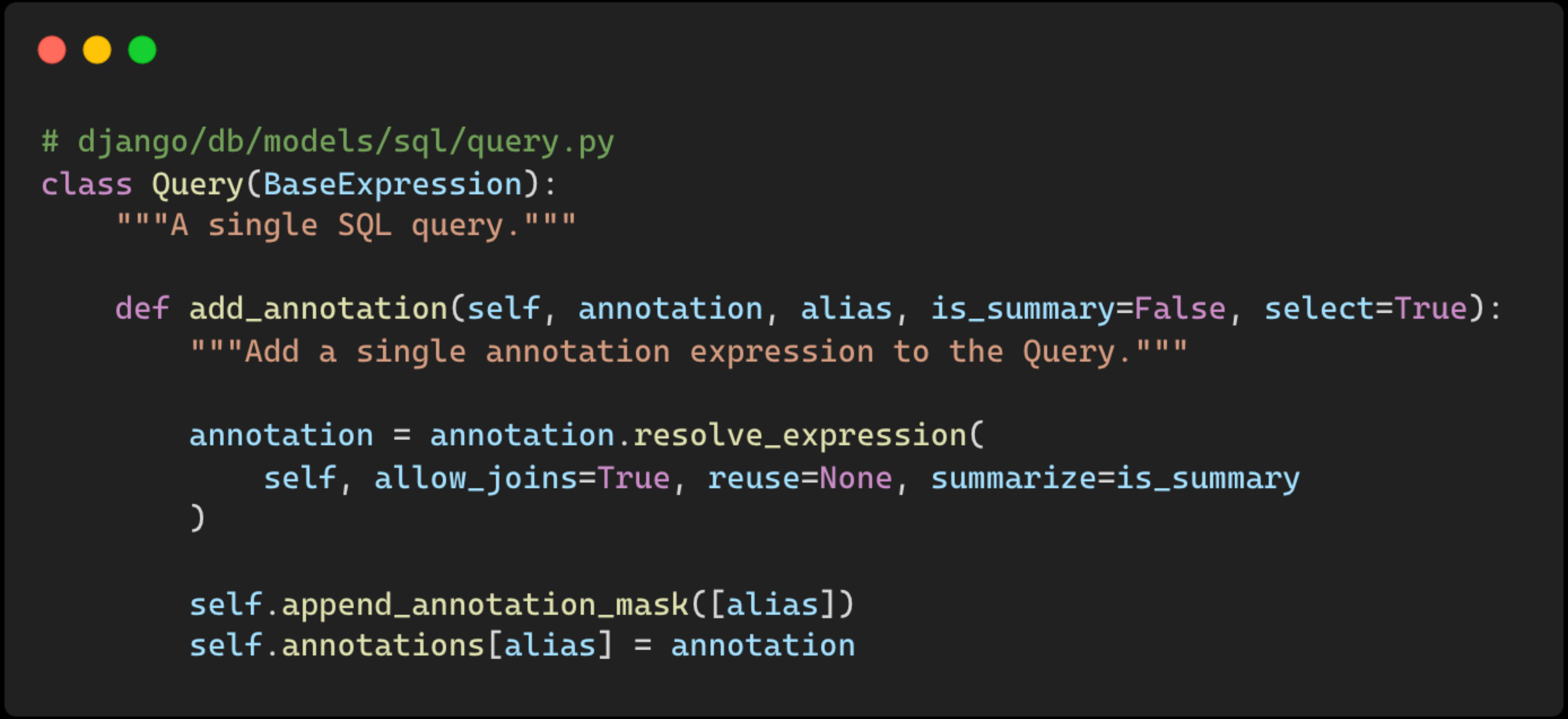
CVE-2022-28346 Step by step!



```python
# django/db/models/query.py
class QuerySet:
    """Represent a lazy database lookup for a set of objects."""

    def _annotate(self, args, kwargs, select=True):
        annotations = {}
        for arg in args:
            annotations[arg.default_alias] = arg
        annotations.update(kwargs)

        clone = self._chain()

        for alias, annotation in annotations.items():
            clone.query.add_annotation(
                annotation,
                alias,
                is_summary=False,
                select=select,
            )

        return clone
```

4-3. Query.add_annotation()

