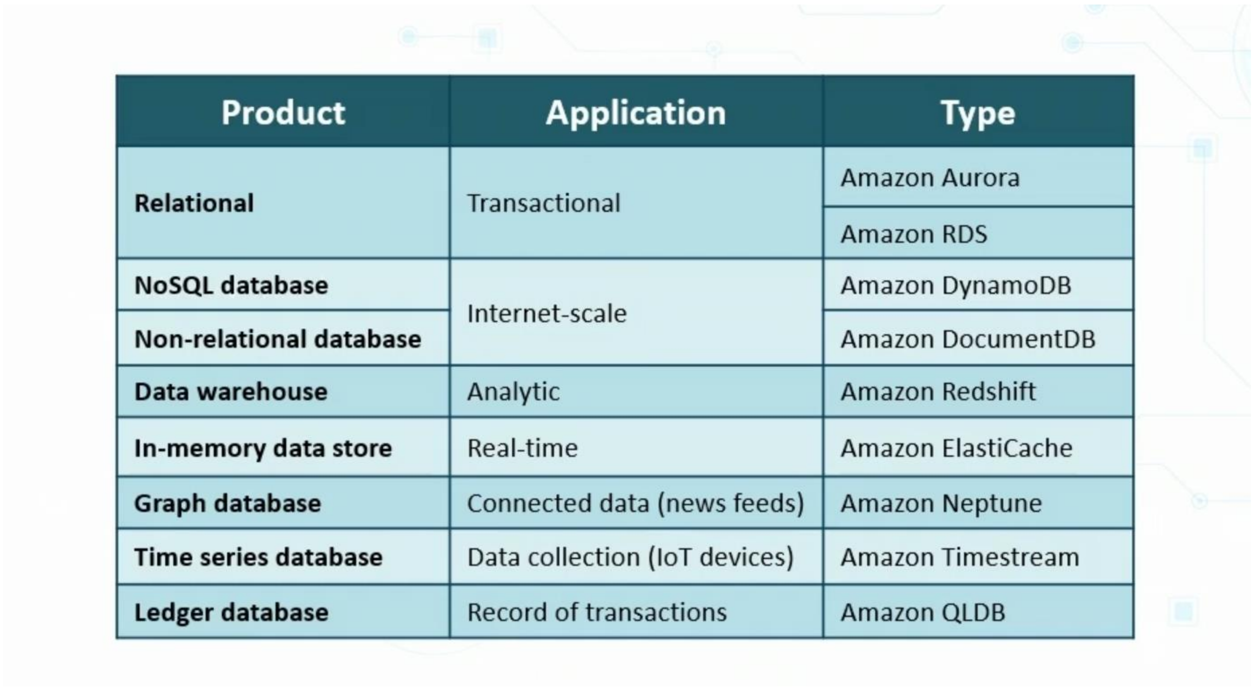
1.8 AWS Database Services

AWS Cloud Database Offerings

- AWS Cloud
  - Databases
    - Amazon Elastic Compute Cloud (EC2)
    - AWS Database Migration Service (DMS)

- AWS Databases
  - Data Warehouse databases
  - Graph databases
  - In-memory data store databases
  - Ledger databases: cryptographically verifiable history of transactions
  - Managed relational databases
  - Non-relational databases
  - Time-series databases



Product	Application	Type
Relational	Transactional	Amazon Aurora
		Amazon RDS
NoSQL database	Internet-scale	Amazon DynamoDB
Non-relational database		Amazon DocumentDB
Data warehouse	Analytic	Amazon Redshift
In-memory data store	Real-time	Amazon ElastiCache
Graph database	Connected data (news feeds)	Amazon Neptune
Time series database	Data collection (IoT devices)	Amazon Timestream
Ledger database	Record of transactions	Amazon QLDB

Relational Databases

- Collection of data items
- Predefined relationships
- Organized as sets of tables
- Store information about objects

Relational Database Characteristics

- Structured query language (SQL)
- Data integrity
- Transactions: atomic, executed as one single unit, commit or rollback
- Atomic, consistent, isolated, and durable (ACID)

Relational Database

- Managed: Amazon RDS
- Unmanaged

Amazon Relational Database Service (RDS)

- Procurement
- Configuration
- Backup
- Security
- Availability

Amazon RDS: Relational database engines

- Amazon Aurora
- Oracle
- Microsoft SQL Server
- PostgreSQL
- MySQL
- MariaDB

Amazon RDS features

- Software patching

- Vertical scaling: for right intensive applications
- Storage scaling: 3 types – General purpose SSD, provisioned IOPS and magnetic storage
- Horizontal scaling: read replicas, asynchronous

RDS, MySQL, PostgreSQL – 5 read replicas

Amazon Aurora – 15 read replicas

RDS oracle and SQL server – do not support

- Backups: point-to-time, snapshots… 7 to 35 days
- Snapshots
- Multi-AZ deployments
- Encryption: SSL
- IAM authentication
- Amazon CloudWatch: to notify

Nonrelational Databases

NoSQL Database

- Optimized for scalable performance
- Schema-less data models
- Ease of deployment
- Low latency
- Resilience

Reviewing SQL versus NoSQL

- Databases reveals that for data storage, SQL uses rows and columns. NoSQL uses key value pairs, documents, wide column and graph objects.
- For schemas SQL is fixed. NoSQL is dynamic.
- For query we use Structured Query Language with SQL databases and with NoSQL databases, we are more focused on a collection of documents.
- With respect to scalability, SQL databases scale very well vertically, while in NoSQL they scale very well horizontally.
- SQL supports transactions. With NoSQL that support can vary. So, it's very flexible.

- For consistency, with the SQL database, we have strong consistency, but with NoSQL database, we can have eventual and strong consistency. So again, quite flexible.

Using NoSQL Databases

- Big data
- Mobile
- Web applications

Types of NoSQL databases

- Columnar: read and write columns of data
- Documents: semi-structured, JSON and XML
- Graph: vertices and edges
- In-memory key-value: very low latency, used for heavy workloads

Amazon DynamoDB

- Fully managed cloud database
- Fast and flexible
- Reliable performance
- Automatic scaling
- Consistent
- Can use global tables

Components of DynamoDB

- Tables
- Items
- Attributes
- Primary key

DynamoDB Secondary Index

- Primary key attributes
- Secondary key attributes
- Subset of other attributes (optional)
- Can define 5 local and global indexes per table

In-memory Data Stores

- Used for caching
- Real-time workloads
- In-memory key-value database options
  ~ Amazon EC2
  ~ Amazon EBS
  ~ Amazon Elasticache

Caching

- Improved performance
- Data retrieval
- Reduced loads
- Eliminates hotspots
- High throughput
- Low-latency access
- Data is stored in memory

Strategies of Caching

- Cache hit: when info is requested, contains info requested
- Cache miss: does not contain requested information
- Lazy loading: loading data into cache when necessary
- Write through: data is added to cache when written

Amazon In-memory Key-Value Data Stores

- In-memory stores
  - Efficient databases
  - Built to be scalable
- Distributed cache
  - More cost-effective
  - Faster performance

Amazon ElastiCache

- Redis
- Memcached

Amazon DynamoDB Accelerator

- Fully managed

- Highly available
- In-memory acceleration

Cloud Database Migration

- Used to migrate databases quickly and securely
- Source database remains fully operational
- Minimizes downtime

Using AWS DMS to perform Migration

- Loading existing data
- Applying any cached changes
- Performing ongoing replication

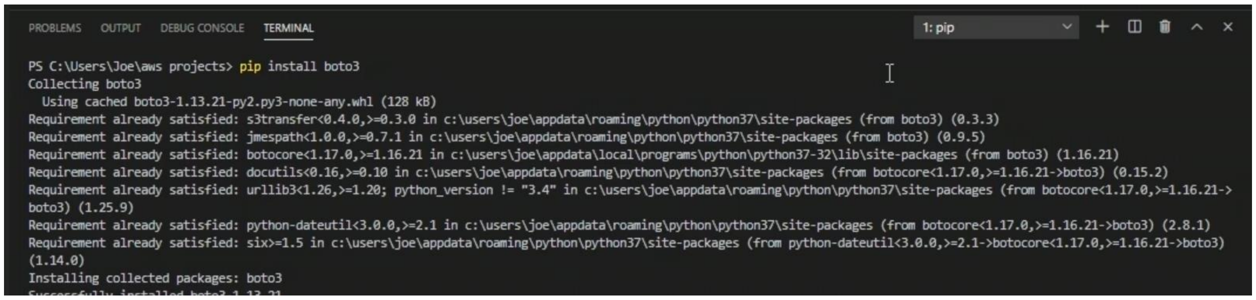
AWS Data Migration

- Homogenous:
- Heterogeneous: use AWS Schema Conversion Tool (SCT)
    - automatically converts database schema
    - scans application source code
    - performs cloud-native code optimization

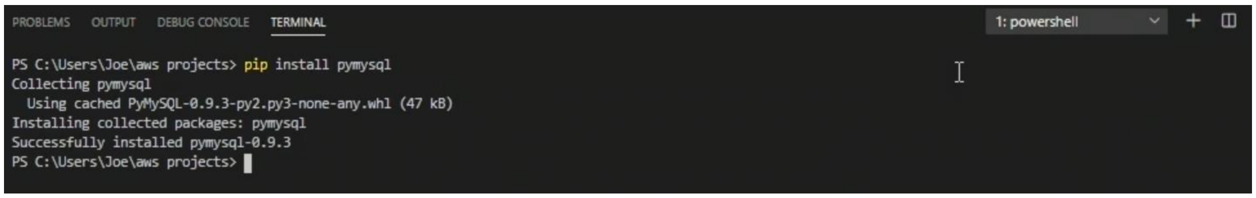
Installing the Python SDK and PyMySQL

Download latest version of python

Then install boto3



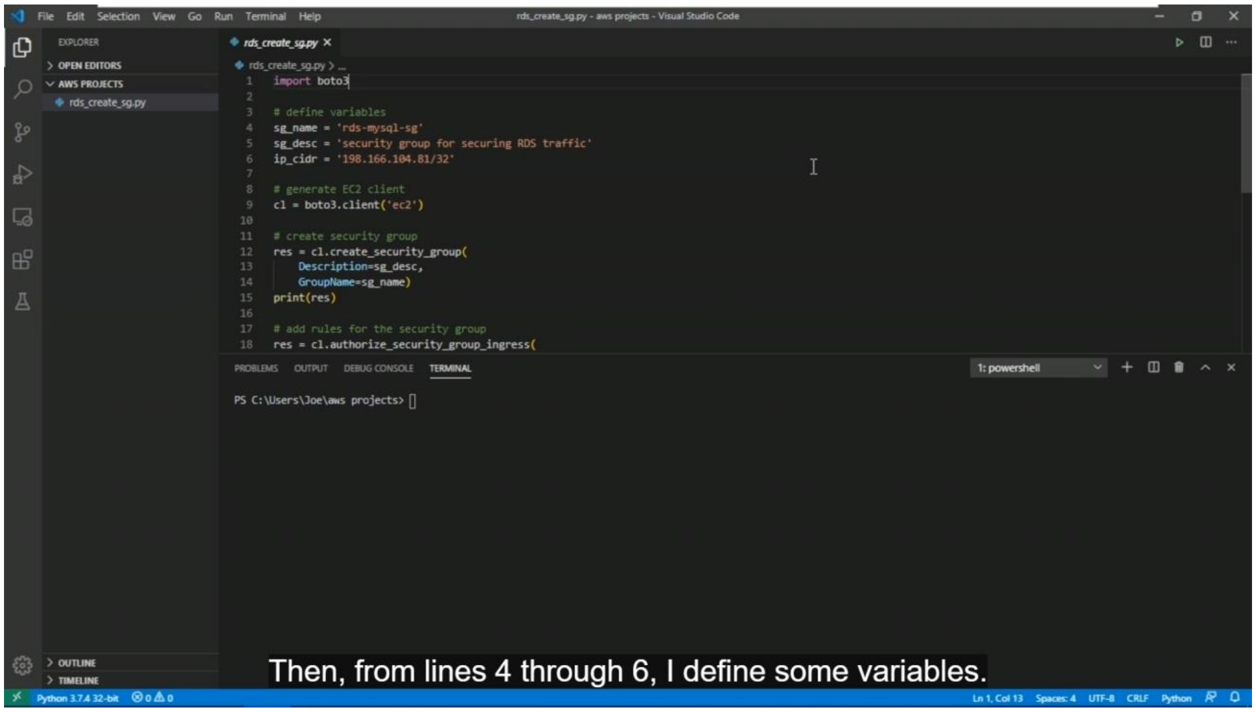
```
PS C:\Users\Joe\aws projects> pip install boto3
Collecting boto3
  Using cached boto3-1.13.21-py2.py3-none-any.whl (128 kB)
Requirement already satisfied: s3transfer<0.4.0,>=0.3.0 in c:\users\joe\appdata\roaming\python\python37\site-packages (from boto3) (0.3.3)
Requirement already satisfied: jmespath<1.0.0,>=0.7.1 in c:\users\joe\appdata\roaming\python\python37\site-packages (from boto3) (0.9.5)
Requirement already satisfied: botocore<1.17.0,>=1.16.21 in c:\users\joe\appdata\local\programs\python\python37-32\lib\site-packages (from boto3) (1.16.21)
Requirement already satisfied: docutils<0.16,>=0.10 in c:\users\joe\appdata\roaming\python\python37\site-packages (from botocore<1.17.0,>=1.16.21->boto3) (0.15.2)
Requirement already satisfied: urllib3<1.26,>=1.20; python_version != "3.4" in c:\users\joe\appdata\roaming\python\python37\site-packages (from botocore<1.17.0,>=1.16.21->
boto3) (1.25.9)
Requirement already satisfied: python-dateutil<3.0.0,>=2.1 in c:\users\joe\appdata\roaming\python\python37\site-packages (from botocore<1.17.0,>=1.16.21->boto3) (2.8.1)
Requirement already satisfied: six>=1.5 in c:\users\joe\appdata\roaming\python\python37\site-packages (from python-dateutil<3.0.0,>=2.1->botocore<1.17.0,>=1.16.21->boto3)
(1.14.0)
Installing collected packages: boto3
```



```
PS C:\Users\Joe\aws projects> pip install pymysql
Collecting pymysql
  Using cached PyMySQL-0.9.3-py2.py3-none-any.whl (47 kB)
Installing collected packages: pymysql
Successfully installed pymysql-0.9.3
PS C:\Users\Joe\aws projects>
```

