Augmenting Pandas With SQLite: Takeaways

Syntax

- Appending rows to a SQLite database table:

```
import sqlite3
import pandas as pd
conn = sqlite3.connect('moma.db')
moma_iter = pd.read_csv('moma.csv', chunksize=1000)
for chunk in moma_iter:
    chunk.to_sql("exhibitions", conn, if_exists='append', index=False)
```

- Using pandas to query a SQLite database:

```
conn = sqlite3.connect('test.db')
df = pd.DataFrame({'A': [0,1,2], 'B': [3,4,5]})
df.to_sql('test', conn)
pd.read_sql('select A from test', conn)
```

Concepts

- Pandas stores and work with data in memory, whereas a database like SQLite can represent data on disk.
- While pandas is limited by the amount of available memory, SQLite is limited only by the amount of available disk space.
- SQLite data types:

Type	Description
NULL	The value is a NULL value
INTEGER	The value is a signed integer, stored in 1, 2, 3, 4, 6, or 8 bytes, depending on the magnitude of the value
REAL	The value is a floating point value, stored as an 8-byte IEEE floating point number
TEXT	The value is a text string, stored using the database encoding (UTF-8, UTF-16BE or UTF-16LE)
BLOB	The value is a blob of data, stored exactly as it was entered

- Selecting the correct types in SQLite reduces the disk footprint of the database file, and can make some SQLite operations faster.
- Generating a pandas dataframe using SQL allows us to do data selection with SQL, but the iterative exploration and analysis using pandas.
- Pandas has several advantages over SQLite such as:
    - Pandas has a large suite of functions and methods for performing common operations.
    - Pandas has a diverse type system we can use to save space and improve code running speed.
    - Pandas works in memory and will be quicker for most tasks.

- Querying data in SQL and working with batches of the results set will help you get the most out of SQL and pandas.

Resources

- [SQLite Datatypes](#)
- [Limits in SQLite](#)