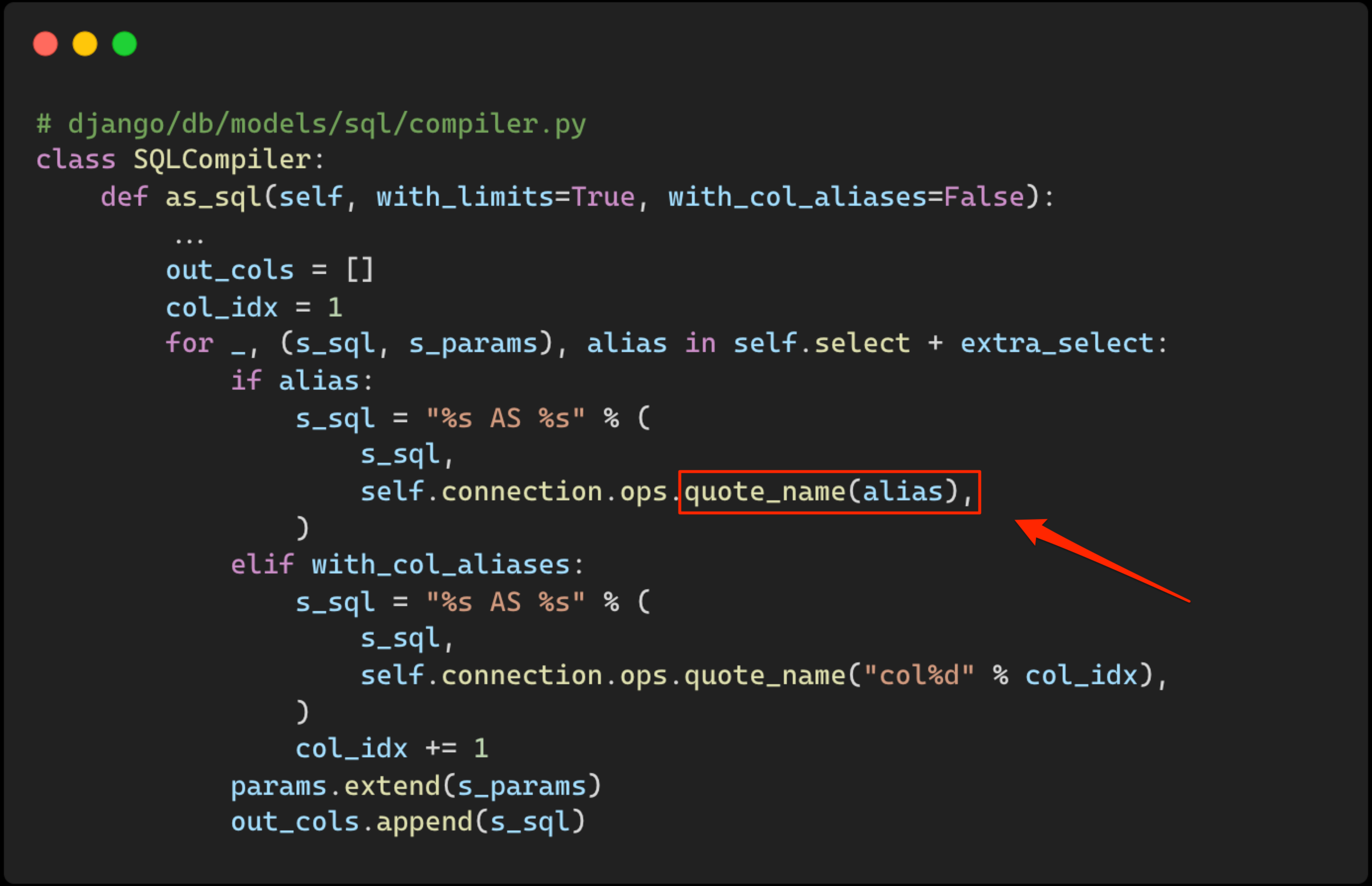
CVE-2022-28346 Step by step!



