Topic 07: Working with Large Datasets

Overview

This topic marks a transition from building database applications to **working with real-world, large-scale datasets**. You'll learn to acquire, import, and explore substantial databases using the tools and techniques from Topics 1-6. This represents the culmination of the SQL portion of the course, where you apply your skills to datasets containing hundreds of thousands or millions of records. The focus shifts from CRUD operations to **data exploration, analysis, and visualization**.

Course Context

Topics 1-6 recap:

- **Topic 01**: DB-API fundamentals with SQLite
- **Topic 02**: Flask web applications with databases
- **Topic 03**: Database abstraction layer
- **Topic 04**: Keys, joins, and relational design
- **Topic 05**: Peewee ORM
- **Topic 06**: Dataset library for exploration

Topic 07 objectives:

- Acquire large, real-world datasets
- Import data efficiently into SQLite
- Apply previous tools to explore complex data
- Build custom applications for analysis
- Visualize data insights
- Prepare for NoSQL topics (MongoDB)

Available Datasets

1. IMDb Non-Commercial Datasets

Source: https://developer.imdb.com/non-commercial-datasets/

Description: Internet Movie Database (IMDb) provides large TSV (Tab-Separated Values) files containing comprehensive film and television data.

Key datasets:

- `title.basics.tsv.gz` - Basic title information (movies, TV shows, episodes)
- `title.crew.tsv.gz` - Director and writer information
- `title.ratings.tsv.gz` - IMDb ratings and vote counts
- `title.principals.tsv.gz` - Cast and crew members
- `name.basics.tsv.gz` - People in the entertainment industry
- `title.episode.tsv.gz` - TV episode information

Data characteristics:

- **Millions of records**: title.basics contains 10+ million titles
- **Complex relationships**: Links between titles, people, and ratings
- **Real-world messiness**: Missing data, inconsistent formats
- **Regular updates**: Files updated daily

Use cases:

- Movie recommendation systems
- Genre analysis and trends
- Rating correlations
- Career trajectory analysis
- Franchise exploration

2. Chinook Database

Source: https://github.com/lerocha/chinook-database

Description: Sample database representing a digital media store with customers, invoices, tracks, albums, artists, and playlists.

Schema includes:

- **Artists** - Music artists
- **Albums** - Album information
- **Tracks** - Individual songs with duration, price
- **Genres** - Music genres
- **MediaTypes** - Format (MP3, AAC, etc.)
- **Playlists** - User-created playlists
- **Customers** - Customer information
- **Invoices** - Purchase records
- **InvoiceLines** - Items purchased
- **Employees** - Store employees

Data characteristics:

- **Moderate size**: Thousands of records
- **Well-structured**: Properly normalized
- **Complete relationships**: Foreign keys throughout
- **Business domain**: Easy to understand

Use cases:

- Sales analysis
- Customer behavior patterns
- Music genre popularity
- Revenue reports
- Employee performance

Acquiring Datasets

IMDb Datasets

1. Download compressed files:

```
# Download title basics (movies, TV shows)
wget https://datasets.imdbws.com/title.basics.tsv.gz

# Download crew information
wget https://datasets.imdbws.com/title.crew.tsv.gz

# Download ratings
wget https://datasets.imdbws.com/title.ratings.tsv.gz
```

File sizes:

- Compressed: 100-300 MB each
- Uncompressed: 500MB - 2GB each

2. Decompress files:

```
gunzip title.basics.tsv.gz
gunzip title.crew.tsv.gz
gunzip title.ratings.tsv.gz
```

3. Verify files:

```
ls -lah
# Check file sizes

head -n 20 title.basics.tsv
# View first 20 lines
```

TSV format example (title.basics.tsv):

```
tconst  titleType   primaryTitle    originalTitle   isAdult startYear
endYear runtimeMinutes  genres
tt0000001   short   Carmencita  Carmencita  0   1894    \N  1
Documentary,Short
tt0000002   short   Le clown et ses chiens  Le clown et ses chiens  0
1892    \N  5   Animation,Short
```

Column meanings:

- `tconst` - Unique identifier (e.g., "tt0000001")
- `titleType` - movie, tvSeries, tvEpisode, etc.

