Notebook

July 2, 2025

1 Problem 0

[1]:
```
import numpy as np
import pandas as pd
import requests
import json
import sqlite3
import psycopg
import mysql.connector
from sqlalchemy import create_engine
import pymongo
from bson.json_util import loads, dumps
import dotenv
import os
```

2 Problem 1

Part a

There are 3 containers being launched.

Postgres - runs a postgre relational database server

MySQL - runs a MYSQL relational database server

MongoDB - runs a MongoDB NoSQL database server

The docker images are stored on Docker Hub.

Part b

Client: Version: 28.2.2 API version: 1.50 Go version: go1.24.3 Git commit: e6534b4 Built: Fri May 30 12:07:16 2025 OS/Arch: windows/amd64 Context: desktop-linux

Server: Docker Desktop 4.42.1 (196648) Engine: Version: 28.2.2 API version: 1.50 (minimum version 1.24) Go version: go1.24.3 Git commit: 45873be Built: Fri May 30 12:07:26 2025 OS/Arch: linux/amd64 Experimental: false containerd: Version: 1.7.27 GitCommit: 05044ec0a9a75232cad458027ca83437aae3f4da runc: Version: 1.2.5 GitCommit: v1.2.5-0-g59923ef docker-init: Version: 0.19.0 GitCommit: de40ad0

Part c

[2]:
```
dotenv.load_dotenv()
```

[2]:
```
True
```

Part d

[+] Running 7/7 Network ds6001databases_default Created 0.1s Volume "ds6001databases_postgresdata" Created 0.1s Volume "ds6001databases_mongodata" Created 0.0s Volume "ds6001databases_mysqldata" Created 0.0s Container ds6001databases-mysql-1 Created 1.0s

Part e

[ ]:
```
# 1. Load .env file
dotenv.load_dotenv()

POSTGRES_PASSWORD = os.getenv('POSTGRES_PASSWORD')
MONGO_INITDB_ROOT_USERNAME = os.getenv('MONGO_INITDB_ROOT_USERNAME')
MONGO_INITDB_ROOT_PASSWORD = os.getenv('MONGO_INITDB_ROOT_PASSWORD')
MYSQL_ROOT_PASSWORD = os.getenv('MYSQL_ROOT_PASSWORD')

# 2. Test MySQL
try:
    mysql_db = mysql.connector.connect(
        user='root',
        password=MYSQL_ROOT_PASSWORD,
        host='localhost',
        port='3306'
    )
    print(" MySQL connection successful!")
    mysql_db.close()
except Exception as e:
    print(" MySQL connection failed:", e)

# 3. Test PostgreSQL
try:
    postgres_db = psycopg.connect(
        user='postgres',
        password=POSTGRES_PASSWORD,
        host='localhost',
        port='5432'
    )
    postgres_db.autocommit = True
    print("PostgreSQL connection successful!")
    postgres_db.close()
except Exception as e:
    print("PostgreSQL connection failed:", e)

# 4. Test MongoDB connection
```

```
try:
    mongo_client = pymongo.MongoClient(
        f"mongodb://{MONGO_INITDB_ROOT_USERNAME}:
 ↪{MONGO_INITDB_ROOT_PASSWORD}@localhost:27017/"
    )
    print("MongoDB connection successful!")
    print(mongo_client.list_database_names())
    mongo_client.close()
except Exception as e:
    print("MongoDB connection failed:", e)


#Was having trouble with given code, so I decided to modify it
```

```
 MySQL connection successful!
PostgreSQL connection successful!
MongoDB connection successful!
['admin', 'config', 'local']
```

3 Problem 2

Part a

The problem the exists in the original data is that there is redundancy shown in the patient's name, date of birth, and insurance across prescriptions.

The data could be reorganized to a single table to conform to 1st normal form. One table could be called patient_prescriptions with this primary key: (patient_name, date_of_birth, prescribed_drug).

Part b

2nd normal form requires partial dependency. The problems that exist are that insurance depends only on patient_name and date_of_birth; hospital and hospital_location depend on prescribing_physician, and drug_dosage depends only on prescribed_drug.

