SQL Alchemy

NAYA College

Agenda

- Introduction
- Engine
- Table
- Select
- Where
- Join
- Group by
- Relation with pandas
- Create database and tables

Introduction

SQLAlchemy provides a nice “Pythonic” way of interacting with databases.
So rather than dealing with the differences between specific dialects of traditional SQL such as MySQL or PostgreSQL or Oracle, you can leverage the Pythonic framework of SQLAlchemy to streamline your workflow and more efficiently query your data.

The power of an API lies in its objects, and SQLAlchemy provides "Pythonic" objects to represent SQL functionalities. More specifically, we will be interested in `FromClause` elements.

```
from sqlalchemy import create_engine, Table, MetaData, Column,
                       Integer, String, Float
from sqlalchemy import select, desc
from sqlalchemy.sql import func
```

Engine

Engine is the most fundamental object of SQLAlchemy, and it defines the database we work with.

- Create engine

```
engine = create_engine('sqlite:///chinook.db', echo=False)
```

- Get tables name

```
engine.table_names()
Answer: ['albums', 'artists', 'customers', 'employees', 'genres']
```

MetaData

- Create Metadata

```
metadata = MetaData()
```

- The connect() method returns a Connection object, through which can send commands to the database.

```
conn = engine.connect()
```

Table

- Get Table from engine with metadata that we created

```
albums = Table('albums', metadata, autoload_with=engine)
```

- Select, execute, and print them..

```
query = select([albums])
result = conn.execute(query)
print(result.fetchmany(5))
Answer:
[(1, 'For Those About To Rock We Salute You', 1),
 (2, 'Balls to the Wall', 2),
 (3, 'Restless and Wild', 2),
 (4, 'Let There Be Rock', 1),
 (5, 'Big Ones', 3)]
```

Select

- Select, execute, and print them..

```
query = select([albums])
result = conn.execute(query)
print(result.fetchmany(5))
Answer:
[(1, 'For Those About To Rock We Salute You', 1),
 (2, 'Balls to the Wall', 2),
 (3, 'Restless and Wild', 2),
 (4, 'Let There Be Rock', 1),
 (5, 'Big Ones', 3)]
```

Select

- Select (only few columns) , execute, and print them..

```
query = select([albums.c.AlbumId])
result = conn.execute(query)
print(result.fetchmany(5))
Answer:
[(1,),
 (2),
 (3,),
 (4,),
 (5,)]
```

SQL - Select

- For “SQL” people…
  A straight-forward approach would be to use our connection and "send" SQL commands.

```
query = '''
SELECT * FROM albums
WHERE Title LIKE '%the best of%'
'''
result = conn.execute(query)
print(result.fetchall())
Answer:
[(13, 'The Best Of Billy Cobham', 10),
 (20, 'The Best Of Buddy Guy - The Millenium Collection', 15)]
```