- `primaryTitle` - Main title
- `startYear` - Release year
- `runtimeMinutes` - Duration
- `genres` - Comma-separated genres
- `\N` - Represents NULL values

Chinook Database

Download from repository:

```
# Clone the repository
git clone https://github.com/lerocha/chinook-database.git

# Navigate to SQLite files
cd chinook-database/ChinookDatabase/DataSources

# The Chinook_Sqlite.sqlite file is ready to use
```

Or copy from the course repository:

```
# Already available in data/ folder
cp ../../data/Chinook_Sqlite.sqlite ./
```

Importing Large Datasets with Pandas

Why Pandas?

Pandas is a powerful Python library for data manipulation and analysis.

Key features for this task:

- **Read large files**: Handles TSV/CSV efficiently
- **Chunked reading**: Process files too large for memory
- **Data cleaning**: Handle missing values, type conversion
- **DataFrame structure**: Tabular data like spreadsheets
- **SQLite integration**: Easy export to database

Installation:

```
pip install pandas
```

Understanding the Import Process

Challenge: IMDb files are too large to load into memory at once.

Solution: Process data in **chunks** (batches).

Workflow:

1. Read file in chunks (e.g., 10,000 rows at a time)
2. Process each chunk
3. Write chunk to database
4. Repeat until entire file is processed

Import Script Explained

import.tsv.py:

```
import pandas as pd
import sqlite3

# Open database connection
connection = sqlite3.connect("imdb.db")

# Configure chunk processing
chunksize = 10000  # Process 10,000 rows at a time
i = 1

# Read TSV file in chunks
for chunk in pd.read_csv("title.basics.tsv", sep="\t",
chunksize=chunksize):
    print("chunk # ", i)
    i = i + 1

    # Append chunk to database table
    chunk.to_sql("title_basics", connection, if_exists="append",
index=False)

connection.close()
```

Breaking it down:

- `pd.read_csv()` with `sep="\t"` - Reads tab-separated values
- `chunksize=10000` - Returns iterator of 10,000-row DataFrames
- `for chunk in ...` - Process one chunk at a time
- `chunk.to_sql()` - Writes DataFrame to SQLite table
- `if_exists="append"` - Adds to existing table (first iteration creates it)
- `index=False` - Don't write DataFrame index as column

Progress tracking:

```
print("chunk # ", i)
```

Shows processing progress - important for large files that take minutes to import.

Importing Multiple Tables

import.crew.tsv.py:

```
import pandas as pd
import sqlite3

connection = sqlite3.connect("imdb.db")
chunksize = 10000
i = 1

for chunk in pd.read_csv("title.crew.tsv", sep="\t", chunksize=chunksize):
    print("chunk # ", i)
    i = i + 1
    chunk.to_sql("title_crew", connection, if_exists="append",
index=False)

connection.close()
```

Pattern:

- Same script, different input file and table name
- Can import multiple TSV files into same database
- Each becomes a separate table

Complete import workflow:

```
# Import titles
python import.tsv.py

# Import crew
python import.crew.tsv.py

# Import ratings (you'd create import.ratings.tsv.py)
# ... etc
```

Import Performance

Typical import times:

- 10,000 rows: ~1-2 seconds
- 1 million rows: ~3-5 minutes
- 10 million rows: ~30-60 minutes

Factors affecting speed:

- Chunk size (10,000 is a good balance)
- Disk speed
- Number of columns

- Data complexity

Optimization tips:

```
# Larger chunks = fewer iterations but more memory
chunksize = 50000  # Faster but uses more RAM

# Smaller chunks = slower but safer for limited memory
chunksize = 1000
```

