Loading and Extracting Data with Tables: Takeaways

Syntax

- Using a prepared statement to insert values:

```
conn = psycopg2.connect("dbname=dq user=dq")
cur = conn.cursor()
with open('ign.csv', 'r') as f:
    next(f)
    reader = csv.reader(f)
    for row in reader:
        cur.execute(
            "INSERT INTO ign_reviews VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s)",
            row
        )
    conn.commit()
```

- Returning the exact string that would be sent to the database:

```
cur.mogrify("INSERT INTO test (num, data) VALUES (%s, %s)", (42, 'bar'))
"INSERT INTO test (num, data) VALUES (42, E'bar')"
```

- Copying data between a file and a table:

```
COPY example_table FROM STDIN
```

- Specifying the type of file when copying using copy_expert:

```
conn = psycopg2.connect("dbname=dq user=dq")
cur = conn.cursor()
with open('ign.csv', 'r') as f:
    cur.copy_expert('COPY ign_reviews FROM STDIN WITH CSV HEADER', f)
    conn.commit()
```

- Extracting a Postgres table to any Python file object:

```
cur = conn.cursor()

with open('example.txt', 'w') as f:

    cur.copy_expert('COPY ign_reviews to STDOUT', f)
```

Concepts

- A prepared statements helps performance by signaling what table and and how many values will be altered so that the Postgres engine can be ready for them.
- The `mogrify()` statement returns a `utf-8` character encoded string.
- The difference between `copy_from()` and `copy_expert()` is that `copy_expert` contains a parameter for the file descriptor.
- As table size increases, it requires even more memory and disk space to load and store the files.

Resources

- Formatted SQL with Psycopg's mogrify
- PostGreSQL COPY method