Select-where

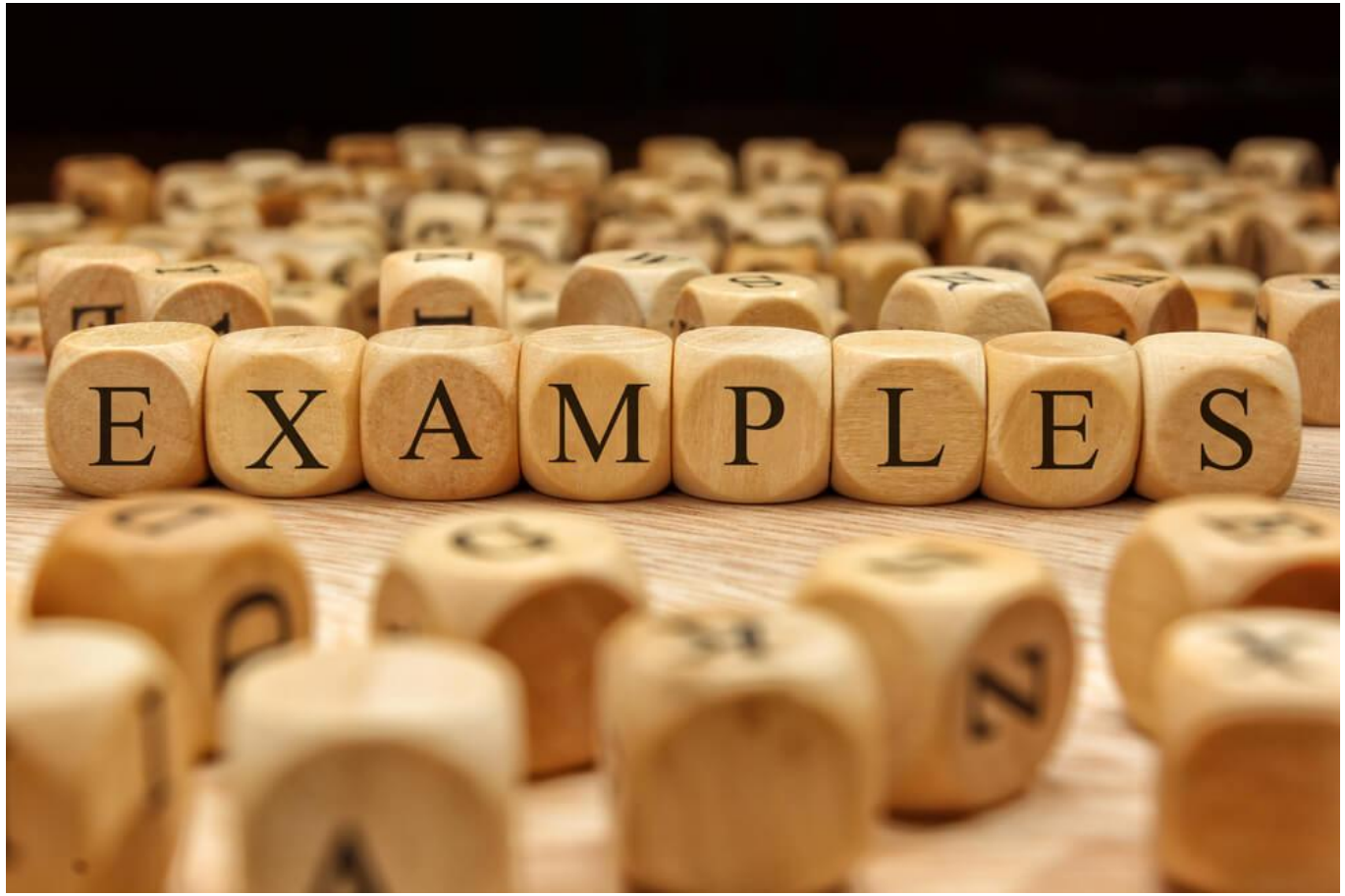
- For select with filter used by whereclause argument of select()

```
query = select([albums],
               whereclause=albums.columns.Title.like('%the best of%')))
```

- Alternatively, can be used by where() method of the Select object.

```
query = select([albums])
                    .where(albums.columns.Title.like('%the best of%'))
```

Example – (1-4)



SQL Alchemy

Exresices:

Part 1 - create the tables tracks, albums and artists both as SQLAlchemy Tables
and print 10 records from any table.

Part 2 - from tracks table filter only the 3th 'AlbumId'
and print 10 records from any table.

Join

- For Join between tables need to create the table first and create join object.

```
#create albums,artists table
artists = Table('artists', metadata, autoload_with=engine)
albums = Table('albums', metadata, autoload_with=engine)
```

Both select() and join() are special cases of the more general FromClause class, which basically means they can be used within the FROM clause of a SELECT statement.

```
# join between tables
join_stmt = artists.join(albums, artists.c.ArtistId == albums.c.ArtistId)
query = select([albums.c.Title, artists.c.Name],
               from_obj=join_stmt)
```

Alternatively, when we use JOINs we know what FROM clause we want, so here we make use of the select_from() method.

```
query = select([albums.c.Title, artists.c.Name]).select_from(join_stmt)
```

