

Databases for Analytics

Beyond MySQL

Learning Objectives

- **Skills:** You should know how to ...
 - Configure and use SQLAlchemy
 - Use %sql magic in either mode
 - Use Pandas to glue relational and non-relational data
 - Get started with SQLite for lightweight databases
- **Theory:** You should be able to explain ...
 - The mechanics of connecting to a database in SQLAlchemy
 - Basic functions of an Object-Relational Mapper
 - The dangers of using non-relational data sources

SQLAlchemy

The de facto standard for RDBMS Access in Python

SQLAlchemy

SQLAlchemy is a library for accessing SQL databases in Python.

- Provides a plug-in system for connecting to various DBMSes.
- Works as an Object Relational Mapper for converting Python \rightarrow SQL and SQL \rightarrow Python
- Key is creating a SQLAlchemy Engine for creating and configuring DB connections as needed.

Engines

An engine is a reusable configuration for connecting to a database. You create one before doing anything else.

```
from sqlalchemy import create_engine
engine = create_engine("mysql+pymysql://root:mysql@localhost/movies_tonight")
# We can now use engine to create new database connections
```

Connection Strings

DBMS Vendor

DB User

User Password

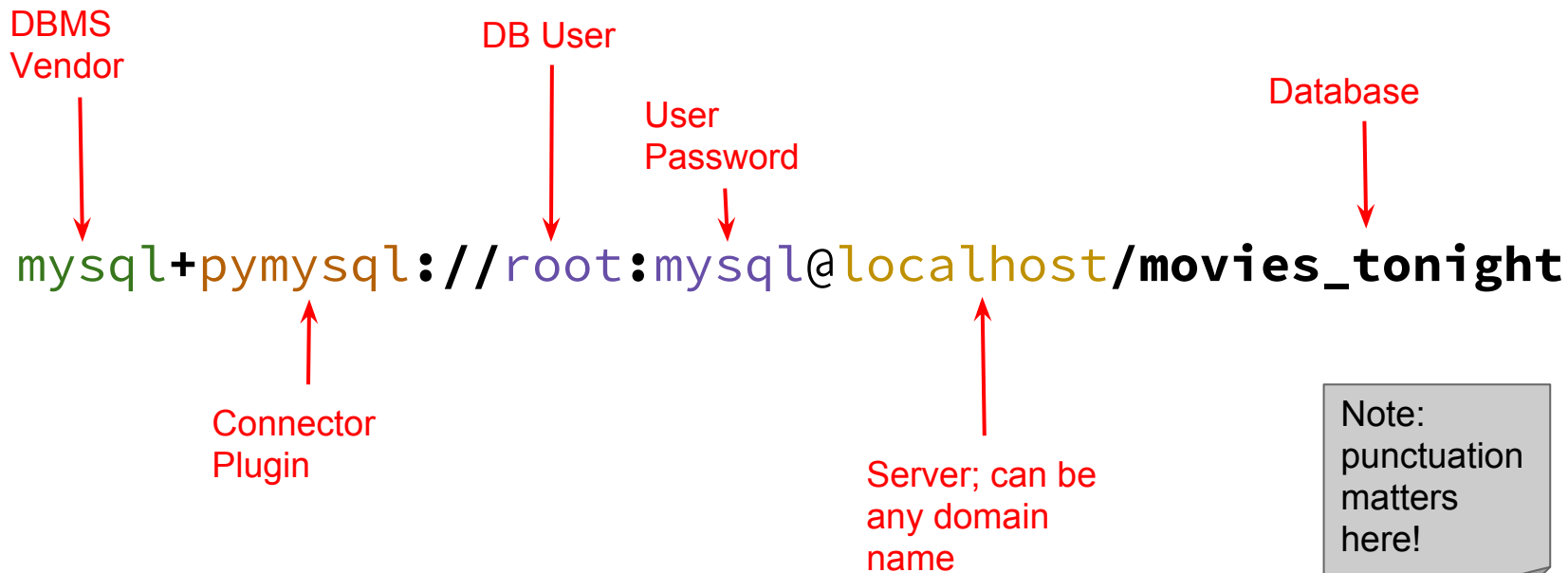
Database

Connector Plugin

Server; can be any domain name

`mysql+pymysql://root:mysql@localhost/movies_tonight`

Note: punctuation matters here!



