Elements Of Data Science - F2020

Week 13: Databases and Review

12/14/2020

TODOs

- No Quiz this week
- HW4, **Due Friday Dec 18th 11:59pm ET**
- Final
    - Release Monday night 12/14
    - Due **Saturday Dec 19th, 11:59pm ET**
    - Have 24hrs after starting exam to finish
    - 30-40 questions (fill in the blank/multiple choice/short answer)
    - Online via Gradescope
    - Cumulative, but focus on material from after the midterm
    - Open-book, open-note, open-python
    - Questions asked/answered **privately** via Piazza

Today

- Connecting to databases with sqlalchemy and pandas
- Review for the final

Questions re Logistics?

Accessing Databases with Python

- databases vs flat-files
- Relational Databases and SQL
- NoSQL databases

Flat Files

Company Details

E_ID	Name	Department	Dept_ID	Manager_Name
101	Anoop	Accounts	AC-10	Mr Gagan Thakral
201	Anurag	Accounts	AC-10	Mr Gagan Thakral
301	Rakesh	Accounts	AC-10	Mr Gagan Thakral
401	Saurav	Accounts	AC-10	Mr Gagan Thakral

- eg: csv, json, etc
- Pros
    - Ease of access
    - Simple to transport
- Cons
    - May include redundant information
    - Slow to search
    - No integity checks

Relational Databases

- Data stored in **tables** (rows/columns)
- Table columns have well defined datatype requirements
- Complex **indexes** can be set up over often used data/searches
- Row level security, separate from the operating system

- Related data is stored in separate tables, referenced by **keys**

- Many commonly used Relational Databases

    - sqlite (small footprint db, might already have it installed)
    - Mysql
    - PostgreSQL
    - Microsoft SQL Server
    - Oracle

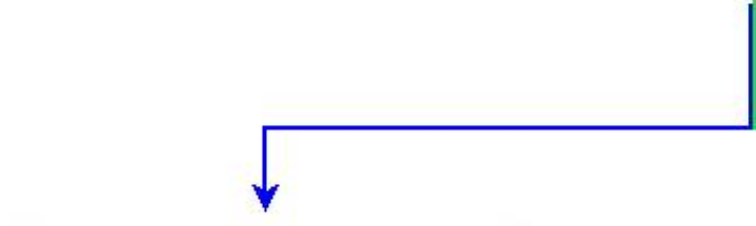
Database Normalization

- Organize data in accordance with **normal forms**
- Rules designed to:
    - reduce data redundancy
    - improve data integrity
- Rules like:
    - Has Primary Key
    - No repeating groups
    - Cells have single values
    - No partial dependencies on keys (use whole key)
    - ...

Database Normalization

Company Details

E_ID	Name	Department	Dept_ID	Manager_Name
101	**Anoop**	**Accounts**	**AC-10**	**Mr Gagan Thakral**
201	**Anurag**	**Accounts**	**AC-10**	**Mr Gagan Thakral**
301	**Rakesh**	**Accounts**	**AC-10**	**Mr Gagan Thakral**
401	**Saurav**	**Accounts**	**AC-10**	**Mr Gagan Thakral**



E_ID	Name	Dept_ID
101	**Anoop**	**AC-10**
201	**Anurag**	**AC-10**
301	**Rakesh**	**AC-10**
401	**Saurav**	**AC-10**

Employee_Details

Dept_ID	Department	Manager_Name
A-10	**Accounts**	**Mr Gagan Thakral**

Department_Details

From https://www.minigranth.com/dbms-tutorial/database-normalization-dbms/ (https://www.minigranth.com/dbms-tutorial/database-normalization-dbms/)

De-Normalization

- But we want a single table/dataframe!
- Very often need to **denormalize**
- .. using joins! (see more later)

Structured Query Language (SQL)

- (Semi) standard language for querying, transforming and returning data
- Notable characteristics:
    - generally case independent
    - white-space is ignored
    - strings denoted with single quotes
    - comments start with double-dash "--"

```
SELECT
    client_id
    ,lastname
FROM
    company_db.bi.clients --usually database.schema.table
WHERE
    lastname LIKE 'Gi%'   --only include rows with lastname starting with Gi
LIMIT 10
```