The data could be reorganized, having 5 tables: 1. Patient table : patient_id (patient_name + date_of_birth) (primary key) patient_name date_of_birth insurance

2. Physcians table: physician_id (priamry key) prescribing_physician hopital_location
3. Hospitals table: hospital_name (primary key) hopital_location
4. Drugs table: drugs_name (primary key) drug_dosage
5. Prescriptions table: patient_id (primary key) physcian_id (primary key) drug_name (primary key) prescription_date

Part c

3rd normal form requires transitive dependency. The problem is that in the Physicians table, hospital_location depends on hospital, not directly on physician_id, thus there is no transitive dependency.

To fix this, hospital_name should also be under the physician table, and hospital_location should only be under hospital.

1. Patient table : patient_id (patient_name + date_of_birth) (primary key) patient_name date_of_birth insurance
2. Physcians table: physician_id (priamry key) prescribing_physician hospital_name
3. Hospitals table: hospital_name (primary key) hopital_location
4. Drugs table: drugs_name (primary key) drug_dosage
5. Prescriptions table: patient_id (primary key) physcian_id (primary key) drug_name (primary key) prescription_date

Part d

[24]:
```
df_1 = pd.read_csv("patients.csv")
df_2 = pd.read_csv("physicians.csv")
df_3 = pd.read_csv("hospitals.csv")
df_4 = pd.read_csv("prescriptions.csv")
df_5 = pd.read_csv("drugs.csv")
```

[25]:
```
df_1
```

[25]:

	patient_id	patient_name	date_of_birth	insurance
0	P1	John Smith	1980-02-14	Aetna
1	P2	Maria Lopez	1975-06-21	Blue Cross
2	P3	Chen Wei	1990-09-09	United Healthcare
3	P4	Fatima Khan	1988-11-03	Cigna
4	P5	Carlos Ruiz	1972-05-22	Aetna

[26]:
```
df_2
```

[26]:

	physician_id	prescribing_physician	hospital_name
0	D1	Dr. James Miller	City Hospital
1	D2	Dr. Olivia Brown	Regional Clinic
2	D3	Dr. Michael Johnson	Mercy Hospital

[27]:
```
df_3
```

[27]:

	hospital_name	hospital_location
0	City Hospital	New York, NY
1	Regional Clinic	San Francisco, CA
2	Mercy Hospital	Chicago, IL

[28]:
```
df_4
```

[28]:

	prescription_id	patient_id	physician_id	drug_name	prescription_date
0	RX1	P1	D1	Lipitor	2024-01-15
1	RX2	P1	D2	Metformin	2024-03-10

```
2             RX3         P2           D3  Amoxicillin        2024-04-01
3             RX4         P3           D2      Lipitor        2024-02-20
4             RX5         P4           D1    Metformin        2024-02-25
```

[29]:
```
df_5
```

[29]:
```
     drug_name            drug_dosage
0      Lipitor             10mg daily
1  Amoxicillin  500mg three times/day
2    Metformin     1000mg twice daily
```

4 Problem 3

[31]:
```
repo = 'https://github.com/jkropko/DS-6001/raw/master/localdata/'
works = pd.read_csv(repo + 'Works.csv')
characters = pd.read_csv(repo + 'Characters.csv')
chapters = pd.read_csv(repo + 'Chapters.csv')
paragraphs = pd.read_csv(repo + 'Paragraphs.csv')
# convert column names to lowercase (needed for postgreSQL to work properly)
characters.columns = characters.columns.str.lower()
chapters.columns = chapters.columns.str.lower()
paragraphs.columns = paragraphs.columns.str.lower()
works.columns = works.columns.str.lower()
# works in the characters tables is a comma separated list.
# Break it out into multiple rows in a new table
charworks = characters[['charid', 'works']]
charworks.loc[:,'works'] = charworks['works'].str.split(',')
charworks = charworks.explode('works')
charworks = charworks.rename({'works':'workid'}, axis=1)
characters = characters.drop('works', axis=1)
#Remove empty rows
chapters = chapters.query("~chapterid.isnull()")
paragraphs = paragraphs.query("~paragraphid.isnull()")
charworks = charworks.query("~workid.isnull()")
# Add chapterid to paragraphs
paragraphs = pd.merge(paragraphs,
chapters.drop('description', axis=1),
how='inner',
on=['workid', 'section', 'chapter'])
#Remove unnecessary columns
paragraphs = paragraphs.drop(['paragraphtype', 'section', 'chapter'],
axis=1)
```

Part a, problem i

charworks and works would join onworkid. Their relationship would be that there would be many charworks entries per works (many-to-one). The reason would be that each character-work pair links to one work, but one work has many character-work pairs.