pip= python packages

Generating Security Groups for Amazon RDS

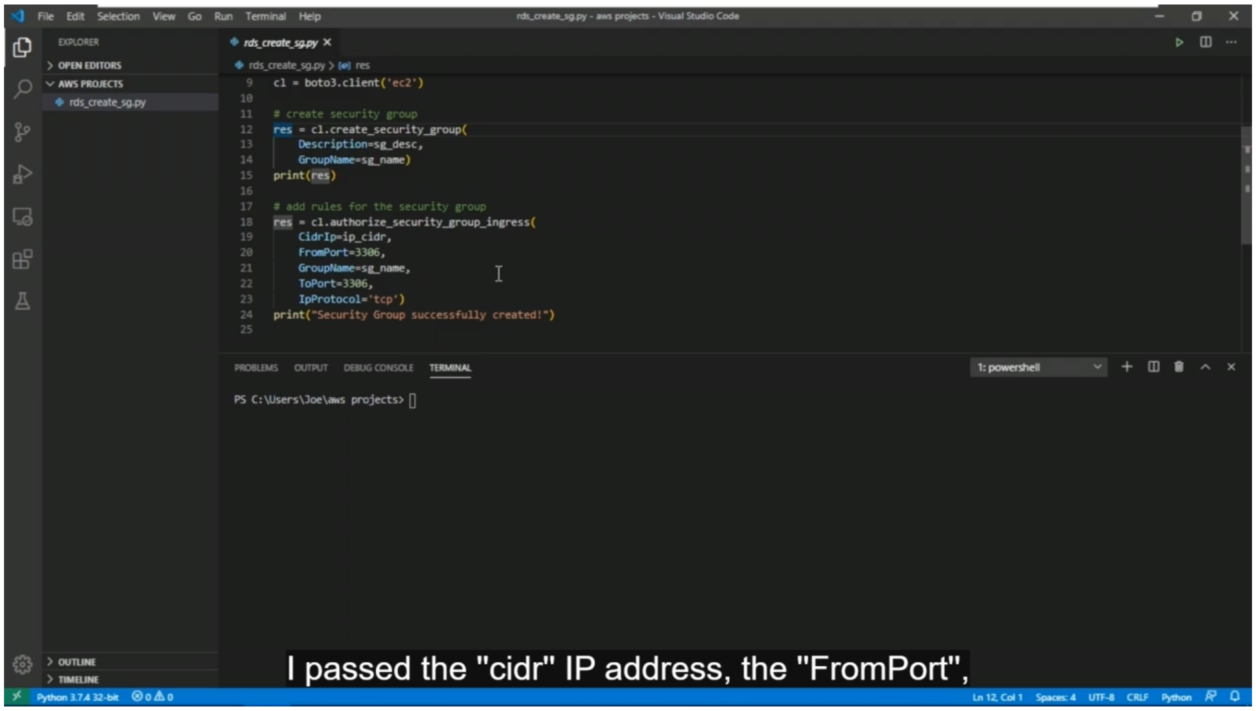


rds_create_sg.py - aws projects - Visual Studio Code

```python
1   import boto3
2
3   # define variables
4   sg_name = 'rds-mysql-sg'
5   sg_desc = 'security group for securing RDS traffic'
6   ip_cidr = '198.166.104.81/32'
7
8   # generate EC2 client
9   cl = boto3.client('ec2')
10
11  # create security group
12  res = cl.create_security_group(
13      Description=sg_desc,
14      GroupName=sg_name)
15  print(res)
16
17  # add rules for the security group
18  res = cl.authorize_security_group_ingress(
```

```
PS C:\Users\Joe\aws projects>
```

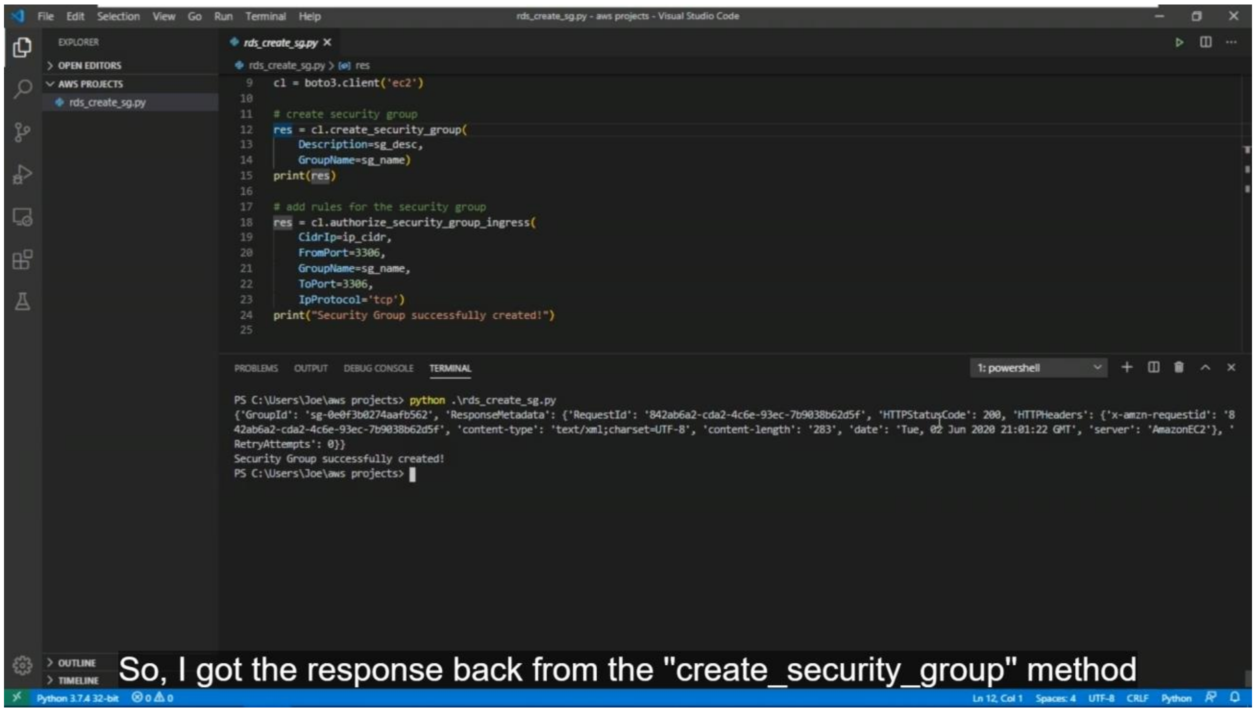
Then, from lines 4 through 6, I define some variables.



```
cl = boto3.client('ec2')

# create security group
res = cl.create_security_group(
    Description=sg_desc,
    GroupName=sg_name)
print(res)

# add rules for the security group
res = cl.authorize_security_group_ingress(
    CidrIp=ip_cidr,
    FromPort=3306,
    GroupName=sg_name,
    ToPort=3306,
    IpProtocol='tcp')
print("Security Group successfully created!")
```

```
PS C:\Users\Joe\aws projects>
```

I passed the "cidr" IP address, the "FromPort",



```
cl = boto3.client('ec2')

# create security group
res = cl.create_security_group(
    Description=sg_desc,
    GroupName=sg_name)
print(res)

# add rules for the security group
res = cl.authorize_security_group_ingress(
    CidrIp=ip_cidr,
    FromPort=3306,
    GroupName=sg_name,
    ToPort=3306,
    IpProtocol='tcp')
print("Security Group successfully created!")
```

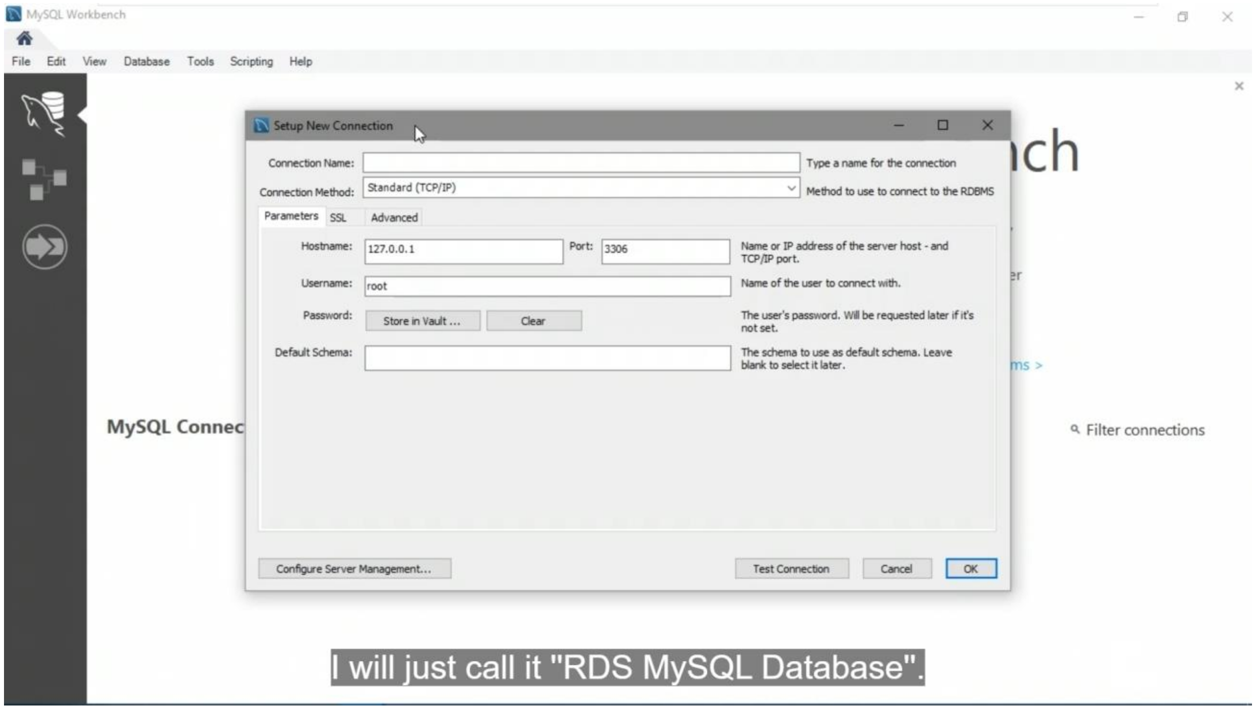
```
PS C:\Users\Joe\aws projects> python .\rds_create_sg.py
{'GroupId': 'sg-0e0f3b0274aafb562', 'ResponseMetadata': {'RequestId': '842ab6a2-cda2-4c6e-93ec-7b9038b62d5f', 'HTTPStatusCode': 200, 'HTTPHeaders': {'x-amzn-requestid': '8
42ab6a2-cda2-4c6e-93ec-7b9038b62d5f', 'content-type': 'text/xml;charset=UTF-8', 'content-length': '283', 'date': 'Tue, 02 Jun 2020 21:01:22 GMT', 'server': 'AmazonEC2'}, '
RetryAttempts': 0}}
Security Group successfully created!
PS C:\Users\Joe\aws projects>
```