The diagram illustrates the components of a MySQL connection string: `mysql+pymysql://root:mysql@localhost/movies_tonight`. Red arrows point from labels to specific parts of the string: 'DBMS Vendor' points to 'mysql', 'Connector Plugin' points to 'pymysql', 'DB User' points to 'root', 'User Password' points to 'mysql', 'Server; can be any domain name' points to 'localhost', and 'Database' points to 'movies_tonight'. A note box in the bottom right corner states 'Note: punctuation matters here!'.

Sessions

Each connection to a database is a *session*.

One can open multiple sessions at the same time.

Each database operation (DDL or DML) exists within a session context.

```
from sqlalchemy.orm import sessionmaker
Session = sessionmaker(bind=engine)
session = Session() # session is a connection
```

ORM: Python → SQL DDL

SQLAlchemy provides full support for most SQL DDL statements. You can even use it to create tables based on a Python class:

```
from sqlalchemy import Column, Integer, String
class Artist(Base):
    __tablename__ = 'artists'
    id = Column(Integer, primary_key=True)
    name = Column(String)
Base.metadata.create_all(engine)
```


ORM: Python → SQL DML

SQLAlchemy also provides Python equivalents to SQL **INSERT**, **UPDATE**, and **DELETE** commands.

The idea is that you create/update/delete Python ***objects*** and then ***persist*** them in the database.

```
artists_list = [Artist(name='Minnie Driver'),  
                Artist(name='STanley Tucci'),...]  
session.add.all(artists_list)  
session.commit()
```

Artist class was
defined on
previous slide.

ORM: SQL Data → Python Objects

SQLAlchemy sessions have a **query()** method for retrieving data as lists of Python objects.

```
# query() returns a list of objects of the specified type  
artist_list = session.query(Artist).order_by(Artist.name)
```

Raw SQL Queries

One can also get away without the ORM by executing SQL directly, which is usually what you want anyway. The result is a list of *tuples*.

```
from sqlalchemy.sql import text
```

```
s = text('''SELECT `Name`  
           FROM `Artists`''')
```

```
conn.execute(s).fetchall()
```

```
→ [(u'Minnie Driver'), (u'Stanley Tucci'), ... ]
```

Triple-quoted query string
with line breaks



More about SQLAlchemy

SQLAlchemy has lots and lots more tricks ...

You can RTFM or just follow along with the tutorial:

<http://docs.sqlalchemy.org/en/latest/orm/tutorial.html>

Pandas

Useful SQL and RDBMS-like features

Pandas & SQLAlchemy

Pandas uses the SQLAlchemy library as the basis for for its `read_sql()`, `read_sql_table()`, and `read_sql_query()` functions.

- This allows you to retrieve query results as a Pandas DataFrame
- However, you need to initiate the database connection with SQLAlchemy first

`read_sql()` Example

```
import pandas as pd
from sqlalchemy import create_engine
engine = create_engine("mysql+pymysql://root:mysql@localhost/deals")

with engine.connect() as conn, conn.begin():
    deals = pd.read_sql('Select * from Deals', conn)
    players=pd.read_sql('Select * from Players', conn)
    companies=pd.read_sql('Select * from Companies', conn)
# deals, players, and companies are Pandas DataFrames
```

merge() and join()

Pandas supports its own **join** syntax for DataFrames.

```
# using the Pandas merge() function, one join at a time
```

```
deal_players = pd.merge(deals,players,on='DealID')
```

```
deal_companies =
```

```
    pd.merge(deal_players,companies,on='CompanyID')
```

```
#using the DataFrame join() method, which can be chained
```

```
deal_players2 =
```

```
    deals.join(players,on='DealID').join(companies,on='CompanyID')
```


Mixing SQL Data with DataFrames

Pandas `merge()` and `join()` work like `SELECT` queries with `JOINS`, except with `DataFrames` instead of tables.