DBML code:

Ref: charworks.workid > works.workid

Part a, problem ii

characters and charworks join on charid. Their relationship is that there are many charworks entries per characters (many-to-one). The reason would be that a character can appear in multiple works.

DBML code:

Ref: charworks.charid > characters.charid

Part a, problem iii

works and paragraphs would join on workid. Their relationship is that there are many paragraphs per work (many-to-one). The reason is that each work contains many paragraphs.

DBML code:

Ref: paragraphs.workid > works.workid

Part a, problem iv

chapters and works join on workid. Their relationship is that there are many chapters per work (many-to-one). The reason would be that each work contains multiple chapters (acts/scenes).

DBML code:

Ref: chapters.workid > works.workid

Part a, problem v

paragraphs and chapters join on chapterid. Their relationship is that there are many paragraphs per chapter (many-to-one). The reason would be that a chapter (scene) contains many lines (paragraphs).

DBML code:

Ref: paragraphs.chapterid > chapters.chapterid

Part a, problem vi

characters and paragraphs join on charid. Their relationship is that there are many paragraphs per character (many-to-one).

The reason would be that each character speaks multiple lines (paragraphs).

DBML code:

Ref: paragraphs.charid > characters.charid

Part b

website: https://dbdocs.io/palmersaevon/Shakespeare-DB?view=relationships

DBML code:

Table works { workid varchar [pk] title varchar longtitle varchar date int genretype varchar notes varchar source varchar totalwords int totalparagraphs int }

Table characters { charid varchar [pk] charname varchar abbrev varchar description varchar speechcount int }

Table chapters { chapterid varchar [pk] workid varchar section varchar chapter varchar description varchar }

Table paragraphs { paragraphid varchar [pk] workid varchar paragraphnum int charid varchar plaintext varchar phonetictext varchar stemtext varchar charcount int wordcount int chapterid varchar }

Table charworks { charid varchar [pk] workid varchar [pk] }

Ref: charworks.workid > works.workid Ref: charworks.charid > characters.charid Ref: paragraphs.workid > works.workid Ref: chapters.workid > works.workid Ref: paragraphs.chapterid > chapters.chapterid Ref: paragraphs.charid > characters.charid

5 Problem 4

Part a

[33]:
```
# Create SQLite DB
engine = create_engine('sqlite:///shakespeare.sqlite')

# Write to DB
works.to_sql('works', con=engine, if_exists = 'replace')
characters.to_sql('characters', con = engine, if_exists = 'replace')
chapters.to_sql('chapters', con = engine, if_exists = 'replace')
paragraphs.to_sql('paragraphs', con = engine, if_exists = 'replace')
charworks.to_sql('charworks', con = engine, if_exists = 'replace')

# Query
query = """
SELECT charname, description, speechcount
FROM characters
WHERE speechcount > 200
"""
result = pd.read_sql_query(query, con=engine)
print(result)

engine.dispose()
```

```
              charname                                        description  \
0                Antony                                (Marcus Antonius)
1             Cleopatra                                   queen of Egypt
2              Falstaff                                Sir John Falstaff
3   Duke of Gloucester                               brother to the King
4                Hamlet  son of the former king and nephew to the prese…
5               Henry V                          Prince, King of England
6                  Iago                            Othello's ancient (?)
7               Othello   A noble Moor in the service of the Ventian state
```

```
8                 Poet                    the voice of Shakespeare's poetry
9          Richard III  son of Richard Plantagenet, duke of York; was …
10            Rosalind                        daughter to the banished Duke
11               Timon                                                  None

    speechcount
0         253.0
1         204.0
2         471.0
3         285.0
4         358.0
5         377.0
6         272.0
7         274.0
8         733.0
9         246.0
10        201.0
11        210.0
```

