# Import create\_engine

from sqlalchemy import create\_engine

# Create an engine that connects to the census.sqlite file: engine

engine = create\_engine('sqlite:///census.sqlite')

# Print table names

print(engine.table\_names())

==

# Import Table

from sqlalchemy import Table

# Reflect census table from the engine: census

census = Table('census', metadata, autoload=True, autoload\_with=engine)

# Print census table metadata

print(repr(census))

==

# Reflect the census table from the engine: census

census = Table('census', metadata, autoload=True, autoload\_with=engine)

# Print the column names

print(census.columns.keys())

# Print full table metadata

print(repr(metadata.tables['census']))

==

# Build select statement for census table: stmt

stmt = 'select \* from census'

# Execute the statement and fetch the results: results

results = connection.execute(stmt).fetchall()

# Print results

print(results)

==

# Import select

from sqlalchemy import select

# Reflect census table via engine: census

census = Table('census', metadata, autoload=True, autoload\_with=engine)

# Build select statement for census table: stmt

stmt = select([census])

# Print the emitted statement to see the SQL emitted

print(stmt)

# Execute the statement and print the results

print(connection.execute(stmt).fetchall())

==

# Get the first row of the results by using an index: first\_row

first\_row = results[0]

# Print the first row of the results

print(first\_row)

# Print the first column of the first row by using an index

print(first\_row[0])

# Print the 'state' column of the first row by using its name

print(first\_row['state'])

==

# Import create\_engine function

from sqlalchemy import create\_engine

# Create an engine to the census database

engine = create\_engine('postgresql+psycopg2://'+'student:datacamp'+'@postgresql.csrrinzqubik.us-east-1.rds.amazonaws.com'+':5432/census')

# Use the .table\_names() method on the engine to print the table names

print(engine.table\_names())

==

# Create a select query: stmt

stmt = select([census])

# Add a where clause to filter the results to only those for New York

stmt = stmt.where(census.columns.state=='New York')

# Execute the query to retrieve all the data returned: results

results = connection.execute(stmt).fetchall()

# Loop over the results and print the age, sex, and pop2008

for result in results:

print(result.age, result.sex, result.pop2008)

==

# Create a query for the census table: stmt

stmt = select([census])

# Append a where clause to match all the states in\_ the list states

stmt = stmt.where(census.columns.state.in\_(states))

# Loop over the ResultProxy and print the state and its population in 2000

for result in connection.execute(stmt):

print(result.state, result.pop2000)

==

# Import and\_

from sqlalchemy import and\_

# Build a query for the census table: stmt

stmt = select([census])

# Append a where clause to select only non-male records from California using and\_

stmt = stmt.where(

# The state of California with a non-male sex

and\_(census.columns.state == 'California',

census.columns.sex != 'M'

)

)

# Loop over the ResultProxy printing the age and sex

for result in connection.execute(stmt):

print(result.age, result.sex)

==

# Build a query to select the state column: stmt

stmt = select([census.columns.state])

# Order stmt by the state column

stmt = stmt.order\_by(census.columns.state)

# Execute the query and store the results: results

results = connection.execute(stmt).fetchall()

# Print the first 10 results

print(results[:10])

==

# Import desc

from sqlalchemy import desc

# Build a query to select the state column: stmt

stmt = select([census.columns.state])

# Order stmt by state in descending order: rev\_stmt

rev\_stmt = stmt.order\_by(desc(census.columns.state))

# Execute the query and store the results: rev\_results

rev\_results = connection.execute(rev\_stmt).fetchall()

# Print the first 10 rev\_results

print(rev\_results[:10])

==

# Build a query to select state and age: stmt

stmt = select([census.columns.state, census.columns.age])

# Append order by to ascend by state and descend by age

stmt = stmt.order\_by(census.columns.state, desc(census.columns.age))

# Execute the statement and store all the records: results

results = connection.execute(stmt).fetchall()

# Print the first 20 results

print(results[:20])

==

# Build a query to count the distinct states values: stmt

stmt = select([func.count(census.columns.state.distinct())])

# Execute the query and store the scalar result: distinct\_state\_count

distinct\_state\_count = connection.execute(stmt).scalar()

# Print the distinct\_state\_count

print(distinct\_state\_count)

==

# Import func

from sqlalchemy import func

# Build a query to select the state and count of ages by state: stmt

stmt = select([census.columns.state, func.count(census.columns.age)])

# Group stmt by state

stmt = stmt.group\_by(census.columns.state)

# Execute the statement and store all the records: results

results = connection.execute(stmt).fetchall()

# Print results

print(results)

# Print the keys/column names of the results returned

print(results[0].keys())