So, who says that the `DataFrames` have to come from a SQL query? They can come from anywhere!

So, we can now glue data together from multiple sources, even if they come from non-relational data sources.

Jupyter

A quick review

The Jupyter Runtime Environment

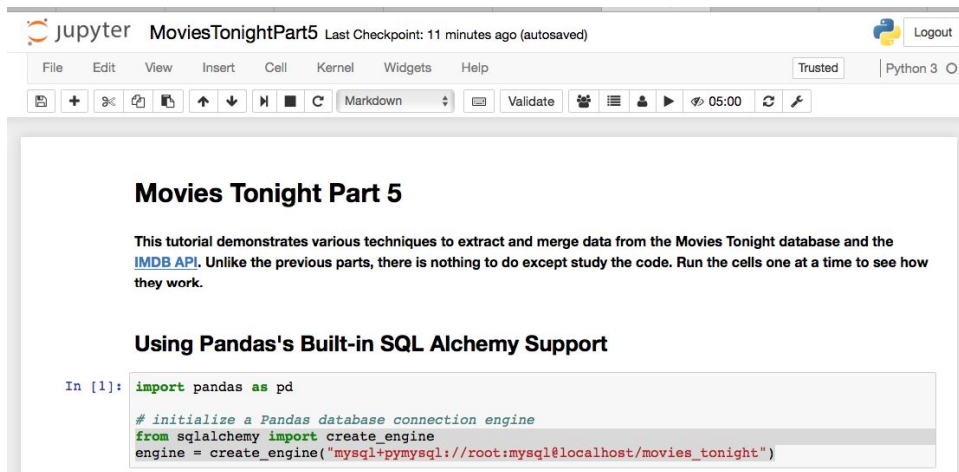
Jupyter is based on the ipython *shell* for running python code in a controlled environment like Anaconda.

- Anaconda handles software packages, integrates a few apps, etc.
- Jupyter makes all of that available with slick interactive interface.

So, while we can use Jupyter in Anaconda, we don't actually need Anaconda to run Jupyter.

Jupyter Notebooks

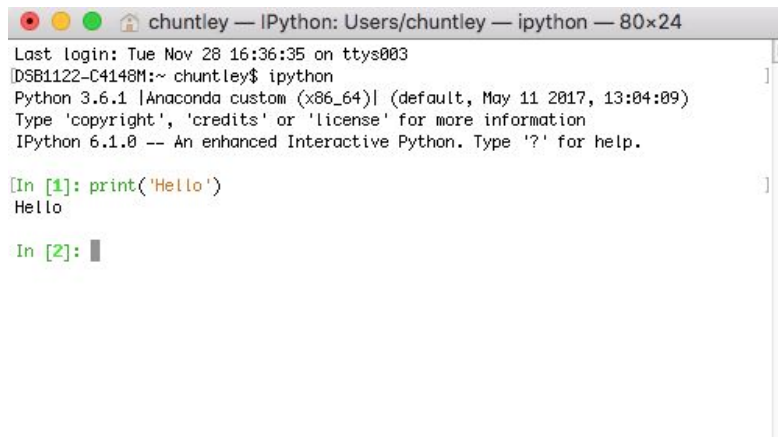
Jupyter Notebooks is a web implementation of the ipython interface. We don't need the web to run Jupyter. We can just use the old-school ipython shell.