Part b

[34]:
```
dotenv.load_dotenv()
MYSQL_ROOT_PASSWORD = os.getenv('MYSQL_ROOT_PASSWORD')

# Connect to server
dbserver = mysql.connector.connect(
    user='root',
    password=MYSQL_ROOT_PASSWORD,
    host='localhost',
    port='3306'
)
cursor = dbserver.cursor()
cursor.execute("DROP DATABASE IF EXISTS shakespeare")
cursor.execute("CREATE DATABASE shakespeare")
cursor.close()
dbserver.close()

# Write data
engine = create_engine(f"mysql+mysqlconnector://root:
 ↪{MYSQL_ROOT_PASSWORD}@localhost/shakespeare")
works.to_sql('works', con=engine, if_exists='replace')
characters.to_sql('characters', con=engine, if_exists='replace')
chapters.to_sql('chapters', con=engine, if_exists='replace')
paragraphs.to_sql('paragraphs', con=engine, if_exists='replace')
charworks.to_sql('charworks', con=engine, if_exists='replace')

# Query
query = """
```

```
SELECT charname, description, speechcount
FROM characters
WHERE speechcount > 200
"""
result = pd.read_sql_query(query, con=engine)
print(result)

engine.dispose()
```

```
              charname                                        description  \
0               Antony                                  (Marcus Antonius)
1            Cleopatra                                     queen of Egypt
2             Falstaff                                  Sir John Falstaff
3   Duke of Gloucester                                brother to the King
4               Hamlet  son of the former king and nephew to the prese…
5              Henry V                           Prince, King of England
6                 Iago                             Othello's ancient (?)
7              Othello   A noble Moor in the service of the Ventian state
8                 Poet                 the voice of Shakespeare's poetry
9          Richard III  son of Richard Plantagenet, duke of York; was …
10            Rosalind                      daughter to the banished Duke
11              Timon                                               None

    speechcount
0         253.0
1         204.0
2         471.0
3         285.0
4         358.0
5         377.0
6         272.0
7         274.0
8         733.0
9         246.0
10        201.0
11        210.0
```

Part c

[35]:
```
dotenv.load_dotenv()
POSTGRES_PASSWORD = os.getenv('POSTGRES_PASSWORD')

# Connect to server
conn = psycopg.connect(
    user="postgres",
    password=POSTGRES_PASSWORD,
    host="localhost",
    port="5432",
```

```
    autocommit=True
)
cur = conn.cursor()
cur.execute('DROP DATABASE IF EXISTS shakespeare')
cur.execute('CREATE DATABASE shakespeare')
cur.close()
conn.close()

# Write data
engine = create_engine(f"postgresql+psycopg://postgres:
  ↪{POSTGRES_PASSWORD}@localhost:5432/shakespeare")
works.to_sql('works', con=engine, if_exists='replace')
characters.to_sql('characters', con=engine, if_exists='replace')
chapters.to_sql('chapters', con=engine, if_exists='replace')
paragraphs.to_sql('paragraphs', con=engine, if_exists='replace')
charworks.to_sql('charworks', con=engine, if_exists='replace')

# Query
query = """
SELECT charname, description, speechcount
FROM characters
WHERE speechcount > 200
"""
result = pd.read_sql_query(query, con=engine)
print(result)

engine.dispose()
```

```
             charname                                        description  \
0               Antony                                  (Marcus Antonius)
1            Cleopatra                                     queen of Egypt
2             Falstaff                                  Sir John Falstaff
3   Duke of Gloucester                                brother to the King
4               Hamlet  son of the former king and nephew to the prese…
5              Henry V                          Prince, King of England
6                 Iago                            Othello's ancient (?)
7              Othello   A noble Moor in the service of the Ventian state
8                 Poet                 the voice of Shakespeare's poetry
9          Richard III  son of Richard Plantagenet, duke of York; was …
10            Rosalind                     daughter to the banished Duke
11               Timon                                              None

    speechcount
0         253.0
1         204.0
2         471.0
3         285.0
4         358.0
```

```
5          377.0
6          272.0
7          274.0
8          733.0
9          246.0
10         201.0
11         210.0
```

6 Problem 5

Part a

[36]:
```
dotenv.load_dotenv()
MONGO_INITDB_ROOT_USERNAME = os.getenv('MONGO_INITDB_ROOT_USERNAME')
MONGO_INITDB_ROOT_PASSWORD = os.getenv('MONGO_INITDB_ROOT_PASSWORD')

# Connect to MongoDB
client = pymongo.MongoClient(f"mongodb://{MONGO_INITDB_ROOT_USERNAME}:
 ↪{MONGO_INITDB_ROOT_PASSWORD}@localhost:27017/")

# Create DB and Collection
db = client["history"]
db.drop_collection("today")  # Clean previous runs
today = db["today"]

# Get API data
history = requests.get("https://history.muffinlabs.com/date")
history_json = json.loads(history.text)
events = history_json['data']['Events']
```