So, I got the response back from the "create_security_group" method

Creating an Amazon RDS Database Instance

Connecting to a Database Using MySQL Workbench

Download MySQL workbench



Setup New Connection

Connection Name: — Type a name for the connection

Connection Method: Standard (TCP/IP) — Method to use to connect to the RDBMS

Parameters | SSL | Advanced

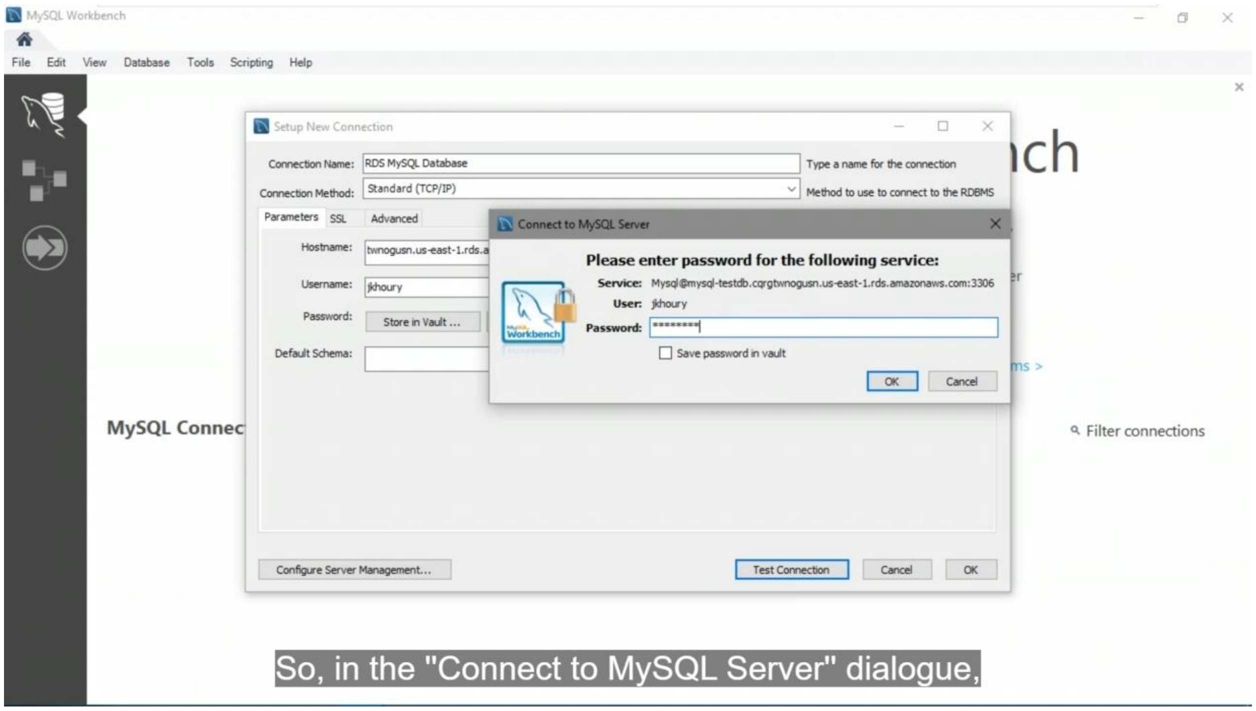
Hostname: 127.0.0.1 Port: 3306 — Name or IP address of the server host - and TCP/IP port.

Username: root — Name of the user to connect with.

Password: — The user's password. Will be requested later if it's not set.

Default Schema: — The schema to use as default schema. Leave blank to select it later.

I will just call it "RDS MySQL Database".



Setup New Connection

Connection Name: RDS MySQL Database — Type a name for the connection

Connection Method: Standard (TCP/IP) — Method to use to connect to the RDBMS

Parameters | SSL | Advanced

Hostname: twnogusn.us-east-1.rds.a

Username: jkhoury

Password:

Default Schema:

Connect to MySQL Server

Please enter password for the following service:

Service: Mysql@mysql-testdb.cqrgtwnogusn.us-east-1.rds.amazonaws.com:3306

User: jkhoury

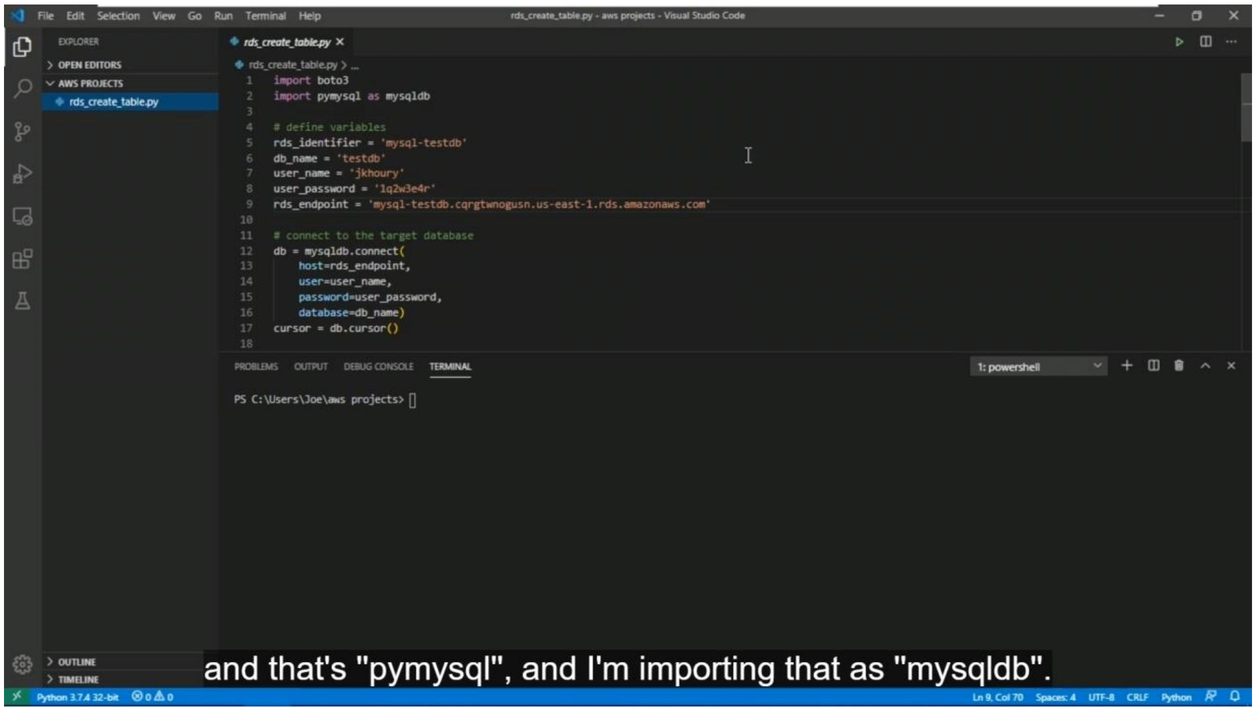
Password: ********

☐ Save password in vault

OK Cancel

So, in the "Connect to MySQL Server" dialogue,

Creating Tables in an Amazon RDS MySQL Database

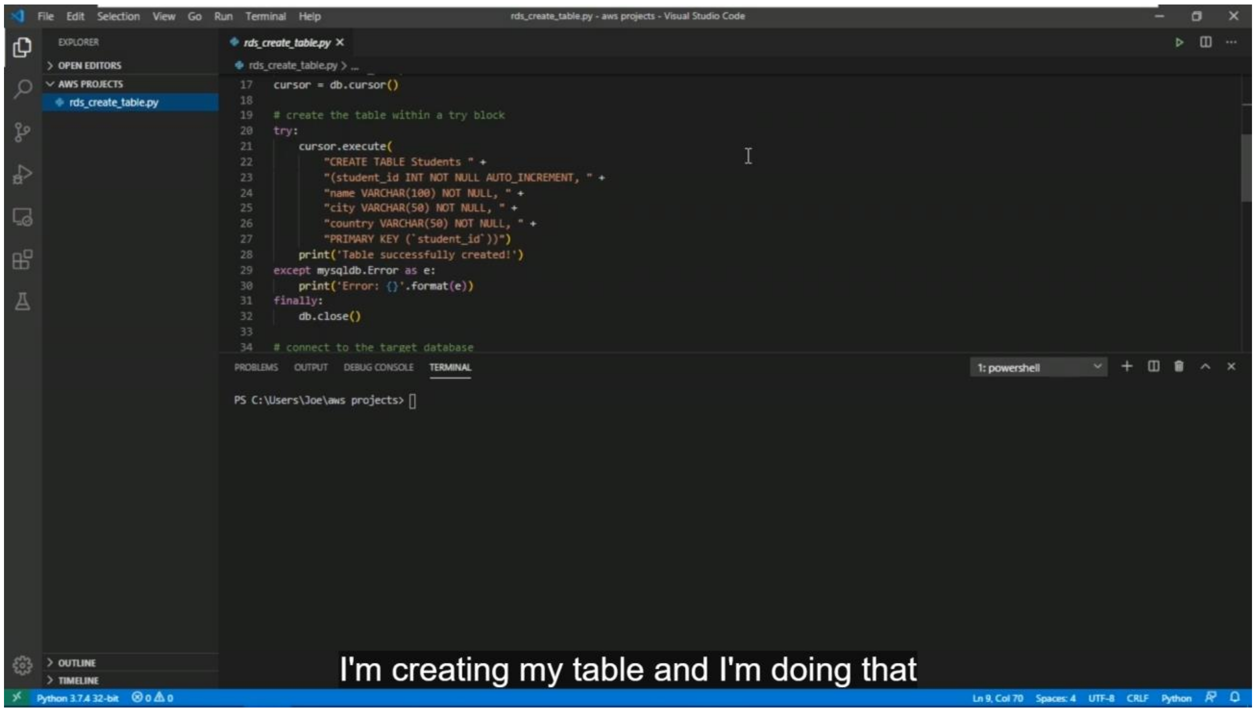


File Edit Selection View Go Run Terminal Help
rds_create_table.py - aws projects - Visual Studio Code
EXPLORER
OPEN EDITORS
AWS PROJECTS
rds_create_table.py
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
1: powershell
OUTLINE
TIMELINE
Python 3.7.4 32-bit
Ln 9, Col 70 Spaces: 4 UTF-8 CRLF Python

```python
import boto3
import pymysql as mysqldb

# define variables
rds_identifier = 'mysql-testdb'
db_name = 'testdb'
user_name = 'jkhoury'
user_password = '1q2w3e4r'
rds_endpoint = 'mysql-testdb.cqrgtwnogusn.us-east-1.rds.amazonaws.com'

# connect to the target database
db = mysqldb.connect(
    host=rds_endpoint,
    user=user_name,
    password=user_password,
    database=db_name)
cursor = db.cursor()

```

```
PS C:\Users\Joe\aws projects> 
```

and that's "pymysql", and I'm importing that as "mysqldb".