Small but Powerful DB: SQLite3

- likely already have it installed
- many programs use it to store configurations, history, etc
- good place to play around with sql

```
bgibson@civet:~$ sqlite3
SQLite version 3.22.0 2018-01-22 18:45:57
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite>
```

Accessing Relational DBs: `sqlalchemy`

- flexible library for accessing a variety of sql dbs
- can use to query through pandas itself to retrieve a dataframe

In [60]:
```
import sqlalchemy

# sqlite sqlalchemy relative path syntax: 'sqlite:///[path to database file]'
engine = sqlalchemy.create_engine('sqlite:///../data/example_business.sqlite')

# read all records from the table sales
sql = """
SELECT
    *
FROM
    clients
"""

pd.read_sql(sql,engine)
```

Out[60]:

	client_id	firstname	lastname	home_address_id
0	102	Mikel	Rouse	1002
1	103	Laura	Gibson	1003
2	104	None	Reeves	1003
3	105	Scott	Payseur	1004

SQL: SELECT

In [61]:
```
sql="""
SELECT
    client_id
    ,lastname
FROM
    clients
"""

pd.read_sql(sql,engine)
```

Out[61]:

	client_id	lastname
0	102	Rouse
1	103	Gibson
2	104	Reeves
3	105	Payseur

SQL: AS alias

In [62]:
```
sql="""
SELECT
    client_id AS Cid
    ,lastname AS Lastname
FROM
    clients ca
"""

pd.read_sql(sql,engine)
```

Out[62]:

	Cid	Lastname
0	102	Rouse
1	103	Gibson
2	104	Reeves
3	105	Payseur

SQL: * (wildcard)

In [63]:
```
sql="""
SELECT
    *
FROM
    clients
"""
clients = pd.read_sql(sql,engine)
clients
```

Out[63]:

	client_id	firstname	lastname	home_address_id
0	102	Mikel	Rouse	1002
1	103	Laura	Gibson	1003
2	104	None	Reeves	1003
3	105	Scott	Payseur	1004

In [64]:
```
sql="""
SELECT
    *
FROM
    addresses
"""
addresses = pd.read_sql(sql,engine)
addresses
```

Out[64]:

	address_id	address
0	1002	1 First Ave.
1	1003	2 Second Ave.
2	1005	3 Third Ave.

SQL: WHERE

In [65]:
```
sql = """
SELECT
    *
FROM
    clients
WHERE home_address_id = 1003
"""

pd.read_sql(sql,engine)
```

Out[65]:

	client_id	firstname	lastname	home_address_id
0	103	Laura	Gibson	1003
1	104	None	Reeves	1003

In [66]:
```
sql = """
SELECT
    *
FROM
    clients
WHERE home_address_id = 1003 AND lastname LIKE 'Gi%'
"""

pd.read_sql(sql,engine)
```

Out[66]:

	client_id	firstname	lastname	home_address_id
0	103	Laura	Gibson	1003

SQL: (INNER) JOIN

In [67]:
```
sql="""
SELECT
    c.firstname
    ,a.address
FROM clients AS c
JOIN addresses AS a ON c.home_address_id = a.address_id
WHERE c.firstname IS NOT NULL
"""

pd.read_sql(sql,engine)
```

Out[67]:

	firstname	address
0	Mikel	1 First Ave.
1	Laura	2 Second Ave.

SQL: LEFT JOIN

In [68]:
```
sql="""
SELECT
    c.firstname,a.address
FROM clients AS c
LEFT JOIN addresses AS a ON c.home_address_id = a.address_id
WHERE c.firstname IS NOT NULL
"""

pd.read_sql(sql,engine)
```

Out[68]:

	firstname	address
0	Mikel	1 First Ave.
1	Laura	2 Second Ave.
2	Scott	None

SQL: RIGHT JOIN

In [69]:
```
# this will cause an error in pandas, right join not supported in sqlalchemy + s
qlite3
sql="""
SELECT
    c.firstname,a.address
FROM clients AS c
RIGHT JOIN addresses AS a ON c.home_address_id = a.address_id
"""
#pd.read_sql(sql,engine)
```

In [70]:
```
sql="""
SELECT
    c.firstname,a.address
from addresses a
LEFT JOIN clients AS c ON c.home_address_id = a.address_id
"""
pd.read_sql(sql,engine)
```

Out[70]:

	firstname	address
0	Mikel	1 First Ave.
1	None	2 Second Ave.
2	Laura	2 Second Ave.
3	None	3 Third Ave.

