

# Reflect the census table from the engine: census

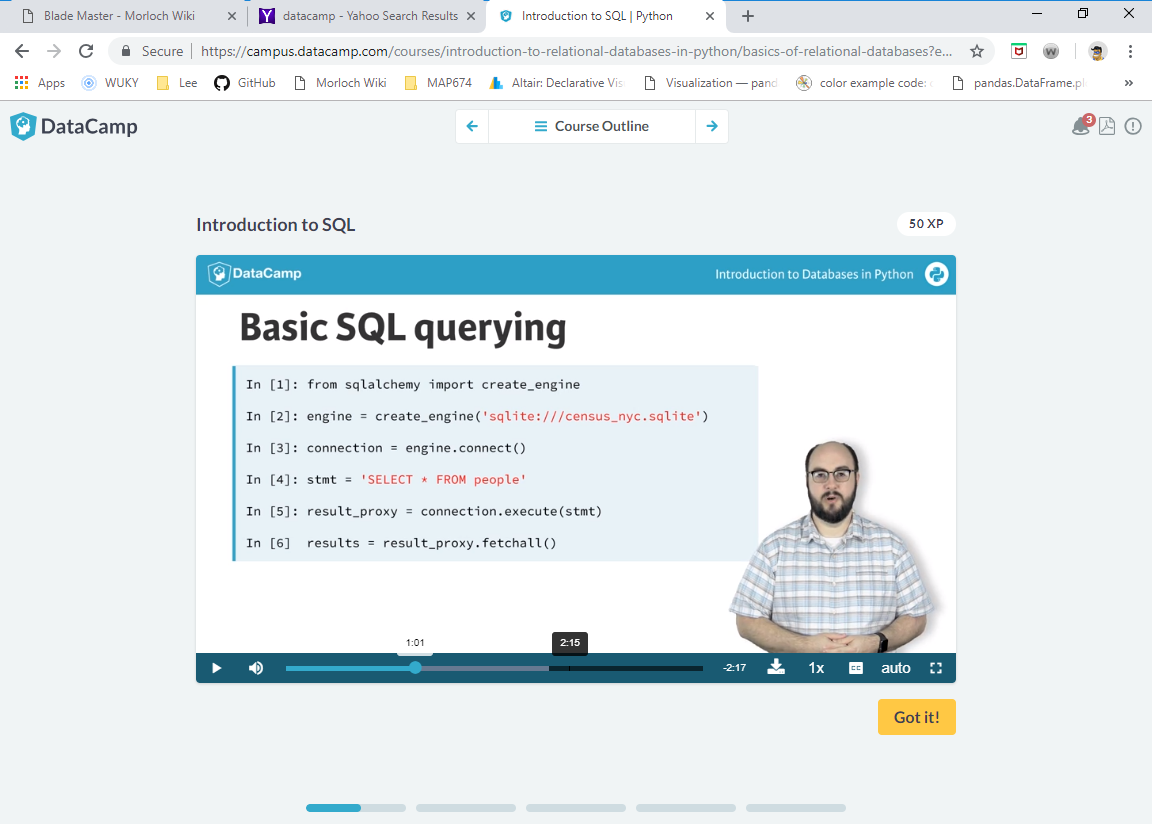
census = Table('census', metadata, autoload=True, autoload\_with=engine)