The screenshot shows a Jupyter Notebook interface. The top bar includes the Jupyter logo, the notebook name 'MoviesTonightPart5', and a 'Last Checkpoint: 11 minutes ago (autosaved)' status. Below the top bar is a menu bar with 'File', 'Edit', 'View', 'Insert', 'Cell', 'Kernel', 'Widgets', and 'Help'. A toolbar with various icons for file operations and execution is visible. The main content area displays a tutorial page titled 'Movies Tonight Part 5'. The text on the page reads: 'This tutorial demonstrates various techniques to extract and merge data from the Movies Tonight database and the [IMDB API](#). Unlike the previous parts, there is nothing to do except study the code. Run the cells one at a time to see how they work.' Below this text is a section header 'Using Pandas's Built-in SQL Alchemy Support'. At the bottom, there is a code cell with the following Python code:

```
In [1]: import pandas as pd

# initialize a Pandas database connection engine
from sqlalchemy import create_engine
engine = create_engine("mysql+pymysql://root:mysql@localhost/movies_tonight")
```



The screenshot shows an IPython terminal window. The title bar reads 'chuntley — IPython: Users/chuntley — ipython — 80x24'. The terminal output shows the last login time and the command 'ipython' being executed. The output indicates that Python 3.6.1 is running on an Anaconda custom environment. Below this, the user types 'print('Hello')' and the output 'Hello' is displayed. The prompt 'In [2]:' is visible at the bottom.

```
Last login: Tue Nov 28 16:36:35 on ttys003
DSB1122-C4148M:~ chuntley$ ipython
Python 3.6.1 |Anaconda custom (x86_64)| (default, May 11 2017, 13:04:09)
Type 'copyright', 'credits' or 'license' for more information
IPython 6.1.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]: print('Hello')
Hello

In [2]:
```

%sql Magic

We can use the **ipython-sql** library to run raw SQL code inside a Notebook cell. We have done this in class already.

First, we need to do a little setup to load the library and establish a database connection.

```
In [41]: # standard imports for %sql magic
         %load_ext sql

         # initialize a %sql database connection; may have to adjust username and password
         %sql mysql+pymysql://root:mysql@localhost/movies_tonight
```

Operating Modes

Inline queries with %sql

```
In [42]: # A one-line %sql call
movie_shows_rs = %sql SELECT * FROM MOVIES JOIN SH
```

All SQL code must be on one line but we can directly capture the results as a Python variable

Multi-line queries with %%sql

```
In [43]: %%sql
SELECT *
FROM MOVIES
      JOIN SHOWS ON (MOVIES.ID=SHOWS.MID)
      JOIN THEATERS ON (SHOWS.TID=THEATERS.ID)
```

```
In [44]: movie_shows_rs = _
```

SQL code can take up a whole cell but we can't use it directly in Python. Instead, we have to capture the results using the special `_` variable in a separate cell.

Converting to DataFrames

%sql and %%sql queries return ResultSets, not DataFrames. Fortunately, there is an built-in method for converting any ResultSet to a DataFrame.

```
In [42]: # A one-line %sql call
movie_shows_rs = %sql SELECT * FROM MOVIES JOIN SHOWS ON (MOVIES.ID=SHOWS.MID) JOIN THEATERS ON
# Then a conversion to a DataFrame
movie_shows_df = movie_shows_rs.DataFrame()
movie_shows_df
```

SQLite

A tiny, file-based DBMS library in Python

SQLite FTW!

SQLite does not require a DBMS or an installer. Any computer with a modern version of Python can use SQLite to store relational data.

- **File-based**, so that we can copy files and use them directly (and even check them into GitHub)
- **Tiny**, requiring almost no CPU time or memory
- **(Mostly) Standard SQL**, so we can use what we already know

No SQLAlchemy Needed

SQLite can be called directly using the `sqlite3` library that's built into Python.

```
import sqlite3  
conn = sqlite3.connect("deals.db")
```

A tutorial:

<https://www.dataquest.io/blog/python-pandas-databases/>

But SQLAlchemy is easier

Most things work exactly the same as with MySQL.
You just have to use a different connection string.
However, the SQLite syntax is its own dialect.

<https://sqlite.org/lang.html>

SQLAlchemy also provides a handy guide:

<http://docs.sqlalchemy.org/en/latest/dialects/sqlite.html>

Databases for Analytics

Beyond MySQL