In [71]:
```
pd.merge(clients,addresses,left_on='home_address_id',right_on='address_id',how=
'right')[['firstname','address']]
```

Out[71]:

	firstname	address
0	Mikel	1 First Ave.
1	Laura	2 Second Ave.
2	None	2 Second Ave.
3	NaN	3 Third Ave.

SQL: FULL OUTER JOIN

In [72]:
```
# this will cause an error in pandas, outer join not supported in sqlalchemy + s
qlite3
sql="""
SELECT
    c.firstname,a.address
FROM clients AS c
OUTER JOIN addresses AS a ON c.home_address_id = a.address_id
"""
#pd.read_sql(sql,engine)
```

In [73]:
```
pd.merge(clients,addresses,left_on='home_address_id',right_on='address_id',how=
'outer')[['firstname','address']]
```

Out[73]:

	firstname	address
0	Mikel	1 First Ave.
1	Laura	2 Second Ave.
2	None	2 Second Ave.
3	Scott	NaN
4	NaN	3 Third Ave.

SQL: And Much More!

- Multiple Joins
- DISTINCT
- COUNT
- ORDER BY
- GROUP BY
- LIMIT
- Operators (string concatenate operator is '||' in sqlite)
- Subqueries
- HAVING
- see Data Science From Scratch Ch. 23 (https://ezproxy.cul.columbia.edu/login?qurl=https%3a%2f%2fsearch.ebscohost.com%2flogin.aspx%3fdirect%3dtrue%26dlive%26scope%3dsite%26ebv%3DEB%26ppid%3Dpp_275)

NoSQL

- Anything that isn't traditional SQL/RDBMS
    - key-value (Redis,Berkely DB)
    - document store (MongoDB, DocumentDB)
    - wide column (Cassandra, HBase, DynamoDB)
    - graph (Neo4j)
- Rapidly growing field to fit needs
- Probably more as we speak

Example: Mongo

- records represented as documents (think json)
- very flexible structure
- great way to store semi-structure data

- a lot of processing needed to turn into feature vectors

- contains databases (db)

    - which contain collections (like tables)
    - which you then do finds on

Example: Mongo

- Need to have Mongo running on your local machine with a 'twitter_db' database

In [74]:
```
# conda install -n eods-f20 pymongo
import pymongo

# start up our client, defaults to the local machine
mdb = pymongo.MongoClient()

# get a connection to a database
db = mdb.twitter_db

# get a connection to a collection in that database
coll = db.twitter_collection
```

Example: Mongo

In [75]:

```
# get one record
coll.find_one()

example_output = """
{'_id': ObjectId('59c95e2c2471847a9783c400'),
 'created_at': 'Mon Sep 25 19:51:08 +0000 2017',
 'id': 912404120484511749,
 'id_str': '912404120484511749',
 'text': 'RT @YarmolukDan: Waste Management Just Got Cleaner and More Efficient
 https://t.co/HtaXzfxbrA #DataScience #DataScientist #BigData #AI #IoT…',
 'source': '<a href="http://twitter.com/download/android" rel="nofollow">Twitter
for Android</a>',
 'truncated': False,
 'in_reply_to_status_id': None,
 'in_reply_to_status_id_str': None,
 'in_reply_to_user_id': None,
 'in_reply_to_user_id_str': None,
 'in_reply_to_screen_name': None,
 'user': {'id': 912391257430794241,
  'id_str': '912391257430794241',
  'name': 'Roxane Wattenbarger',
  'screen_name': 'roxanewattenba6',
  'location': None,
  'url': None,
  'description': 'l',
  'translator_type': 'none',
  ...'
}
"""
```

Questions re Databases?

Final Review