==

# Import func

from sqlalchemy import func

# Build an expression to calculate the sum of pop2008 labeled as population

pop2008\_sum = func.sum(census.columns.pop2008).label('population')

# Build a query to select the state and sum of pop2008: stmt

stmt = select([census.columns.state, pop2008\_sum])

# Group stmt by state

stmt = stmt.group\_by(census.columns.state)

# Execute the statement and store all the records: results

results = connection.execute(stmt).fetchall()

# Print results

print(results)

# Print the keys/column names of the results returned

print(results[0].keys())

==

# import pandas

import pandas as pd

# Create a DataFrame from the results: df

df = pd.DataFrame(results)

# Set column names

df.columns = results[0].keys()

# Print the Dataframe

print(df)

==

# Import pyplot as plt from matplotlib

from matplotlib import pyplot as plt

# Create a DataFrame from the results: df

df = pd.DataFrame(results)

# Set Column names

df.columns = results[0].keys()

# Print the DataFrame

print(df)

# Plot the DataFrame

df.plot.bar()

plt.show()

==

# Import create\_engine function

from sqlalchemy import create\_engine

# Create an engine to the census database

engine = create\_engine('mysql+pymysql://'+'student:datacamp'+'@courses.csrrinzqubik.us-east-1.rds.amazonaws.com:3306/'+'census')

# Print the table names

print(engine.table\_names())

==

# Build query to return state names by population difference from 2008 to 2000: stmt

stmt = select([census.columns.state, (census.columns.pop2008-census.columns.pop2000).label('pop\_change')])

# Append group by for the state: stmt

stmt = stmt.group\_by(census.columns.state)

# Append order by for pop\_change descendingly: stmt

stmt = stmt.order\_by(desc('pop\_change'))

# Return only 5 results: stmt

stmt = stmt.limit(5)

# Use connection to execute the statement and fetch all results

results = connection.execute(stmt).fetchall()

# Print the state and population change for each record

for result in results:

print('{}:{}'.format(result.state, result.pop\_change))

==

# import case, cast and Float from sqlalchemy

from sqlalchemy import case, cast, Float

# Build an expression to calculate female population in 2000

female\_pop2000 = func.sum(

case([

(census.columns.sex == 'F', census.columns.pop2000)

], else\_=0))

# Cast an expression to calculate total population in 2000 to Float

total\_pop2000 = cast(func.sum(census.columns.pop2000), Float)

# Build a query to calculate the percentage of females in 2000: stmt

stmt = select([female\_pop2000 / total\_pop2000 \* 100])

# Execute the query and store the scalar result: percent\_female

percent\_female = connection.execute(stmt).scalar()

# Print the percentage

print(percent\_female)

==

# Build a statement to join census and state\_fact tables: stmt

stmt = select([census.columns.pop2000, state\_fact.columns.abbreviation])

# Execute the statement and get the first result: result

result = connection.execute(stmt).first()

# Loop over the keys in the result object and print the key and value

for key in result.keys():

print(key, getattr(result, key))

==

# Build a statement to select the census and state\_fact tables: stmt

stmt = select([census, state\_fact])

# Add a select\_from clause that wraps a join for the census and state\_fact

# tables where the census state column and state\_fact name column match

stmt = stmt.select\_from(

census.join(state\_fact, census.columns.state == state\_fact.columns.name))

# Execute the statement and get the first result: result

result = connection.execute(stmt).first()

# Loop over the keys in the result object and print the key and value

for key in result.keys():

print(key, getattr(result, key))

==

# Build a statement to select the state, sum of 2008 population and census

# division name: stmt

stmt = select([

census.columns.state,

func.sum(census.columns.pop2008),

state\_fact.columns.census\_division\_name

])

# Append select\_from to join the census and state\_fact tables by the census state and state\_fact name columns

stmt = stmt.select\_from(

census.join(state\_fact, census.columns.state == state\_fact.columns.name)

)

# Append a group by for the state\_fact name column

stmt = stmt.group\_by(state\_fact.columns.name)

# Execute the statement and get the results: results

results = connection.execute(stmt).fetchall()

# Loop over the the results object and print each record.

for record in results:

print(record)

==

# Make an alias of the employees table: managers

managers = employees.alias()

# Build a query to select manager's and their employees names: stmt

stmt = select(

[managers.columns.name.label('manager'),

employees.columns.name.label('employee')]

)

# Match managers id with employees mgr: stmt

stmt = stmt.where(managers.columns.id == employees.columns.mgr)

# Order the statement by the managers name: stmt

stmt = stmt.order\_by(managers.columns.name)

# Execute statement: results

results = connection.execute(stmt).fetchall()

# Print records

for record in results:

print(record)

==

# Make an alias of the employees table: managers

managers = employees.alias()