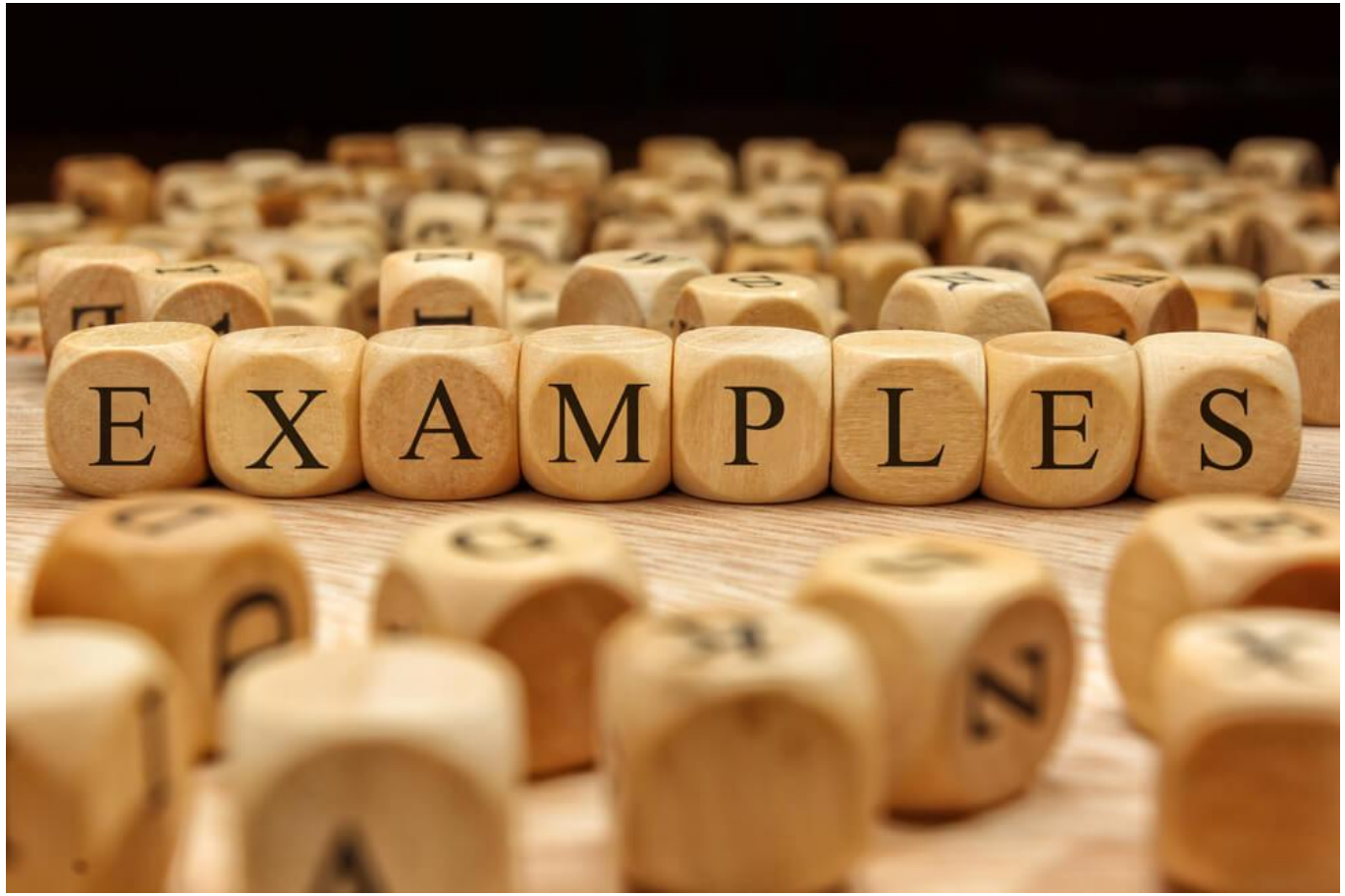
Group by

The GROUP BY statement groups rows that have the same values into summary rows, like "find the number of customers in each country".

The GROUP BY statement is often used with aggregate functions (COUNT, MAX, MIN, SUM, AVG) to group the result-set by one or more columns.

```
#create invoices,customers table
invoices = Table('invoices', metadata, autoload_with=engine)

query = select([invoices.c.CustomerId, func.count(invoices.c.InvoiceId)])\
            .select_from(invoices)\
            .group_by(invoices.c.CustomerId)
```

Example – (5-6)



Relation to pandas

Very often save the query result as a data-frame.
Luckily, the ResultProxy.fetchall() method returns a list of results that can be constructed as a data-frame using the standard pd.DataFrame() constructor.