Exploring Imported Data

Using DB-API (Topic 01)

```
import sqlite3

connection = sqlite3.connect("imdb.db")
cursor = connection.cursor()

# Count total titles
cursor.execute("SELECT COUNT(*) FROM title_basics")
print(cursor.fetchone()[0])

# Find all movies from 2020
cursor.execute("""
    SELECT primaryTitle, startYear, runtimeMinutes
    FROM title_basics
    WHERE titleType = 'movie' AND startYear = '2020'
    LIMIT 10
""")
for row in cursor.fetchall():
    print(row)

connection.close()
```

Using Dataset Library (Topic 06)

```
import dataset

db = dataset.connect('sqlite:///imdb.db')
titles = db['title_basics']

# Find long movies
long_movies = titles.find(titleType='movie', runtimeMinutes={'>=': 180})
for movie in long_movies:
    print(f"{movie['primaryTitle']} - {movie['runtimeMinutes']} minutes")

# Count by type
```

```
from collections import Counter
types = Counter(row['titleType'] for row in titles.all())
print(types)
```

Using Peewee ORM (Topic 05)

```
from peewee import *

db = SqliteDatabase('imdb.db')

class TitleBasics(Model):
    tconst = CharField()
    titleType = CharField()
    primaryTitle = CharField()
    startYear = CharField()
    runtimeMinutes = CharField()
    genres = CharField()

    class Meta:
        database = db
        table_name = 'title_basics'

# Query
movies_2020 = TitleBasics.select().where(
    TitleBasics.titleType == 'movie',
    TitleBasics.startYear == '2020'
).limit(10)

for movie in movies_2020:
    print(movie.primaryTitle)
```

Project Ideas

1. Movie Browser Application

Flask application features:

- Search movies by title
- Filter by year, genre, runtime
- Display ratings and vote counts
- Show directors and writers
- Paginated results (100+ per page)

Technologies:

- Flask for web framework
- Dataset or Peewee for database access
- HTML templates with Bootstrap
- JavaScript for interactive filtering

2. Genre Analysis Dashboard

Visualizations:

- Genre popularity over time
- Average runtime by genre
- Rating distributions
- Most prolific directors

Technologies:

- Matplotlib or Plotly for charts
- Pandas for data aggregation
- Flask to serve visualizations
- Dataset for queries

3. TV Series Explorer

Features:

- List all TV series
- Episode guide for each series
- Season statistics
- Rating trends across seasons

Technologies:

- Join title_basics and title_episode tables
- Raw SQL for complex queries
- Flask templates
- Bootstrap tables

4. Chinook Music Store Analytics

Reports:

- Top-selling artists
- Revenue by country
- Customer purchase history
- Genre preferences by region
- Employee sales performance

Technologies:

- SQL joins across multiple tables
- Aggregation queries (SUM, AVG, COUNT)
- Chart visualizations
- Flask dashboard

5. Career Trajectory Analyzer

Features:

- Track actor/director careers over time
- Show all films by person
- Visualize career timeline
- Identify peak years

Technologies:

- Join title_basics, title_principals, name_basics
- Timeline visualizations
- Complex SQL queries
- Search functionality

Technical Challenges

1. Handling NULL Values

IMDb uses `\N` for NULL:

```
# In Pandas import
chunk = chunk.replace('\\N', None)
chunk.to_sql(...)
```

Or handle in queries:

```
SELECT * FROM title_basics
WHERE startYear IS NOT NULL
  AND startYear != '\N'
```

2. Data Type Conversions

Years and numbers stored as strings:

```
# Convert in application
year = int(movie['startYear']) if movie['startYear'] != '\\N' else None
```

Or convert during import:

```
chunk['startYear'] = pd.to_numeric(chunk['startYear'], errors='coerce')
```

3. Large Result Sets

Don't load millions of rows at once:

```
# BAD: Loads entire table
movies = list(db['title_basics'].all())  # Out of memory!

# GOOD: Use filters
movies = db['title_basics'].find(titleType='movie', startYear='2020')
```