Part b

[37]:
```
today.insert_many(events)
```

[37]:
```
InsertManyResult([ObjectId('68649c4670d6ace56d5ae9bd'),
ObjectId('68649c4670d6ace56d5ae9be'), ObjectId('68649c4670d6ace56d5ae9bf'),
ObjectId('68649c4670d6ace56d5ae9c0'), ObjectId('68649c4670d6ace56d5ae9c1'),
ObjectId('68649c4670d6ace56d5ae9c2'), ObjectId('68649c4670d6ace56d5ae9c3'),
ObjectId('68649c4670d6ace56d5ae9c4'), ObjectId('68649c4670d6ace56d5ae9c5'),
ObjectId('68649c4670d6ace56d5ae9c6'), ObjectId('68649c4670d6ace56d5ae9c7'),
ObjectId('68649c4670d6ace56d5ae9c8'), ObjectId('68649c4670d6ace56d5ae9c9'),
ObjectId('68649c4670d6ace56d5ae9ca'), ObjectId('68649c4670d6ace56d5ae9cb'),
ObjectId('68649c4670d6ace56d5ae9cc'), ObjectId('68649c4670d6ace56d5ae9cd'),
ObjectId('68649c4670d6ace56d5ae9ce'), ObjectId('68649c4670d6ace56d5ae9cf'),
ObjectId('68649c4670d6ace56d5ae9d0'), ObjectId('68649c4670d6ace56d5ae9d1'),
ObjectId('68649c4670d6ace56d5ae9d2'), ObjectId('68649c4670d6ace56d5ae9d3'),
ObjectId('68649c4670d6ace56d5ae9d4'), ObjectId('68649c4670d6ace56d5ae9d5'),
ObjectId('68649c4670d6ace56d5ae9d6'), ObjectId('68649c4670d6ace56d5ae9d7'),
```

```
ObjectId('68649c4670d6ace56d5ae9d8'), ObjectId('68649c4670d6ace56d5ae9d9'),
ObjectId('68649c4670d6ace56d5ae9da'), ObjectId('68649c4670d6ace56d5ae9db'),
ObjectId('68649c4670d6ace56d5ae9dc'), ObjectId('68649c4670d6ace56d5ae9dd'),
ObjectId('68649c4670d6ace56d5ae9de'), ObjectId('68649c4670d6ace56d5ae9df'),
ObjectId('68649c4670d6ace56d5ae9e0'), ObjectId('68649c4670d6ace56d5ae9e1'),
ObjectId('68649c4670d6ace56d5ae9e2'), ObjectId('68649c4670d6ace56d5ae9e3'),
ObjectId('68649c4670d6ace56d5ae9e4'), ObjectId('68649c4670d6ace56d5ae9e5'),
ObjectId('68649c4670d6ace56d5ae9e6'), ObjectId('68649c4670d6ace56d5ae9e7'),
ObjectId('68649c4670d6ace56d5ae9e8'), ObjectId('68649c4670d6ace56d5ae9e9'),
ObjectId('68649c4670d6ace56d5ae9ea'), ObjectId('68649c4670d6ace56d5ae9eb'),
ObjectId('68649c4670d6ace56d5ae9ec'), ObjectId('68649c4670d6ace56d5ae9ed'),
ObjectId('68649c4670d6ace56d5ae9ee'), ObjectId('68649c4670d6ace56d5ae9ef'),
ObjectId('68649c4670d6ace56d5ae9f0'), ObjectId('68649c4670d6ace56d5ae9f1'),
ObjectId('68649c4670d6ace56d5ae9f2'), ObjectId('68649c4670d6ace56d5ae9f3'),
ObjectId('68649c4670d6ace56d5ae9f4')], acknowledged=True)
```