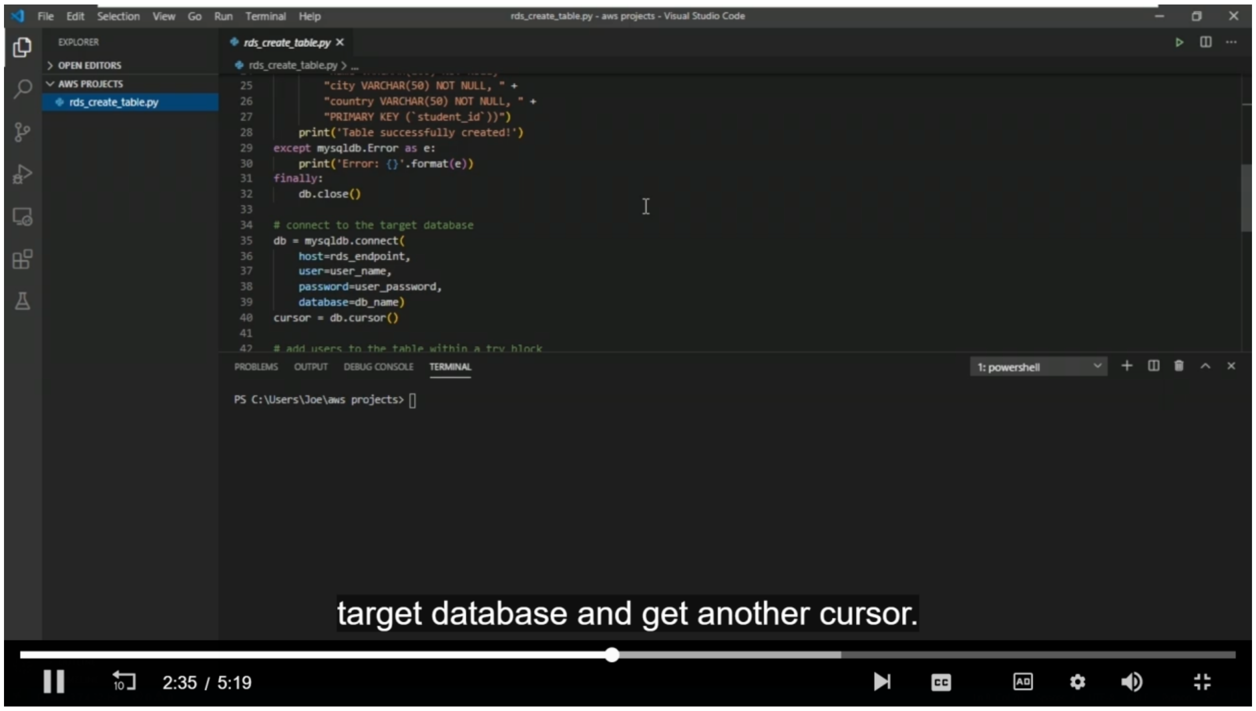


File Edit Selection View Go Run Terminal Help
rds_create_table.py - aws projects - Visual Studio Code
EXPLORER
OPEN EDITORS
AWS PROJECTS
rds_create_table.py
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
1: powershell
OUTLINE
TIMELINE
Python 3.7.4 32-bit
Ln 9, Col 70 Spaces: 4 UTF-8 CRLF Python

```python
cursor = db.cursor()

# create the table within a try block
try:
    cursor.execute(
        "CREATE TABLE Students " +
        "(student_id INT NOT NULL AUTO_INCREMENT, " +
        "name VARCHAR(100) NOT NULL, " +
        "city VARCHAR(50) NOT NULL, " +
        "country VARCHAR(50) NOT NULL, " +
        "PRIMARY KEY (`student_id`))")
    print('Table successfully created!')
except mysqldb.Error as e:
    print('Error: {}'.format(e))
finally:
    db.close()

# connect to the target database
```

```
PS C:\Users\Joe\aws projects> 
```

I'm creating my table and I'm doing that

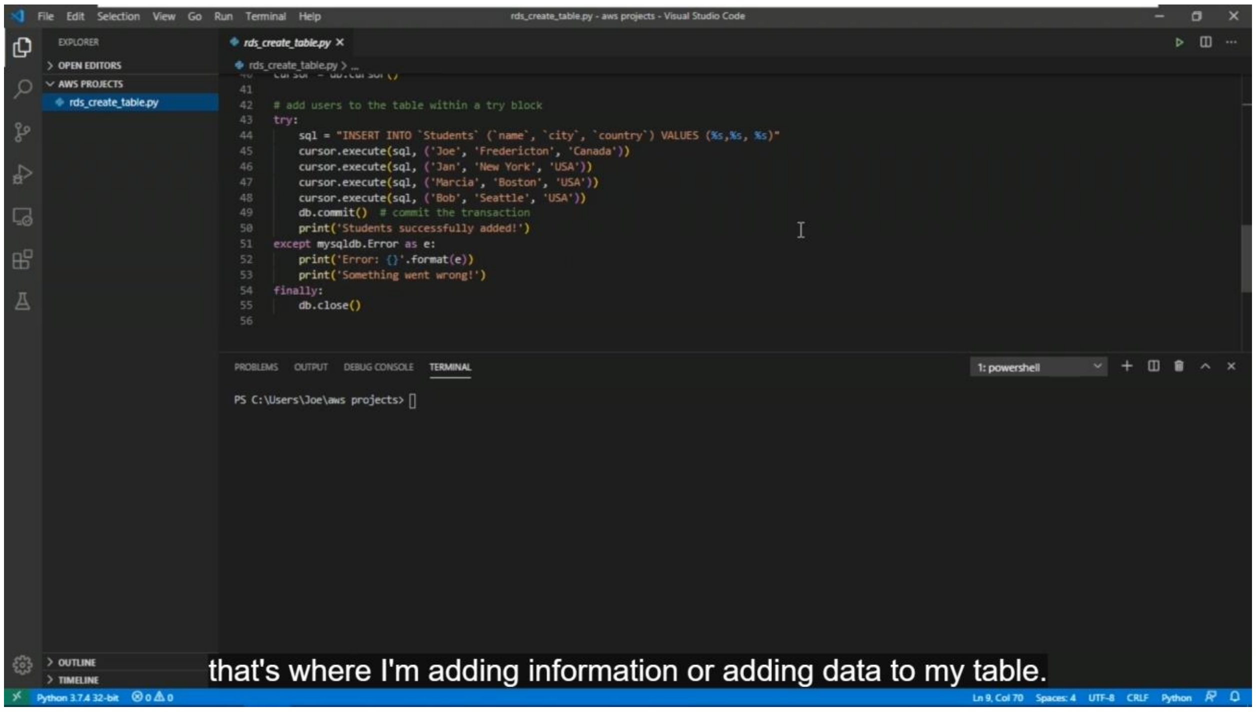


```python
        "city VARCHAR(50) NOT NULL, " +
        "country VARCHAR(50) NOT NULL, " +
        "PRIMARY KEY (`student_id`))")
    print('Table successfully created!')
except mysqldb.Error as e:
    print('Error: {}'.format(e))
finally:
    db.close()

# connect to the target database
db = mysqldb.connect(
    host=rds_endpoint,
    user=user_name,
    password=user_password,
    database=db_name)
cursor = db.cursor()
```

```
PS C:\Users\Joe\aws projects>
```

target database and get another cursor.

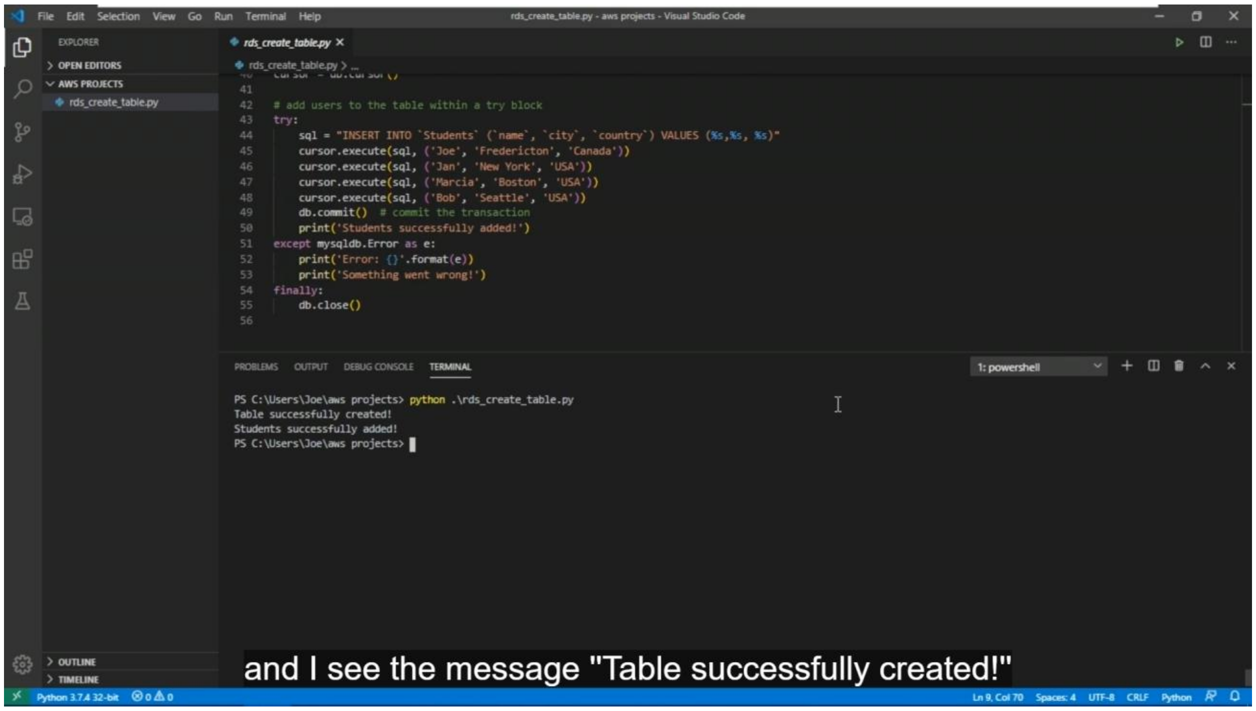
2:35 / 5:19



```python
# add users to the table within a try block
try:
    sql = "INSERT INTO `Students` (`name`, `city`, `country`) VALUES (%s,%s, %s)"
    cursor.execute(sql, ('Joe', 'Fredericton', 'Canada'))
    cursor.execute(sql, ('Jan', 'New York', 'USA'))
    cursor.execute(sql, ('Marcia', 'Boston', 'USA'))
    cursor.execute(sql, ('Bob', 'Seattle', 'USA'))
    db.commit()  # commit the transaction
    print('Students successfully added!')
except mysqldb.Error as e:
    print('Error: {}'.format(e))
    print('Something went wrong!')
finally:
    db.close()
```

```
PS C:\Users\Joe\aws projects>
```