Use pagination:

```
# Flask route with pagination
@app.route("/movies")
def movies():
    page = int(request.args.get('page', 1))
    per_page = 100
    offset = (page - 1) * per_page

    cursor.execute("""
        SELECT * FROM title_basics
        WHERE titleType = 'movie'
        LIMIT ? OFFSET ?
    """, (per_page, offset))

    movies = cursor.fetchall()
    return render_template('movies.html', movies=movies, page=page)
```

4. Slow Queries

Create indexes for frequently queried columns:

```
connection = sqlite3.connect("imdb.db")
cursor = connection.cursor()

# Create indexes
cursor.execute("CREATE INDEX idx_title_type ON title_basics(titleType)")
cursor.execute("CREATE INDEX idx_start_year ON title_basics(startYear)")
cursor.execute("CREATE INDEX idx_primary_title ON
title_basics(primaryTitle)")

connection.commit()
```

Speeds up:

- `WHERE titleType = 'movie'`
- `WHERE startYear = '2020'`
- `WHERE primaryTitle LIKE 'Star%'`

5. Complex Relationships

Joining multiple tables:

```
-- Get movie with ratings and crew
SELECT
    tb.primaryTitle,
    tb.startYear,
    tr.averageRating,
    tr.numVotes,
    tc.directors,
    tc.writers
FROM title_basics tb
LEFT JOIN title_ratings tr ON tb.tconst = tr.tconst
LEFT JOIN title_crew tc ON tb.tconst = tc.tconst
WHERE tb.titleType = 'movie'
  AND tr.averageRating > 8.0
ORDER BY tr.numVotes DESC
LIMIT 100
```

Visualization Techniques

Using Matplotlib

```
import matplotlib.pyplot as plt
import sqlite3

connection = sqlite3.connect("imdb.db")
cursor = connection.cursor()

# Get genre counts
cursor.execute("""
    SELECT genres, COUNT(*) as count
    FROM title_basics
    WHERE titleType = 'movie'
    GROUP BY genres
    ORDER BY count DESC
    LIMIT 10
""")

data = cursor.fetchall()
genres = [row[0] for row in data]
counts = [row[1] for row in data]

plt.figure(figsize=(10, 6))
plt.barh(genres, counts)
plt.xlabel('Number of Movies')
plt.title('Top 10 Movie Genres')
plt.tight_layout()
plt.savefig('static/genres.png')
```

Using Plotly (Interactive)

```
import plotly.express as px
import pandas as pd

# Query data
df = pd.read_sql("""
    SELECT startYear, COUNT(*) as count
    FROM title_basics
    WHERE titleType = 'movie'
      AND startYear != '\\N'
    GROUP BY startYear
""", connection)

# Create interactive line chart
fig = px.line(df, x='startYear', y='count',
              title='Movies Released Per Year')
fig.write_html('templates/chart.html')
```

Embedding in Flask

```
@app.route("/charts")
def charts():
    # Generate chart
    create_genre_chart()
    return render_template("charts.html")
```

```
<!-- charts.html -->
<img src="/static/genres.png" alt="Genre Chart">
```

Assignment Suggestions

Beginner Level

1. Import IMDb title.basics dataset
2. Create a Flask app that lists all movies
3. Add search by title functionality
4. Display movie count statistics

Intermediate Level

1. Import multiple IMDb tables
2. Build a movie detail page with ratings
3. Create genre filter
4. Add pagination for large result sets

5. Show basic visualizations (bar charts)

Advanced Level

1. Full-featured movie database browser
2. Advanced search (multiple filters)
3. Interactive dashboards with Plotly
4. Recommendation system (similar movies)
5. Career analysis tools
6. Performance optimization with indexes

Data Science Focus

1. Analyze genre trends over decades
2. Correlate runtime with ratings
3. Identify most prolific directors
4. Study rating distributions
5. Predict movie success factors
6. Export analysis to Jupyter notebooks