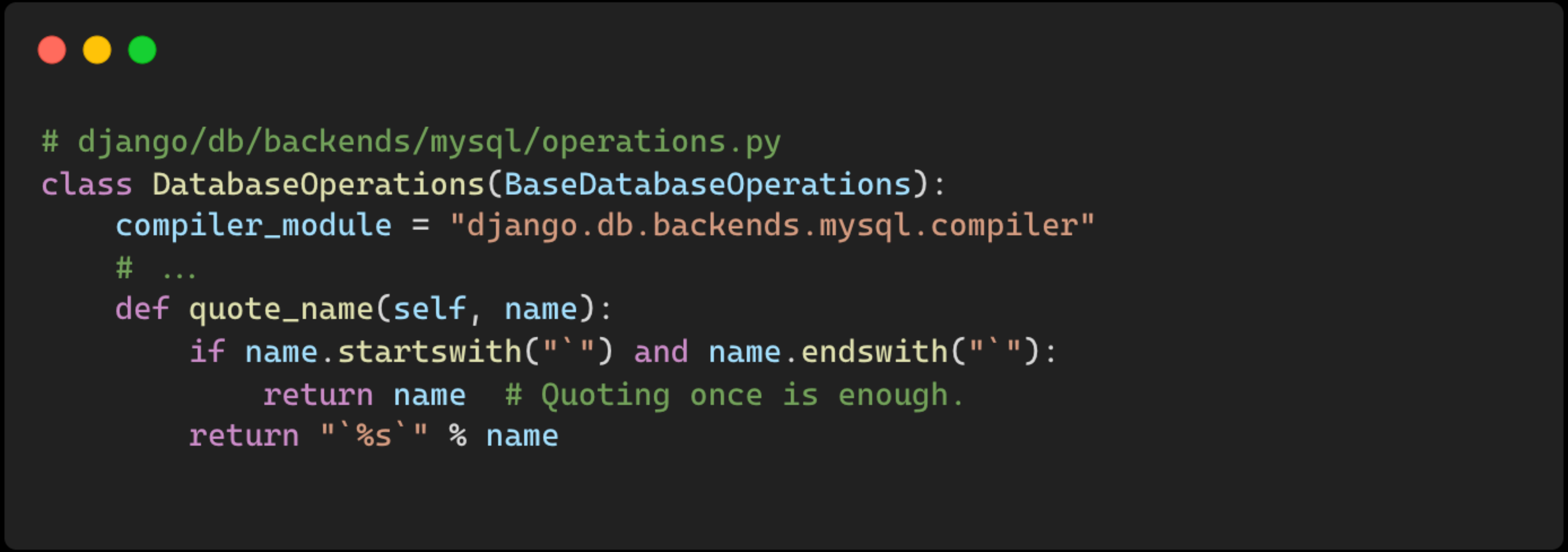
```python
# django/db/models/sql/query.py
class Query(BaseExpression):
    """A single SQL query."""

    def add_annotation(self, annotation, alias, is_summary=False, select=True):
        """Add a single annotation expression to the Query."""

        annotation = annotation.resolve_expression(
            self, allow_joins=True, reuse=None, summarize=is_summary
        )

        self.append_annotation_mask([alias])
        self.annotations[alias] = annotation
```

4-4. `django.db.models.sql.compiler`

CVE-2022-28346 Step by step!



```python
# django/db/models/sql/compiler.py
class SQLCompiler:
    def as_sql(self, with_limits=True, with_col_aliases=False):
        ...
        out_cols = []
        col_idx = 1
        for _, (s_sql, s_params), alias in self.select + extra_select:
            if alias:
                s_sql = "%s AS %s" % (
                    s_sql,
                    self.connection.ops.quote_name(alias),
                )
            elif with_col_aliases:
                s_sql = "%s AS %s" % (
                    s_sql,
                    self.connection.ops.quote_name("col%d" % col_idx),
                )
                col_idx += 1
            params.extend(s_params)
            out_cols.append(s_sql)
```



```python
# django/db/backends/mysql/operations.py
class DatabaseOperations(BaseDatabaseOperations):
    compiler_module = "django.db.backends.mysql.compiler"
    # ...
    def quote_name(self, name):
        if name.startswith("`") and name.endswith("`"):
            return name  # Quoting once is enough.
        return "`%s`" % name
```