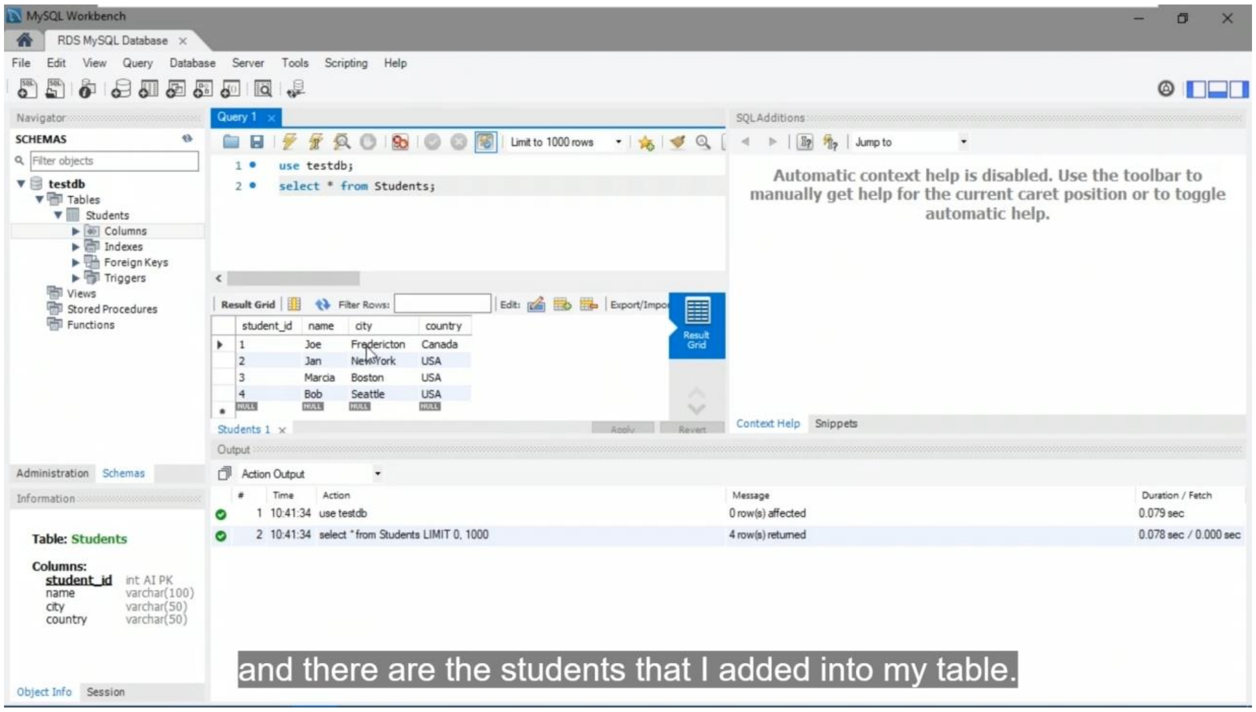
that's where I'm adding information or adding data to my table.



```python
# add users to the table within a try block
try:
    sql = "INSERT INTO `Students` (`name`, `city`, `country`) VALUES (%s,%s, %s)"
    cursor.execute(sql, ('Joe', 'Fredericton', 'Canada'))
    cursor.execute(sql, ('Jan', 'New York', 'USA'))
    cursor.execute(sql, ('Marcia', 'Boston', 'USA'))
    cursor.execute(sql, ('Bob', 'Seattle', 'USA'))
    db.commit()  # commit the transaction
    print('Students successfully added!')
except mysqldb.Error as e:
    print('Error: {}'.format(e))
    print('Something went wrong!')
finally:
    db.close()
```

```
PS C:\Users\Joe\aws projects> python .\rds_create_table.py
Table successfully created!
Students successfully added!
PS C:\Users\Joe\aws projects>
```

and I see the message "Table successfully created!"



```sql
use testdb;
select * from Students;
```

student_id	name	city	country
1	Joe	Fredericton	Canada
2	Jan	New York	USA
3	Marcia	Boston	USA
4	Bob	Seattle	USA
NULL	NULL	NULL	NULL

#	Time	Action	Message	Duration / Fetch
1	10:41:34	use testdb	0 row(s) affected	0.079 sec
2	10:41:34	select * from Students LIMIT 0, 1000	4 row(s) returned	0.078 sec / 0.000 sec

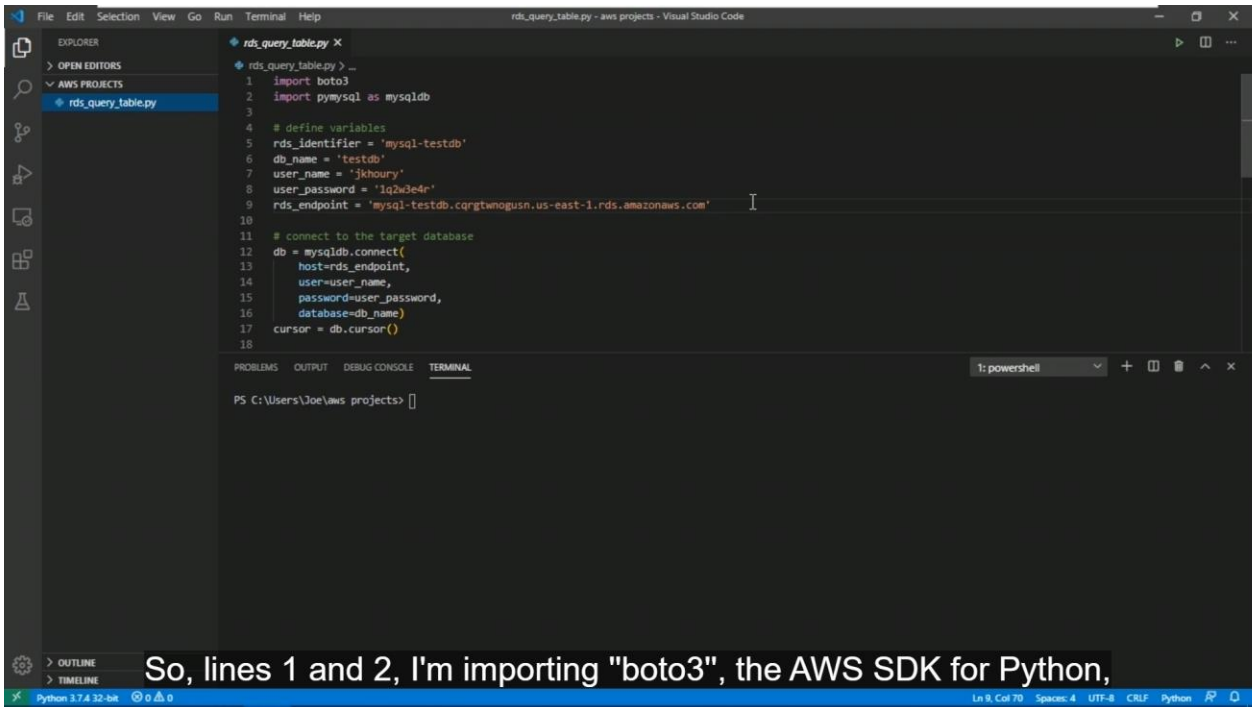
Table: Students

Columns:

student_id	int AI PK
name	varchar(100)
city	varchar(50)
country	varchar(50)

and there are the students that I added into my table.

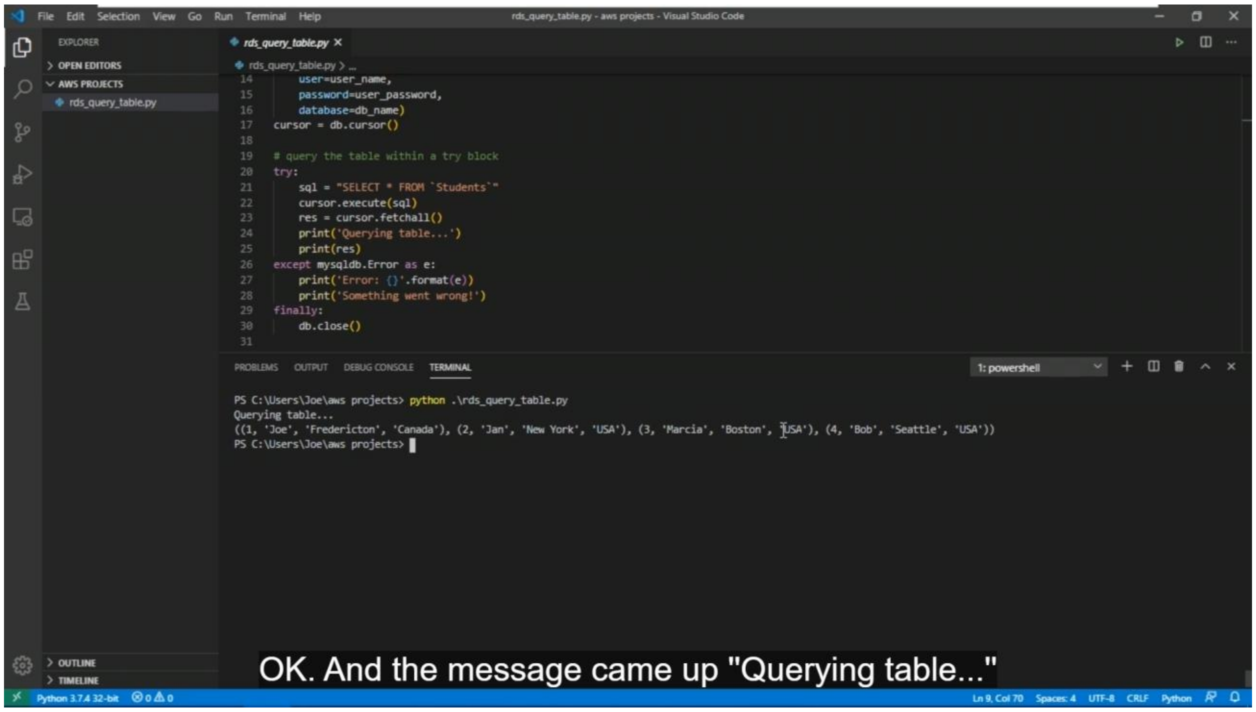
Querying Tables in an Amazon RDS MySQL Database



```python
1   import boto3
2   import pymysql as mysqldb
3
4   # define variables
5   rds_identifier = 'mysql-testdb'
6   db_name = 'testdb'
7   user_name = 'jkhoury'
8   user_password = '1q2w3e4r'
9   rds_endpoint = 'mysql-testdb.cqrgtwnogusn.us-east-1.rds.amazonaws.com'
10
11  # connect to the target database
12  db = mysqldb.connect(
13      host=rds_endpoint,
14      user=user_name,
15      password=user_password,
16      database=db_name)
17  cursor = db.cursor()
18
```

```
PS C:\Users\Joe\aws projects>
```

So, lines 1 and 2, I'm importing "boto3", the AWS SDK for Python,

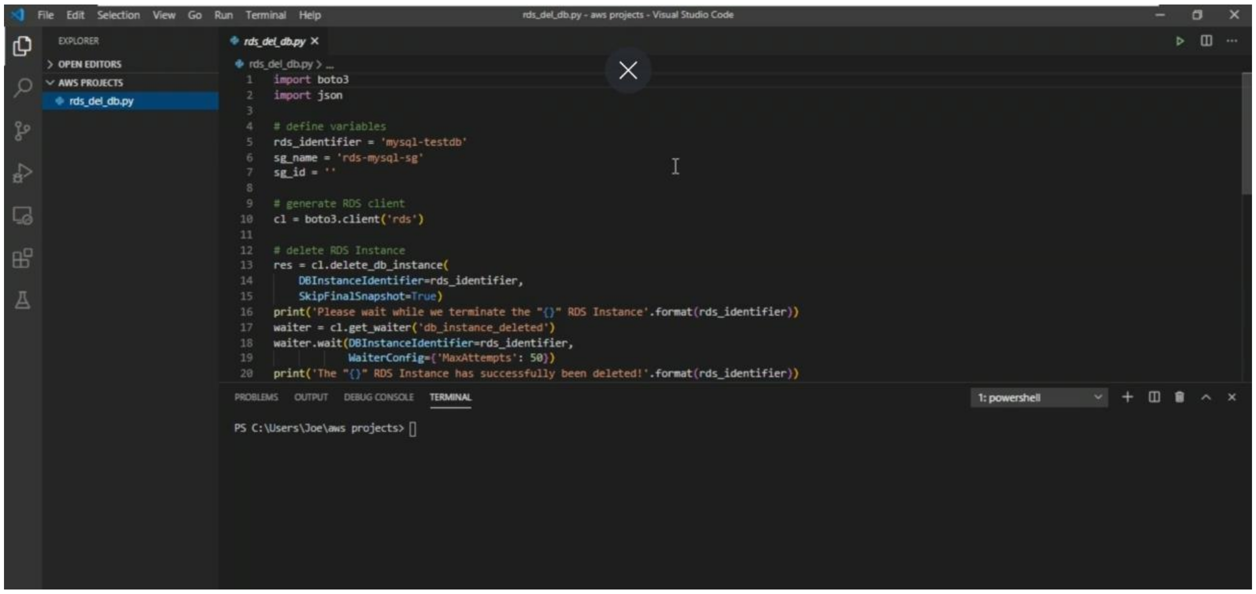


```python
14      user=user_name,
15      password=user_password,
16      database=db_name)
17  cursor = db.cursor()
18
19  # query the table within a try block
20  try:
21      sql = "SELECT * FROM `Students`"
22      cursor.execute(sql)
23      res = cursor.fetchall()
24      print('Querying table...')
25      print(res)
26  except mysqldb.Error as e:
27      print('Error: {}'.format(e))
28      print('Something went wrong!')
29  finally:
30      db.close()
31
```

```
PS C:\Users\Joe\aws projects> python .\rds_query_table.py
Querying table...
((1, 'Joe', 'Fredericton', 'Canada'), (2, 'Jan', 'New York', 'USA'), (3, 'Marcia', 'Boston', 'USA'), (4, 'Bob', 'Seattle', 'USA'))
PS C:\Users\Joe\aws projects>
```

