Loading and Extracting Data with Tables: Takeaways

Syntax

- Inserting mogrified values:

  ```
  user = (1, 'alice@dataquest.io', 'Alice', '100, Fake Street')
  mogrified_values = cur.mogrify("(%s, %s, %s, %s)", user)
  cur.execute("INSERT INTO users VALUES" + mogrified_values.decode(conn.encoding) +
  ";")
  ```

- Loading data from a CSV file:

  ```
  with open("file_name.csv", "r") as f:
      cur.copy_expert("COPY table_name FROM STDIN WITH CSV HEADER;", f)
  ```

- Copying data into a CSV file:

  ```
  with open("file_name.txt", "w") as f:
      cur.copy_expert("COPY table_name TO STDOUT;", f)
  ```

- Copying a table with

  ```
  INSERT
  ```

  and

  ```
  SELECT
  ```

  :

  ```
  INSERT INTO users_copy (id, email, name, address) SELECT * FROM users;
  ```

- Extracting data from one table into another:

  ```
  INSERT INTO emails (id, email) SELECT id, email FROM users;
  ```

Concepts

- Data is sent between the client and the server using a system dependent encoding. This encoding can be found by inspecting the `connection.encoding` parameter.
- The `cursor.mogrify()` method is the method that is used internally to safely convert Python types into Postgres types. It returns a `bytes` object encoded with the connection encoding. It needs to be decoded into a string to be used in a query.
- The `copy_expert()` method is more robust for loading a CSV into a table. Using it is also much faster than manually performing inserts.
- As table size increases, it requires even more memory and disk space to load and store the files. Hence, for large table, it is recommended to copy data directly using SQL commands on the Postgres server.

Resources

- Encodings supported by Postgres
- Formatted SQL with Psycopg's mogrify
- Postgres COPY method