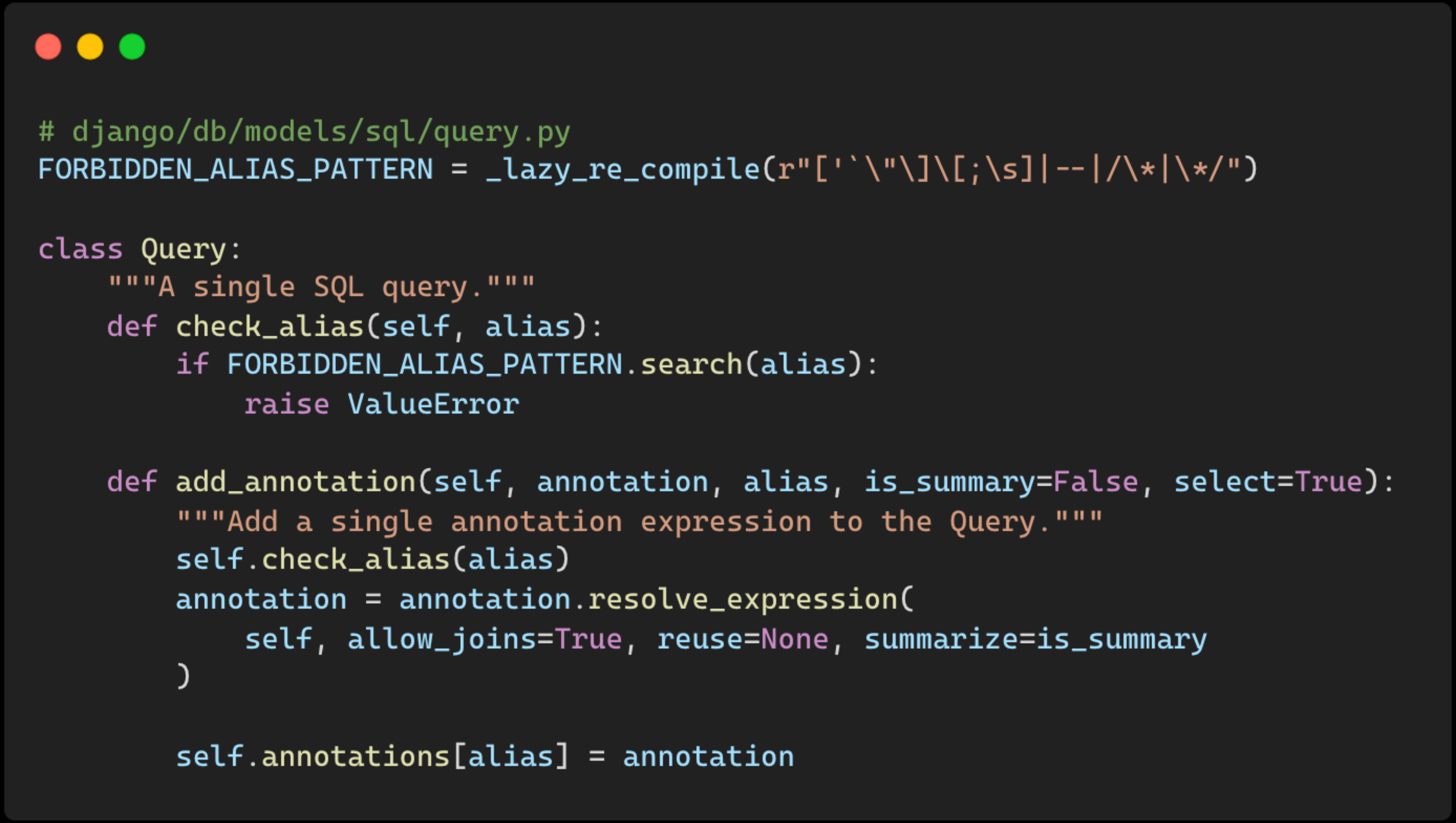
PWNED!

How was it patched?

5-1. `Query.add_annotation()`

How was it patched?

- Added `check_alias()` method in `Query` class
- This method will find malicious characters



```python
# django/db/models/sql/query.py
FORBIDDEN_ALIAS_PATTERN = _lazy_re_compile(r"['`\"\]\[;\s]|--|/\*|\*/")

class Query:
    """A single SQL query."""
    def check_alias(self, alias):
        if FORBIDDEN_ALIAS_PATTERN.search(alias):
            raise ValueError

    def add_annotation(self, annotation, alias, is_summary=False, select=True):
        """Add a single annotation expression to the Query."""
        self.check_alias(alias)
        annotation = annotation.resolve_expression(
            self, allow_joins=True, reuse=None, summarize=is_summary
        )

        self.annotations[alias] = annotation
```

How to report
Your security issue?

6-1. Reporting your own issue

How to report your security issue?

- Visit the official web page for project
  - e.g. Django-Project for Django framework
- Contact developers who manage project
- Use vulnerability managing service like mitre
- Use bug bounty service like huntr.dev, hacker-one

6-2. Remember!

How to report your security issue?

- If your issue is not **EXPLOITABLE**,
  they **MAY NOT ACCEPT** it as a security issue.
- Do not disclose your issues
  UNTIL THEY ARE PATCHED.

Quiz

1. Why do we use Django ORM?

2. ORM 약자?

Conclusion

7-1. Advantages of analyzing open-sourced

Conclusion

- Improves your coding skill
- Expands your developer / hacker connections
- Makes you to gain honor

감사합니다.

QnA (-> @ch4n3.yoon)

윤석찬, 2023.02.12.

POC SECURITY