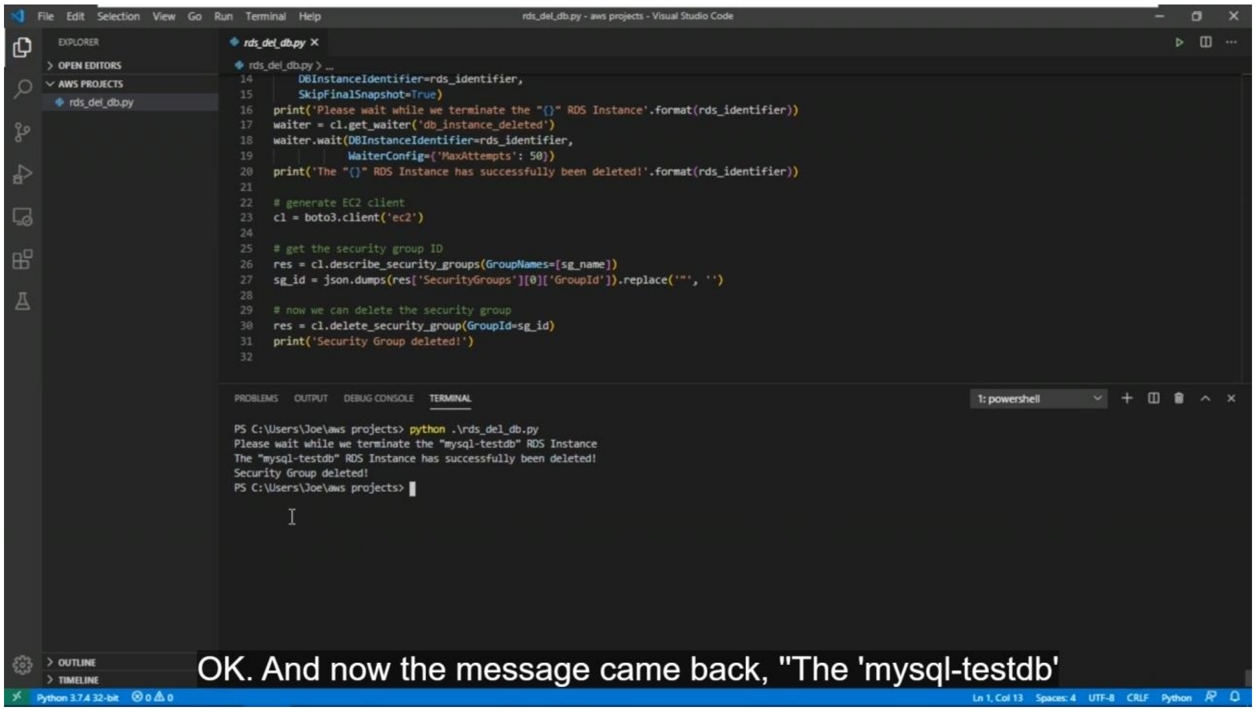
OK. And the message came up "Querying table..."

Deleting Databases and Security Groups



```python
import boto3
import json

# define variables
rds_identifier = 'mysql-testdb'
sg_name = 'rds-mysql-sg'
sg_id = ''

# generate RDS client
cl = boto3.client('rds')

# delete RDS Instance
res = cl.delete_db_instance(
    DBInstanceIdentifier=rds_identifier,
    SkipFinalSnapshot=True)
print('Please wait while we terminate the "{}" RDS Instance'.format(rds_identifier))
waiter = cl.get_waiter('db_instance_deleted')
waiter.wait(DBInstanceIdentifier=rds_identifier,
            WaiterConfig={'MaxAttempts': 50})
print('The "{}" RDS Instance has successfully been deleted!'.format(rds_identifier))
```

```
PS C:\Users\Joe\aws projects>
```

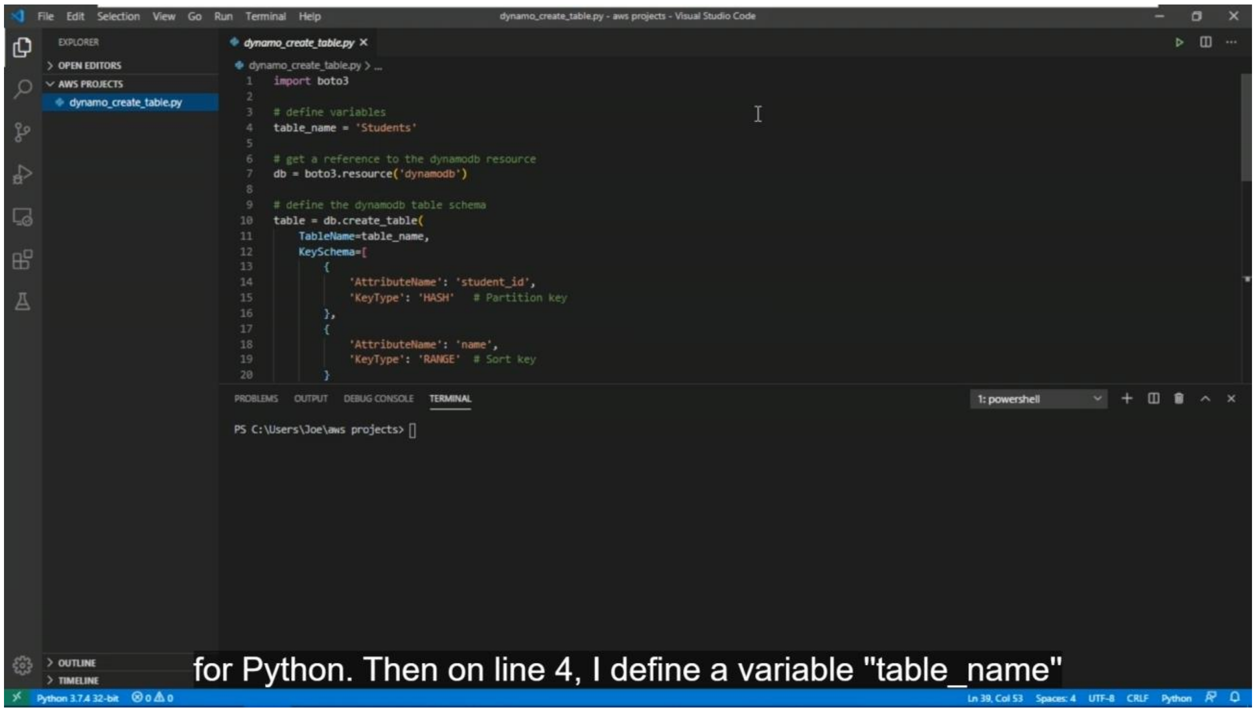


```python
    DBInstanceIdentifier=rds_identifier,
    SkipFinalSnapshot=True)
print('Please wait while we terminate the "{}" RDS Instance'.format(rds_identifier))
waiter = cl.get_waiter('db_instance_deleted')
waiter.wait(DBInstanceIdentifier=rds_identifier,
            WaiterConfig={'MaxAttempts': 50})
print('The "{}" RDS Instance has successfully been deleted!'.format(rds_identifier))

# generate EC2 client
cl = boto3.client('ec2')

# get the security group ID
res = cl.describe_security_groups(GroupNames=[sg_name])
sg_id = json.dumps(res['SecurityGroups'][0]['GroupId']).replace('"', '')

# now we can delete the security group
res = cl.delete_security_group(GroupId=sg_id)
print('Security Group deleted!')
```

```
PS C:\Users\Joe\aws projects> python .\rds_del_db.py
Please wait while we terminate the "mysql-testdb" RDS Instance
The "mysql-testdb" RDS Instance has successfully been deleted!
Security Group deleted!
PS C:\Users\Joe\aws projects>
```

OK. And now the message came back, "The 'mysql-testdb'

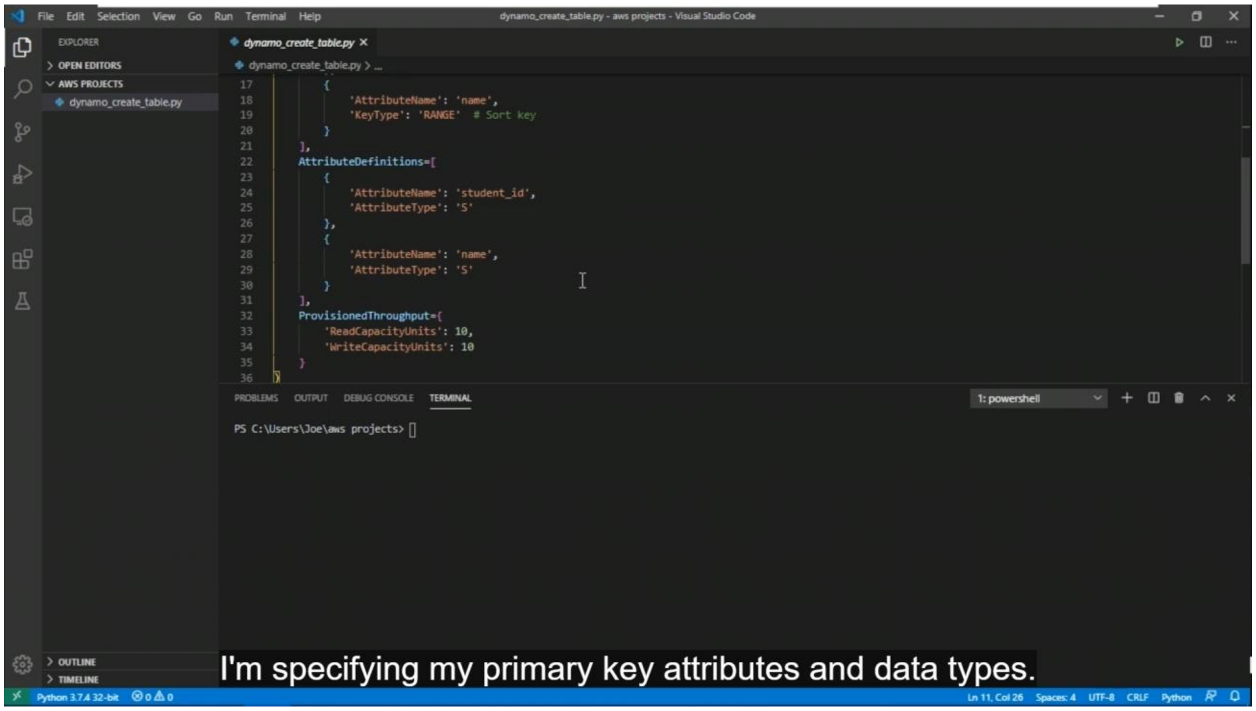
Creating a Table in Amazon DynamoDB



```python
1   import boto3
2
3   # define variables
4   table_name = 'Students'
5
6   # get a reference to the dynamodb resource
7   db = boto3.resource('dynamodb')
8
9   # define the dynamodb table schema
10  table = db.create_table(
11      TableName=table_name,
12      KeySchema=[
13          {
14              'AttributeName': 'student_id',
15              'KeyType': 'HASH'   # Partition key
16          },
17          {
18              'AttributeName': 'name',
19              'KeyType': 'RANGE'  # Sort key
20          }
```

```
PS C:\Users\Joe\aws projects> 
```