# Build a query to select managers and counts of their employees: stmt

stmt = select([managers.columns.name, func.count(employees.columns.id)])

# Append a where clause that ensures the manager id and employee mgr are equal

stmt = stmt.where(managers.columns.id == employees.columns.mgr)

# Group by Managers Name

stmt = stmt.group\_by(managers.columns.name)

# Execute statement: results

results = connection.execute(stmt).fetchall()

# print manager

for record in results:

print(record)

==

# Start a while loop checking for more results

while more\_results:

# Fetch the first 50 results from the ResultProxy: partial\_results

partial\_results = results\_proxy.fetchmany(50)

# if empty list, set more\_results to False

if partial\_results == []:

more\_results = False

# Loop over the fetched records and increment the count for the state

for row in partial\_results:

if row.state in state\_count:

state\_count[row.state] +=1

else:

state\_count[row.state] = 1

# Close the ResultProxy, and thus the connection

results\_proxy.close()

# Print the count by state

print(state\_count)

==

# Import Table, Column, String, Integer, Float, Boolean from sqlalchemy

from sqlalchemy import Table, Column, String, Integer, Float, Boolean

# Define a new table with a name, count, amount, and valid column: data

data = Table('data', metadata,

Column('name', String(255)),

Column('count', Integer()),

Column('amount', Float()),

Column('valid', Boolean())

)

# Use the metadata to create the table

metadata.create\_all(engine)

# Print table details

print(repr(data))

==

# Import Table, Column, String, Integer, Float, Boolean from sqlalchemy

from sqlalchemy import Table, Column, String, Integer, Float, Boolean

# Define a new table with a name, count, amount, and valid column: data

data = Table('data', metadata,

Column('name', String(255), unique=True),

Column('count', Integer(), default=1),

Column('amount', Float()),

Column('valid', Boolean(), default=False)

)

# Use the metadata to create the table

metadata.create\_all(engine)

# Print the table details

print(repr(metadata.tables['data']))

==

# Import insert and select from sqlalchemy

from sqlalchemy import insert, select

# Build an insert statement to insert a record into the data table: stmt

stmt = insert(data).values(name='Anna', count=1, amount=1000.00, valid=True)

# Execute the statement via the connection: results

results = connection.execute(stmt)

# Print result rowcount

print(results.rowcount)

# Build a select statement to validate the insert

stmt = select([data]).where(data.columns.name == 'Anna')

# Print the result of executing the query.

print(connection.execute(stmt).first())

==

# Build a list of dictionaries: values\_list

values\_list = [

{'name': 'Anna', 'count': 1, 'amount': 1000.00, 'valid': True},

{'name': 'Taylor', 'count': 1, 'amount': 750.00, 'valid': False}

]

# Build an insert statement for the data table: stmt

stmt = insert(data)

# Execute stmt with the values\_list: results

results = connection.execute(stmt, values\_list)

# Print rowcount

print(results.rowcount)

==

# Create a insert statement for census: stmt

stmt = insert(census)

# Create an empty list and zeroed row count: values\_list, total\_rowcount

values\_list = []

total\_rowcount = 0

# Enumerate the rows of csv\_reader

for idx, row in enumerate(csv\_reader):

#create data and append to values\_list

data = {'state': row[0], 'sex': row[1], 'age': row[2], 'pop2000': row[3],

'pop2008': row[4]}

values\_list.append(data)

# Check to see if divisible by 51

if idx % 51 == 0:

results = connection.execute(stmt, values\_list)

total\_rowcount += results.rowcount

values\_list = []

# Print total rowcount

print(total\_rowcount)

==

# Build a select statement: select\_stmt

select\_stmt = select([state\_fact]).where(state\_fact.columns.name == 'New York')

# Print the results of executing the select\_stmt

print(connection.execute(select\_stmt).fetchall())

# Build a statement to update the fips\_state to 36: stmt

stmt = update(state\_fact).values(fips\_state = 36)

# Append a where clause to limit it to records for New York state

stmt = stmt.where(state\_fact.columns.name == 'New York')

# Execute the statement: results

results = connection.execute(stmt)

# Print rowcount

print(results.rowcount)

# Execute the select\_stmt again to view the changes

print(connection.execute(select\_stmt).fetchall())

==

# Build a statement to update the notes to 'The Wild West': stmt

stmt = update(state\_fact).values(notes='The Wild West')

# Append a where clause to match the West census region records

stmt = stmt.where(state\_fact.columns.census\_region\_name == 'West')

# Execute the statement: results

results = connection.execute(stmt)

# Print rowcount

print(results.rowcount)

==

# Build a statement to select name from state\_fact: stmt

fips\_stmt = select([state\_fact.columns.name])