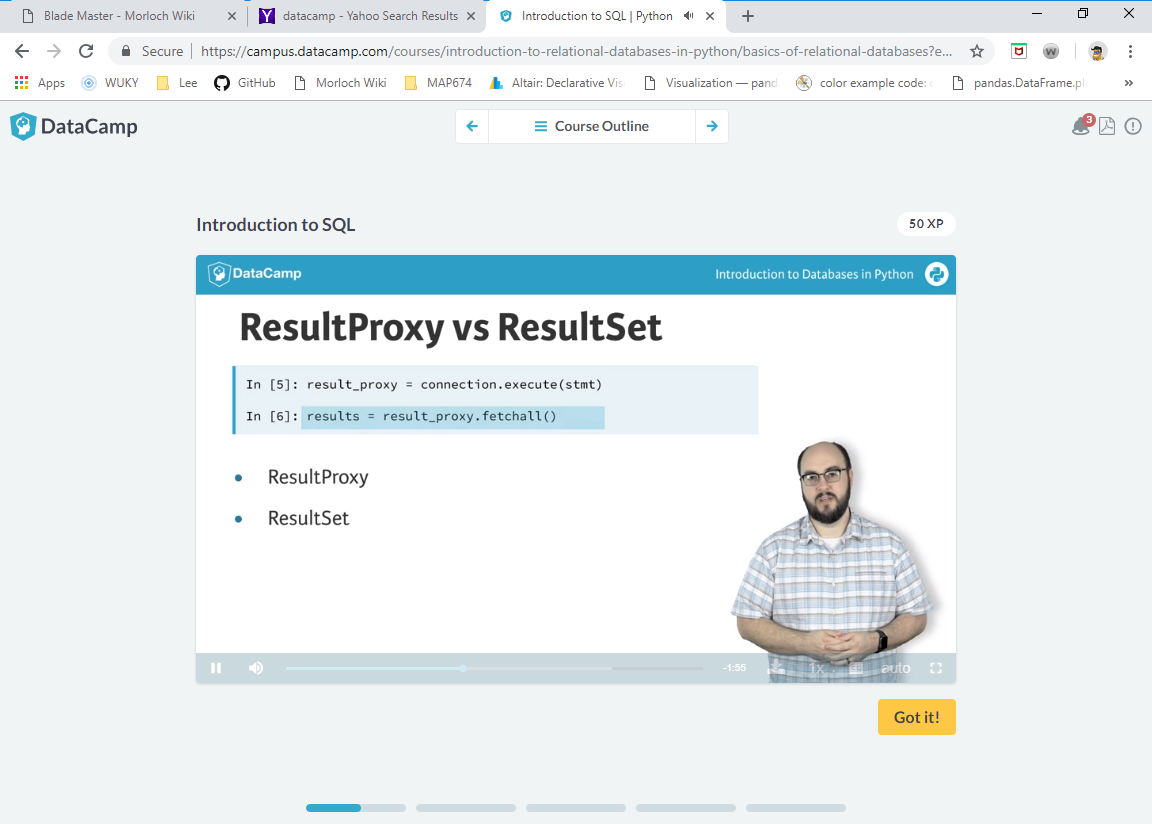
# Print the column names

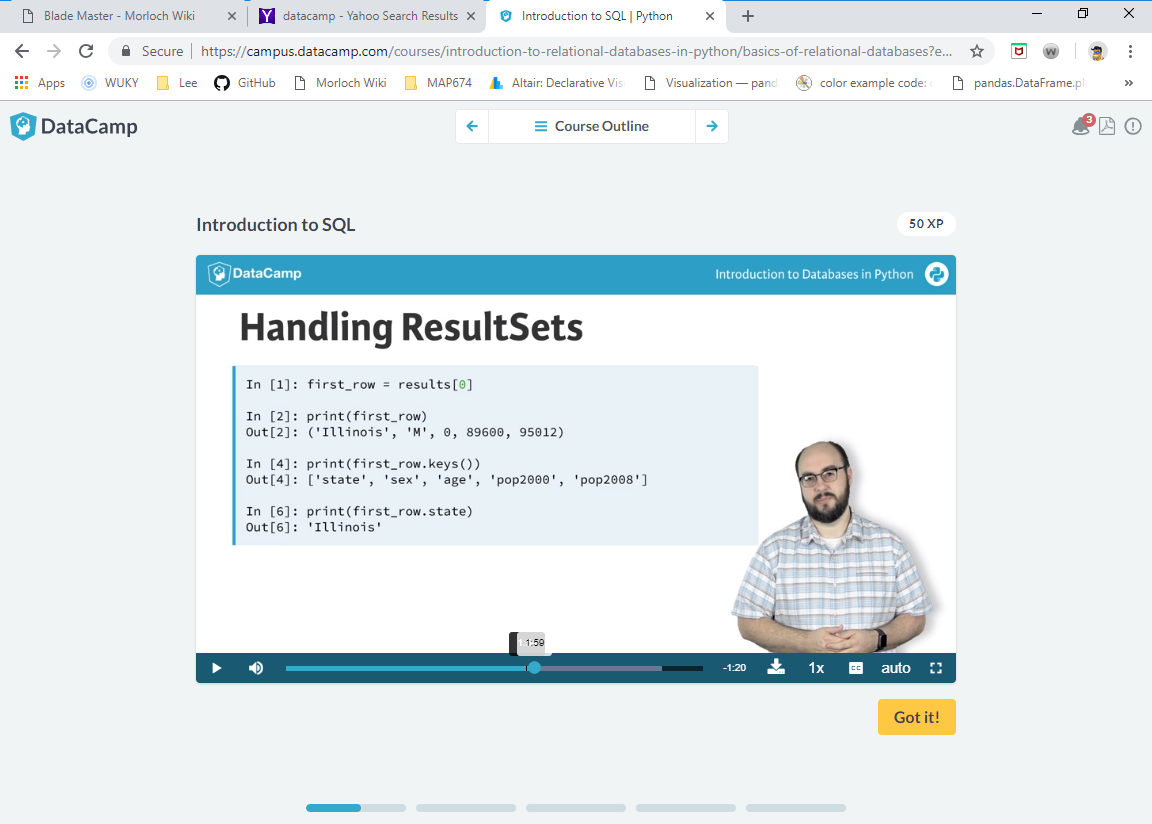
print(census.columns.keys())