Comparing with Previous Topics

Aspect	Topics 1-6	Topic 7
Data size	Small (dozens of rows)	Large (millions of rows)
Schema	Simple (2-3 tables)	Complex (5+ tables)
Data source	Manual creation	Real-world import
Focus	CRUD operations	Exploration & analysis
Queries	Simple lookups	Aggregations, joins
Goal	Learn fundamentals	Apply to real data
Visualization	None	Charts and graphs

Running Your Project

1. Set Up Environment

```
# Create project directory
mkdir imdb-explorer
cd imdb-explorer

# Install dependencies
pip install flask pandas matplotlib plotly dataset peewee
```

2. Download and Import Data

```
# Download data
wget https://datasets.imdbws.com/title.basics.tsv.gz
gunzip title.basics.tsv.gz

# Import to database
python import.tsv.py
```

3. Build Application

```
# app.py
from flask import Flask, render_template, request
import dataset

app = Flask(__name__)
db = dataset.connect('sqlite:///imdb.db')

@app.route("/")
def index():
    titles = db['title_basics']
    movies = titles.find(titleType='movie', _limit=100)
    return render_template('index.html', movies=list(movies))

if __name__ == "__main__":
    app.run(debug=True)
```

4. Run and Explore

```
flask --app app run --debug
```

Key Concepts and Terminology

- **TSV**: Tab-Separated Values file format
- **Chunked Reading**: Processing large files in batches
- **Pandas DataFrame**: Two-dimensional labeled data structure
- **Data Import**: Loading external data into database
- **Indexing**: Database optimization for faster queries
- **Pagination**: Displaying results in pages
- **Aggregation**: Computing statistics (COUNT, SUM, AVG)
- **Visualization**: Graphical representation of data
- **Real-world Data**: Messy, incomplete data from actual sources

Best Practices

1. ☑ **Import in chunks** to handle large files
2. ☑ **Create indexes** on frequently queried columns

3. ☑ **Use pagination** for large result sets
4. ☑ **Handle NULL values** appropriately
5. ☑ **Add progress indicators** for long operations
6. ☑ **Test queries** before building UI
7. ☑ **Document data schema** for reference
8. ☑ **Optimize slow queries** with EXPLAIN
9. ☑ **Use appropriate tool** for the task (Dataset for exploration, Peewee for structure)
10. ☑ **Visualize insights** to communicate findings

Transition to NoSQL

This topic **concludes the SQL/relational portion** of the course.

What you've learned:

- Database fundamentals (Topics 1-2)
- Abstraction and design patterns (Topics 3-4)
- ORM concepts (Topic 5)
- Data exploration tools (Topic 6)
- Working with real, large datasets (Topic 7)

Coming next (Topics 8+):

- MongoDB and document databases
- NoSQL data modeling
- Schema-less design
- MapReduce and aggregation
- Geospatial queries

Skills that transfer:

- Query thinking (filters, aggregations)
- Data modeling concepts
- Application design patterns
- Visualization techniques

Key Takeaways

1. **Real-world datasets are large and messy** - requires different techniques than toy examples
2. **Pandas enables efficient import** of large files with chunked reading
3. **All previous tools apply** - Dataset, Peewee, raw SQL all work with large data
4. **Performance matters** - indexes, pagination, and query optimization crucial
5. **Visualization communicates insights** - charts more powerful than tables
6. **Project work solidifies learning** - building exploratory tools cements concepts
7. **Data exploration is different** from CRUD application development
8. **Choose tools appropriately** - Dataset for exploration, ORM for structure
9. **Handle NULL values carefully** - real data has missing information
10. **This prepares you for NoSQL** - concepts and patterns transfer

Topic 7 represents the culmination of SQL learning, where you take all the tools and techniques from Topics 1-6 and apply them to substantial, real-world datasets. The projects you build here demonstrate mastery of relational database concepts and prepare you for the shift to document-oriented databases in the MongoDB topics that follow.