Part c

[39]:
```python
query = {"text": {"$regex": "China"}}
results = list(today.find(query))

print(f"Number of documents containing 'China': {len(results)}")
for doc in results:
    print(doc)

# Convert to JSON formatted string
json_output = json.dumps(results, indent = 4, default = str)
print(json_output)
```

```
Number of documents containing 'China': 1
{'_id': ObjectId('68649c4670d6ace56d5ae9bf'), 'year': '706', 'text': "In China,
Emperor Zhongzong of Tang inters the bodies of relatives in the Qianling
Mausoleum, located on Mount Liang outside Chang'an.", 'html': '706 - In <a
href="https://wikipedia.org/wiki/China" title="China">China</a>, <a
href="https://wikipedia.org/wiki/Emperor_Zhongzong_of_Tang" title="Emperor
Zhongzong of Tang">Emperor Zhongzong of Tang</a> inters the bodies of relatives
in the <a href="https://wikipedia.org/wiki/Qianling_Mausoleum" title="Qianling
Mausoleum">Qianling Mausoleum</a>, located on <a
href="https://wikipedia.org/wiki/Mount_Liang" title="Mount Liang">Mount
Liang</a> outside <a href="https://wikipedia.org/wiki/Chang%27an"
title="Chang\'an">Chang\'an</a>.', 'no_year_html': 'In <a
href="https://wikipedia.org/wiki/China" title="China">China</a>, <a
href="https://wikipedia.org/wiki/Emperor_Zhongzong_of_Tang" title="Emperor
Zhongzong of Tang">Emperor Zhongzong of Tang</a> inters the bodies of relatives
in the <a href="https://wikipedia.org/wiki/Qianling_Mausoleum" title="Qianling
Mausoleum">Qianling Mausoleum</a>, located on <a
href="https://wikipedia.org/wiki/Mount_Liang" title="Mount Liang">Mount
Liang</a> outside <a href="https://wikipedia.org/wiki/Chang%27an"
```

title="Chang\'an">Chang\'an.', 'links': [{'title': 'China', 'link':
'https://wikipedia.org/wiki/China'}, {'title': 'Emperor Zhongzong of Tang',
'link': 'https://wikipedia.org/wiki/Emperor_Zhongzong_of_Tang'}, {'title':
'Qianling Mausoleum', 'link': 'https://wikipedia.org/wiki/Qianling_Mausoleum'},
{'title': 'Mount Liang', 'link': 'https://wikipedia.org/wiki/Mount_Liang'},
{'title': "Chang'an", 'link': 'https://wikipedia.org/wiki/Chang%27an'}]}
[
    {
        "_id": "68649c4670d6ace56d5ae9bf",
        "year": "706",
        "text": "In China, Emperor Zhongzong of Tang inters the bodies of
relatives in the Qianling Mausoleum, located on Mount Liang outside Chang'an.",
        "html": "706 - In <a href=\"https://wikipedia.org/wiki/China\"
title=\"China\">China, <a
href=\"https://wikipedia.org/wiki/Emperor_Zhongzong_of_Tang\" title=\"Emperor
Zhongzong of Tang\">Emperor Zhongzong of Tang inters the bodies of relatives
in the <a href=\"https://wikipedia.org/wiki/Qianling_Mausoleum\"
title=\"Qianling Mausoleum\">Qianling Mausoleum, located on <a
href=\"https://wikipedia.org/wiki/Mount_Liang\" title=\"Mount Liang\">Mount
Liang outside <a href=\"https://wikipedia.org/wiki/Chang%27an\"
title=\"Chang'an\">Chang'an.",
        "no_year_html": "In <a href=\"https://wikipedia.org/wiki/China\"
title=\"China\">China, <a
href=\"https://wikipedia.org/wiki/Emperor_Zhongzong_of_Tang\" title=\"Emperor
Zhongzong of Tang\">Emperor Zhongzong of Tang inters the bodies of relatives
in the <a href=\"https://wikipedia.org/wiki/Qianling_Mausoleum\"
title=\"Qianling Mausoleum\">Qianling Mausoleum, located on <a
href=\"https://wikipedia.org/wiki/Mount_Liang\" title=\"Mount Liang\">Mount
Liang outside <a href=\"https://wikipedia.org/wiki/Chang%27an\"
title=\"Chang'an\">Chang'an.",
        "links": [
            {
                "title": "China",
                "link": "https://wikipedia.org/wiki/China"
            },
            {
                "title": "Emperor Zhongzong of Tang",
                "link": "https://wikipedia.org/wiki/Emperor_Zhongzong_of_Tang"
            },
            {
                "title": "Qianling Mausoleum",
                "link": "https://wikipedia.org/wiki/Qianling_Mausoleum"
            },
            {
                "title": "Mount Liang",
                "link": "https://wikipedia.org/wiki/Mount_Liang"
            },
            {

```
                "title": "Chang'an",
                "link": "https://wikipedia.org/wiki/Chang%27an"
            }
        ]
    }
]
```


This notebook was converted with convert.ploomber.io