# Append a where clause to Match the fips\_state to flat\_census fips\_code

fips\_stmt = fips\_stmt.where(state\_fact.columns.fips\_state == flat\_census.columns.fips\_code)

# Build an update statement to set the name to fips\_stmt: update\_stmt

update\_stmt = update(flat\_census).values(state\_name=fips\_stmt)

# Execute update\_stmt: results

results = connection.execute(update\_stmt)

# Print rowcount

print(results.rowcount)

==

# Import delete, select

from sqlalchemy import delete, select

# Build a statement to empty the census table: stmt

stmt = delete(census)

# Execute the statement: results

results = connection.execute(stmt)

# Print affected rowcount

print(results.rowcount)

# Build a statement to select all records from the census table

stmt = select([census])

# Print the results of executing the statement to verify there are no rows

print(connection.execute(stmt).fetchall())

==

# Build a statement to count records using the sex column for Men ('M') age 36: stmt

stmt = select([func.count(census.columns.sex)]).where(

and\_(census.columns.sex == 'M',

census.columns.age == 36)

)

# Execute the select statement and use the scalar() fetch method to save the record count

to\_delete = connection.execute(stmt).scalar()

# Build a statement to delete records from the census table: stmt\_del

stmt\_del = delete(census)

# Append a where clause to target Men ('M') age 36

stmt\_del = stmt\_del.where(

and\_(census.columns.sex == 'M',

census.columns.age == 36)

)

# Execute the statement: results

results = connection.execute(stmt\_del)

# Print affected rowcount and to\_delete record count, make sure they match

print(results.rowcount, to\_delete)

==

# Drop the state\_fact table

state\_fact.drop(engine)

# Check to see if state\_fact exists

print(state\_fact.exists(engine))

# Drop all tables

metadata.drop\_all(engine)

# Check to see if census exists

print(census.exists(engine))

==

# Import create\_engine, MetaData

from sqlalchemy import create\_engine, MetaData

# Define an engine to connect to chapter5.sqlite: engine

engine = create\_engine('sqlite:///chapter5.sqlite')

# Initialize MetaData: metadata

metadata = MetaData()

==

# Import Table, Column, String, and Integer

from sqlalchemy import (Table, Column, String, Integer)

# Build a census table: census

census = Table('census', metadata,

Column('state', String(30)),

Column('sex', String(1)),

Column('age', Integer()),

Column('pop2000', Integer()),

Column('pop2008', Integer()))

# Create the table in the database

metadata.create\_all(engine)

==

# Create an empty list: values\_list

values\_list = []

# Iterate over the rows

for row in csv\_reader:

# Create a dictionary with the values

data = {'state': row[0], 'sex': row[1], 'age':row[2], 'pop2000': row[3],

'pop2008': row[4]}

# Append the dictionary to the values list

values\_list.append(data)

==

# Import insert

from sqlalchemy import insert

# Build insert statement: stmt

stmt = insert(census)

# Use values\_list to insert data: results

results = connection.execute(stmt, values\_list)

# Print rowcount

print(results.rowcount)

==

# Import select

from sqlalchemy import select

# Calculate weighted average age: stmt

stmt = select([census.columns.sex,

(func.sum(census.columns.pop2008 \* census.columns.age) /

func.sum(census.columns.pop2008)).label('average\_age')

])

# Group by sex

stmt = stmt.group\_by(census.columns.sex)

# Execute the query and store the results: results

results = connection.execute(stmt).fetchall()

# Print the average age by sex

for res in results:

print(res.sex, res.average\_age)

==

# import case, cast and Float from sqlalchemy

from sqlalchemy import case, cast, Float

# Build a query to calculate the percentage of females in 2000: stmt

stmt = select([census.columns.state,

(func.sum(

case([

(census.columns.sex == 'F', census.columns.pop2000)

], else\_=0)) /

cast(func.sum(census.columns.pop2000), Float) \* 100).label('percent\_female')

])

# Group By state

stmt = stmt.group\_by(census.columns.state)

# Execute the query and store the results: results

results = connection.execute(stmt).fetchall()

# Print the percentage

for result in results:

print(result.state, result.percent\_female)

==

# Build query to return state name and population difference from 2008 to 2000

stmt = select([census.columns.state,

(census.columns.pop2008-census.columns.pop2000).label('pop\_change')

])

# Group by State

stmt = stmt.group\_by(census.columns.state)

# Order by Population Change

stmt = stmt.order\_by(desc('pop\_change'))

# Limit to top 10

stmt = stmt.limit(10)

# Use connection to execute the statement and fetch all results

results = connection.execute(stmt).fetchall()

# Print the state and population change for each record

for result in results:

print('{}:{}'.format(result.state, result.pop\_change))

==