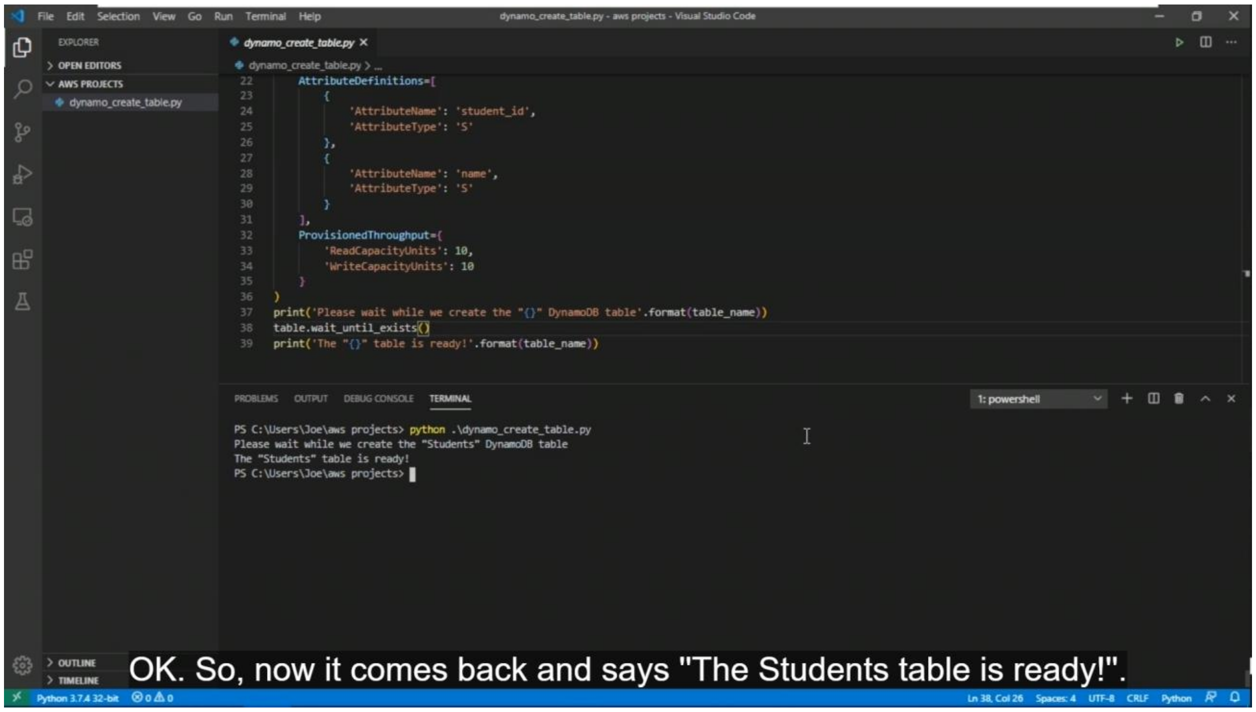
for Python. Then on line 4, I define a variable "table_name"



```
        {
            'AttributeName': 'name',
            'KeyType': 'RANGE'  # Sort key
        }
    ],
    AttributeDefinitions=[
        {
            'AttributeName': 'student_id',
            'AttributeType': 'S'
        },
        {
            'AttributeName': 'name',
            'AttributeType': 'S'
        }
    ],
    ProvisionedThroughput={
        'ReadCapacityUnits': 10,
        'WriteCapacityUnits': 10
    }
)
```

```
PS C:\Users\Joe\aws projects> 
```

I'm specifying my primary key attributes and data types.

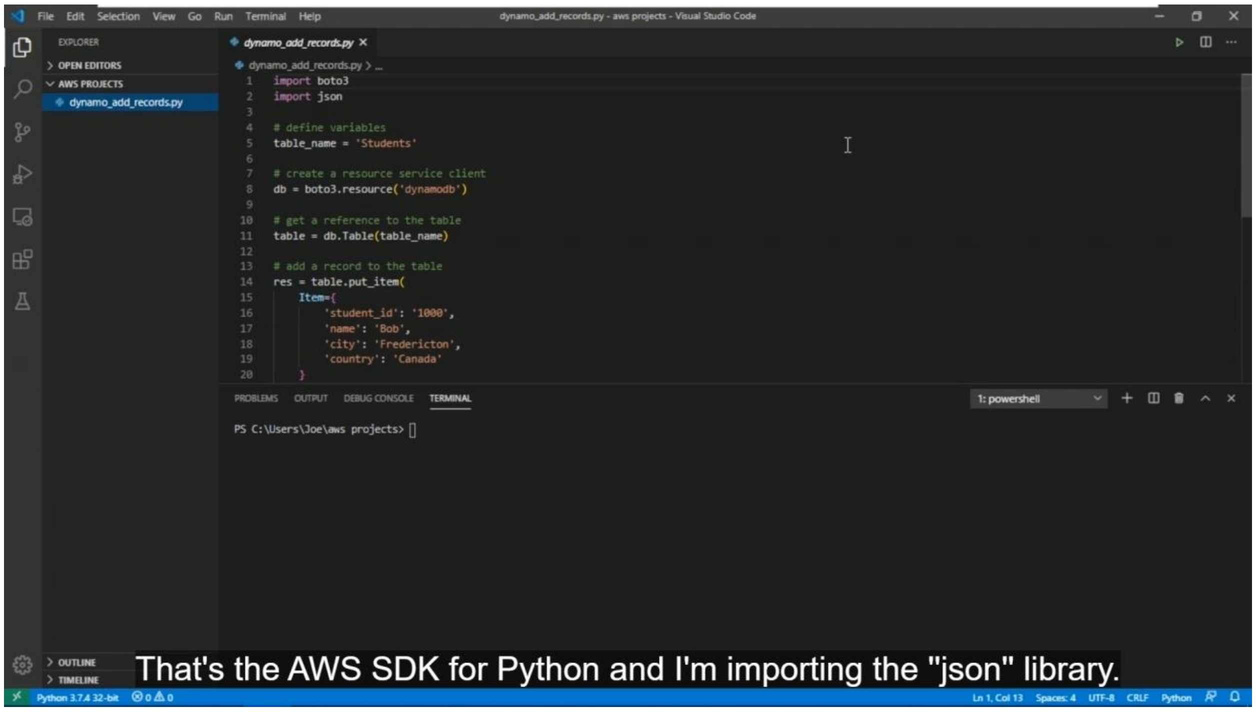


```
    AttributeDefinitions=[
        {
            'AttributeName': 'student_id',
            'AttributeType': 'S'
        },
        {
            'AttributeName': 'name',
            'AttributeType': 'S'
        }
    ],
    ProvisionedThroughput={
        'ReadCapacityUnits': 10,
        'WriteCapacityUnits': 10
    }
)
print('Please wait while we create the "{}" DynamoDB table'.format(table_name))
table.wait_until_exists()
print('The "{}" table is ready!'.format(table_name))
```

```
PS C:\Users\Joe\aws projects> python .\dynamo_create_table.py
Please wait while we create the "Students" DynamoDB table
The "Students" table is ready!
PS C:\Users\Joe\aws projects> 
```

OK. So, now it comes back and says "The Students table is ready!".

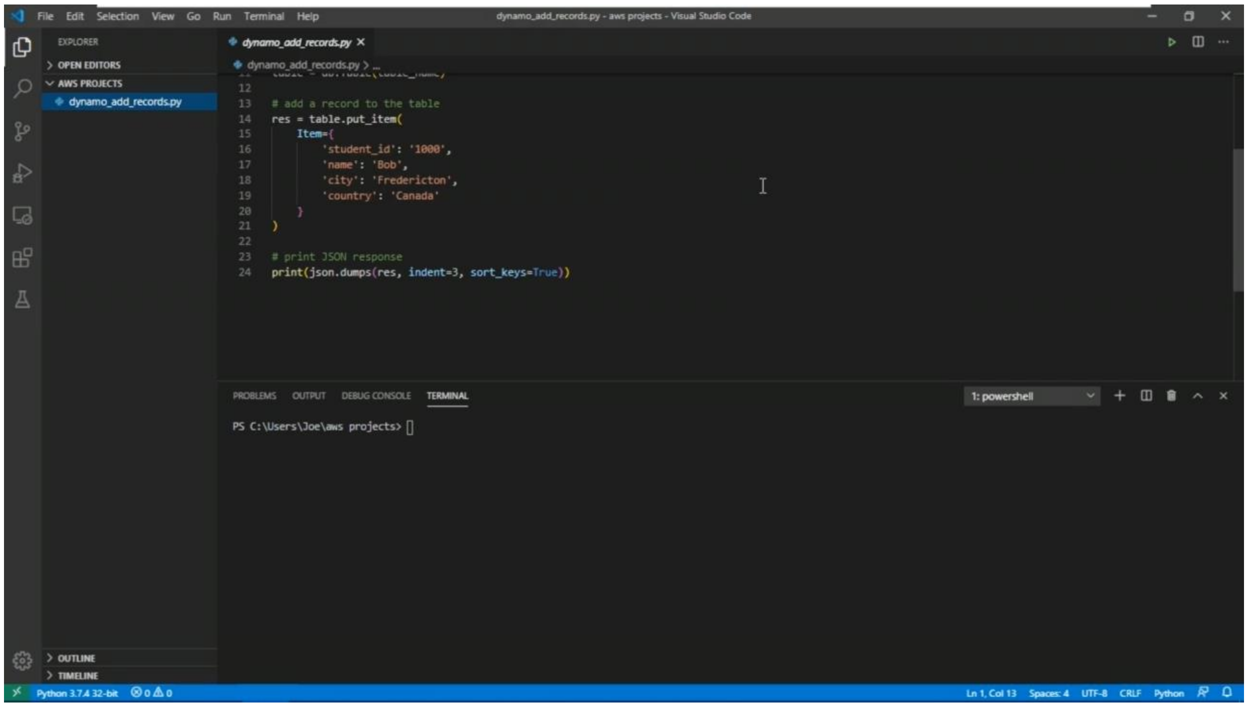
Adding Data to a DynamoDB Table



```python
import boto3
import json

# define variables
table_name = 'Students'

# create a resource service client
db = boto3.resource('dynamodb')

# get a reference to the table
table = db.Table(table_name)

# add a record to the table
res = table.put_item(
    Item={
        'student_id': '1000',
        'name': 'Bob',
        'city': 'Fredericton',
        'country': 'Canada'
    }
```

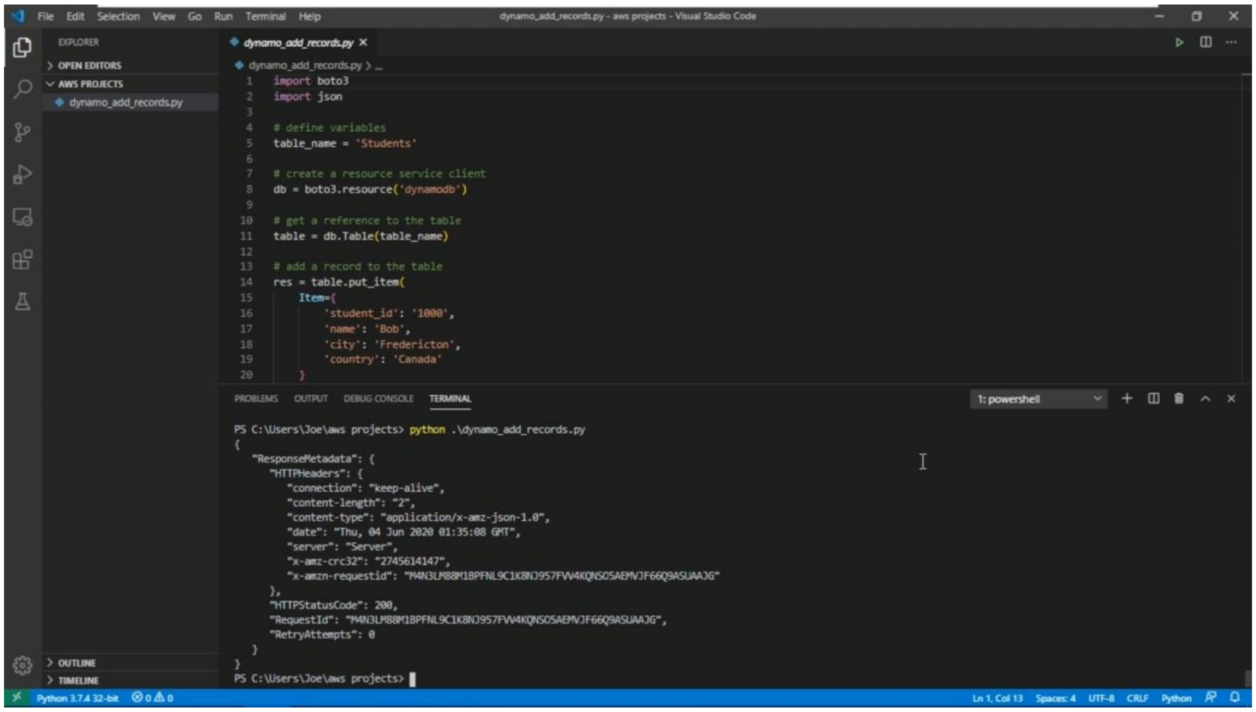
```
PS C:\Users\Joe\aws projects> 
```

That's the AWS SDK for Python and I'm importing the "json" library.



```python
# add a record to the table
res = table.put_item(
    Item={
        'student_id': '1000',
        'name': 'Bob',
        'city': 'Fredericton',
        'country': 'Canada'
    }
)

# print JSON response
print(json.dumps(res, indent=3, sort_keys=True))
```