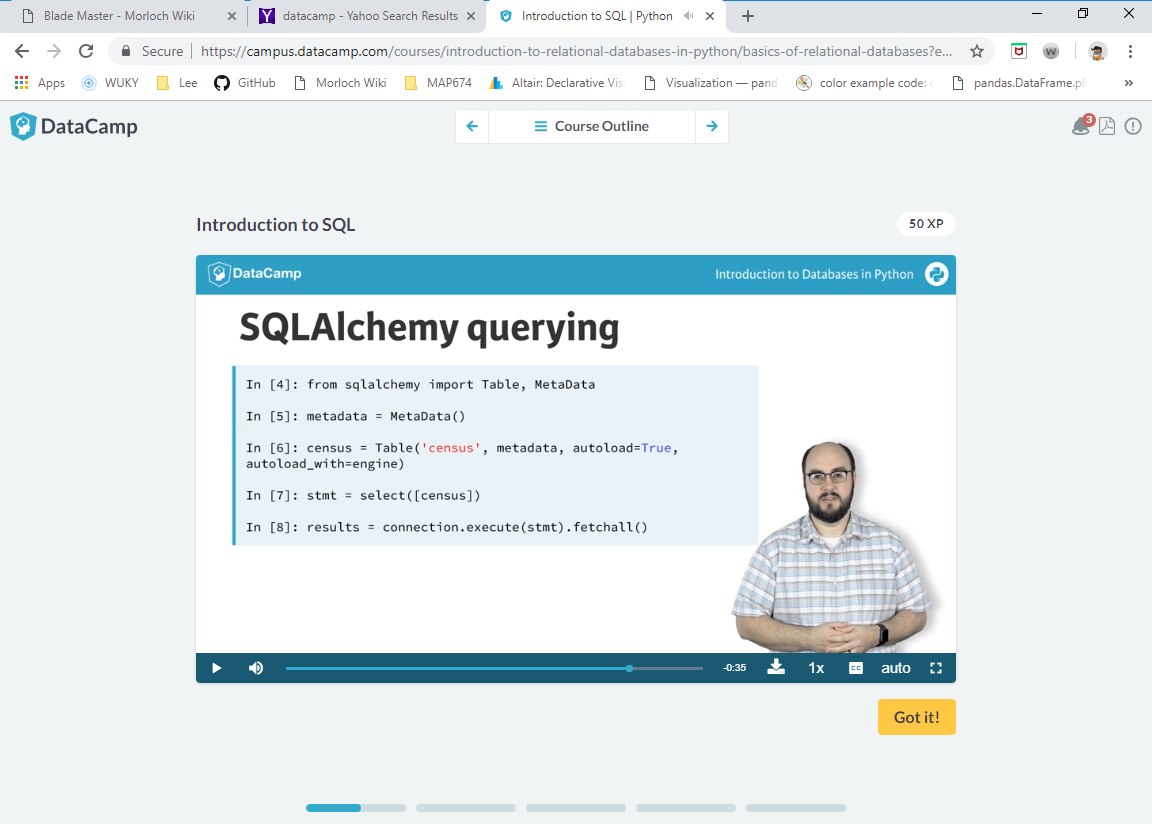
# Print full table metadata

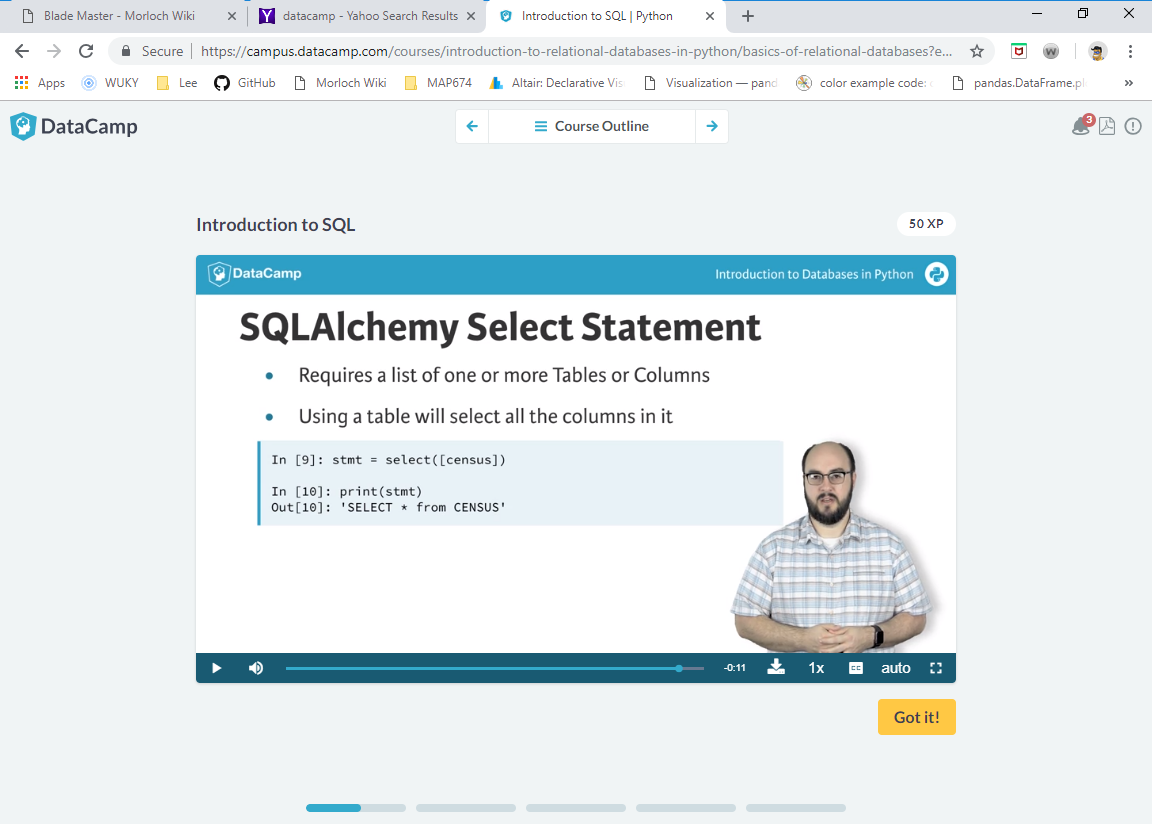
print(repr(metadata.tables['census']))











# 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)

**Selecting data from a Table with SQLAlchemy**

# 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())

# Handling a ResultSet

Recall the differences between a ResultProxy and a ResultSet:

* ResultProxy: The object returned by the .execute() method. It can be used in a variety of ways to get the data returned by the query.
* ResultSet: The actual data asked for in the query when using a fetch method such as .fetchall() on a ResultProxy.

This separation between the ResultSet and ResultProxy allows us to fetch as much or as little data as we desire.

Once we have a ResultSet, we can use Python to access all the data within it by column name and by list style indexes. For example, you can get the first row of the results by using results[0]. With that first row then assigned to a variable first\_row, you can get data from the first column by either using first\_row[0] or by column name such as first\_row['column\_name']. You'll now practice exactly this using the ResultSet you obtained from the census table in the previous exercise. It is stored in the variable results. Enjoy!