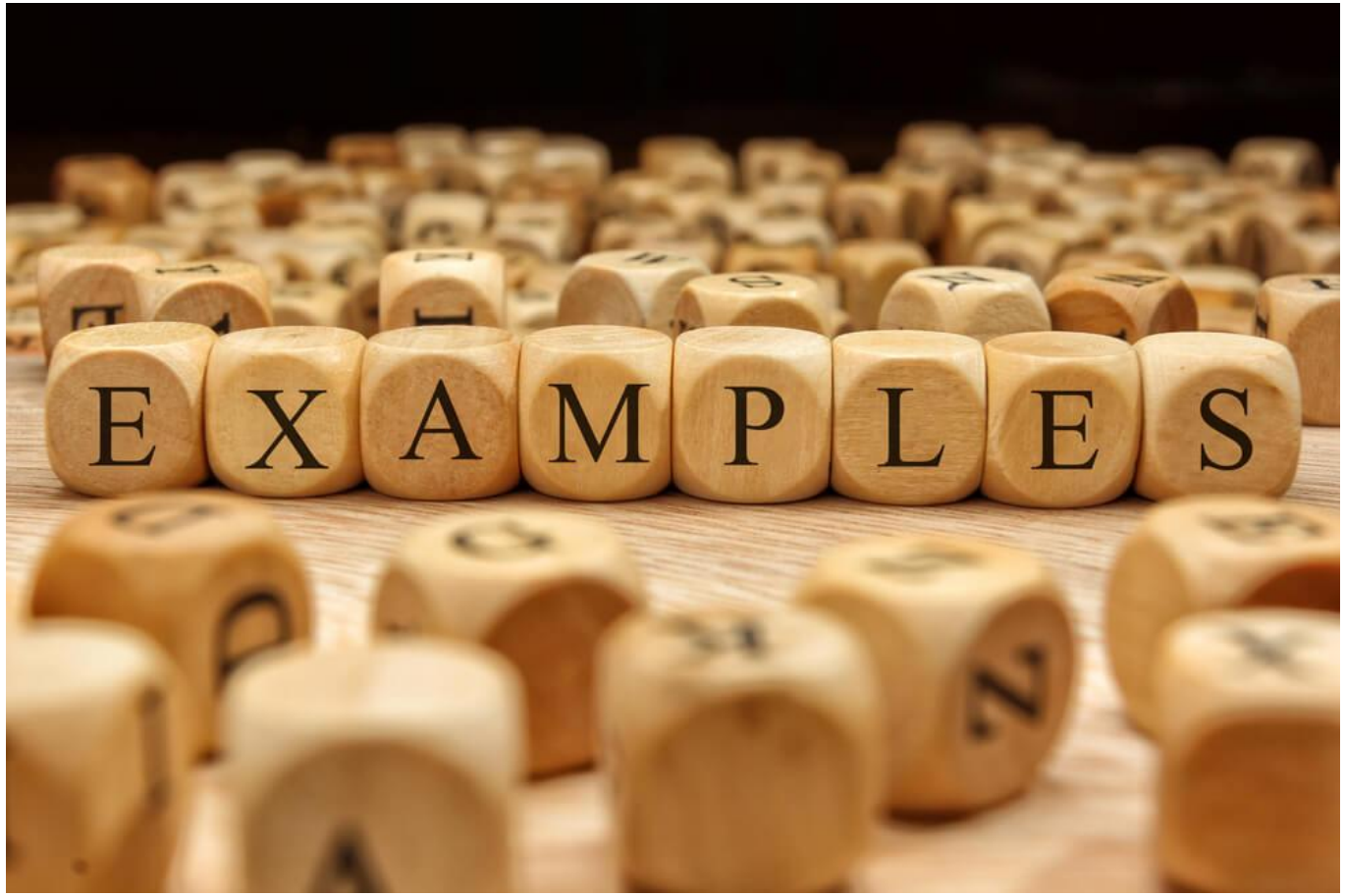
```
#create albums,customers table
albums = Table('albums', metadata, autoload_with=engine)

query = select([albums])
result = conn.execute(query)

df_albums = pd.DataFrame(result.fetchall(), columns=result.keys())
print(df_albums.head())
```

Relation to pandas

Pandas offers pd.read_sql_query(sql, con) to directly send SQL commands through a given DB connection. SQLAlchemy engine is one of the options for con.

```
sql = 'SELECT * FROM employees Order By BirthDate'
df_tracks = pd.read_sql_query(sql, con=engine)
print(df_tracks.head())
```

Example – 7



SQL Alchemy

Exresices:

- Part 1 - create the tables tracks, albums and artists both as SQLAlchemy Tables and as pandas DataFrames.
- Part 2 - Answer the following questions in two ways - using SQLAlchemy and using pandas.
  1. What is the size of the table tracks?
  2. Which artist has the highest number of tracks?

Create DB and tables

Used by in-memory-only SQLite database. This is an easy way to test things without needing to have an actual database defined anywhere.

```
# Create engine
engine = create_engine('sqlite:///:memory:', echo=False)
# Create metadata
metadata = MetaData()

users = Table('users', metadata,
    Column('id', Integer),
    Column('name', String),
    Column('fullname', String),
)

metadata.create_all(engine)
```

Insert data

The insert() method is a wrapper for SQL's INSERT command.

```
query = users.insert().values(id=1234, name='jack', fullname='Jack Jones')
result = conn.execute(query)
```

Example – (8-9)