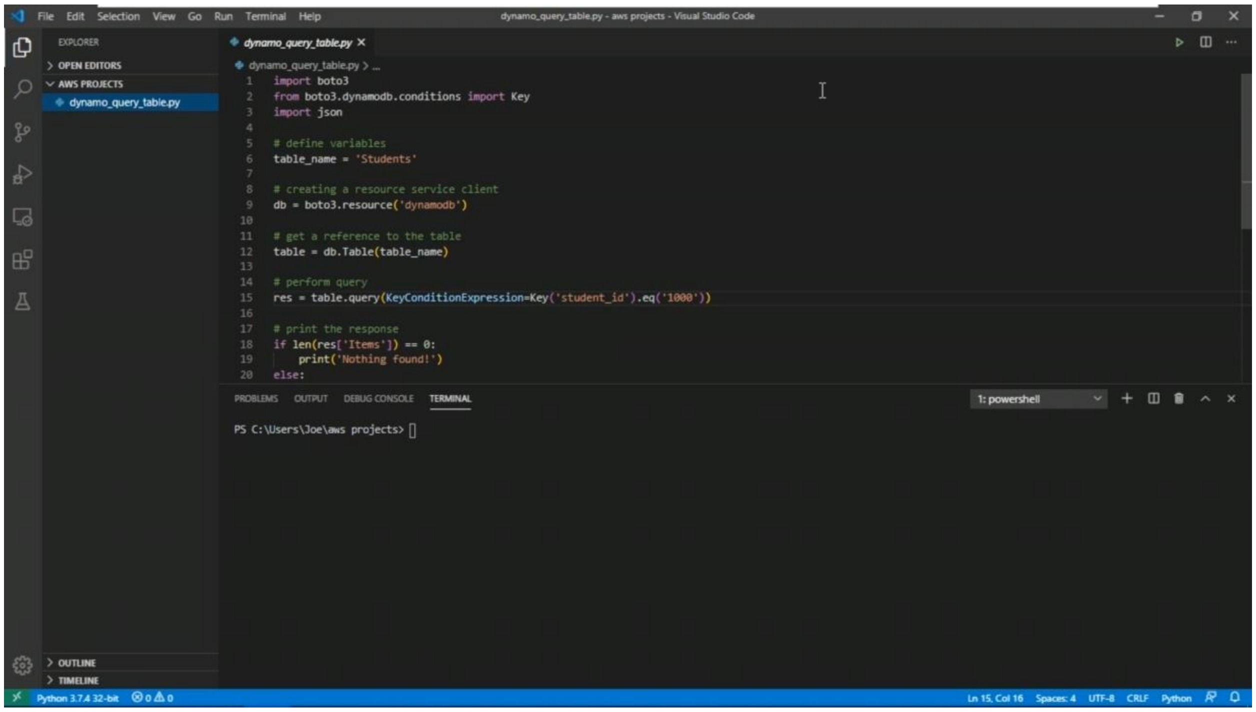
```
PS C:\Users\Joe\aws projects> 
```



```python
import boto3
import json

# define variables
table_name = 'Students'

# create a resource service client
db = boto3.resource('dynamodb')

# get a reference to the table
table = db.Table(table_name)

# add a record to the table
res = table.put_item(
    Item={
        'student_id': '1000',
        'name': 'Bob',
        'city': 'Fredericton',
        'country': 'Canada'
    }
```

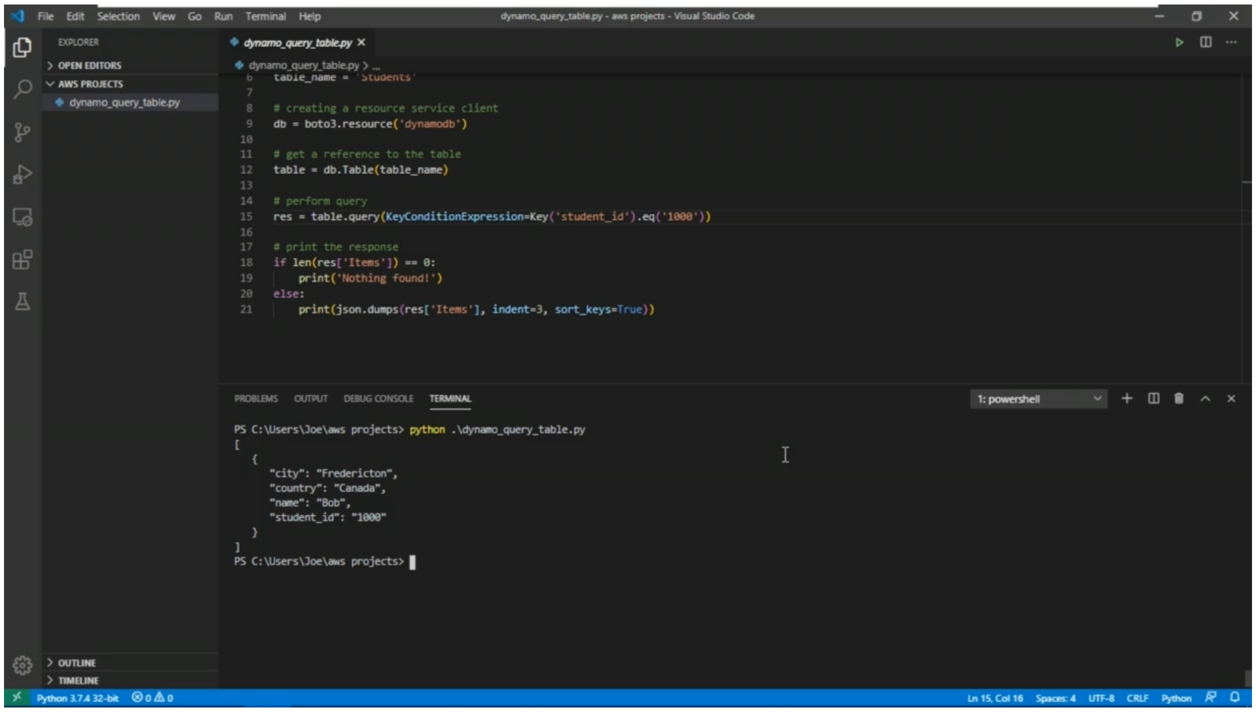
```
PS C:\Users\Joe\aws projects> python .\dynamo_add_records.py
{
   "ResponseMetadata": {
      "HTTPHeaders": {
         "connection": "keep-alive",
         "content-length": "2",
         "content-type": "application/x-amz-json-1.0",
         "date": "Thu, 04 Jun 2020 01:35:08 GMT",
         "server": "Server",
         "x-amz-crc32": "2745614147",
         "x-amzn-requestid": "M4N3LM88M1BPFNL9C1K8NJ957FVV4KQNSO5AEMVJF66Q9ASUAAJG"
      },
      "HTTPStatusCode": 200,
      "RequestId": "M4N3LM88M1BPFNL9C1K8NJ957FVV4KQNSO5AEMVJF66Q9ASUAAJG",
      "RetryAttempts": 0
   }
}
PS C:\Users\Joe\aws projects> 
```

Querying a DynamoDB Table



```python
import boto3
from boto3.dynamodb.conditions import Key
import json

# define variables
table_name = 'Students'

# creating a resource service client
db = boto3.resource('dynamodb')

# get a reference to the table
table = db.Table(table_name)

# perform query
res = table.query(KeyConditionExpression=Key('student_id').eq('1000'))

# print the response
if len(res['Items']) == 0:
    print('Nothing found!')
else:
```

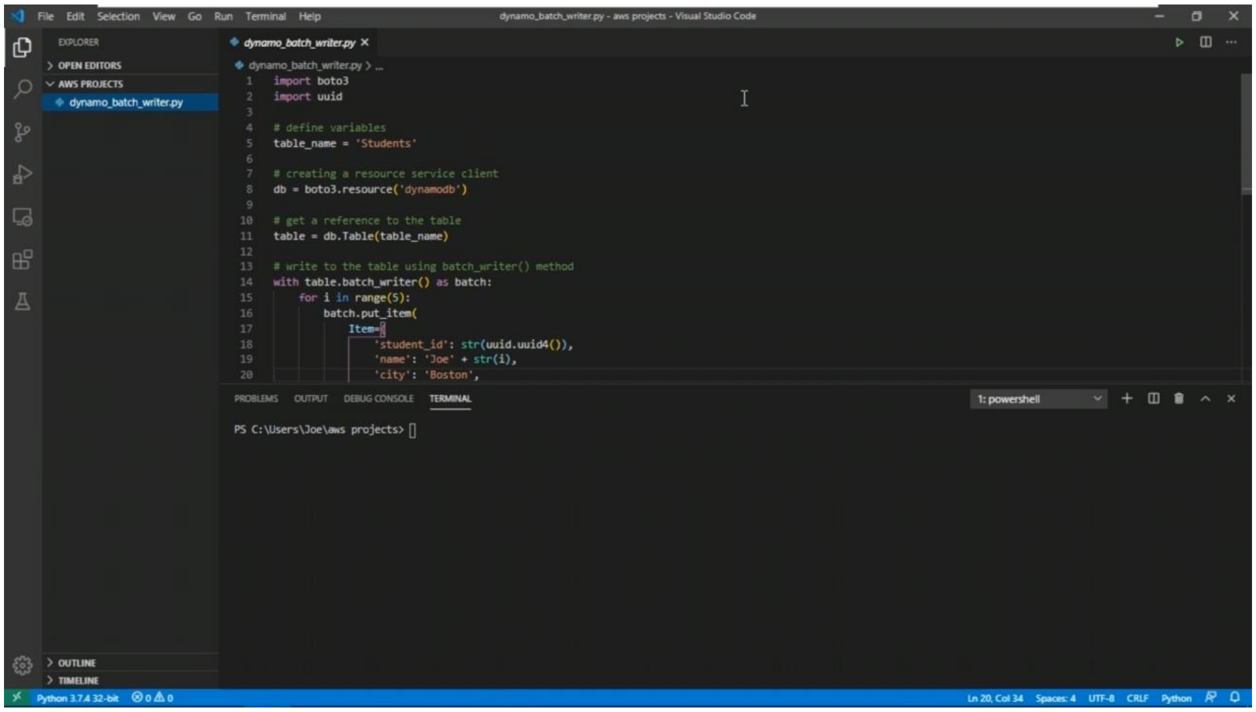
```
PS C:\Users\Joe\aws projects>
```



```python
table_name = 'Students'

# creating a resource service client
db = boto3.resource('dynamodb')

# get a reference to the table
table = db.Table(table_name)

# perform query
res = table.query(KeyConditionExpression=Key('student_id').eq('1000'))

# print the response
if len(res['Items']) == 0:
    print('Nothing found!')
else:
    print(json.dumps(res['Items'], indent=3, sort_keys=True))
```

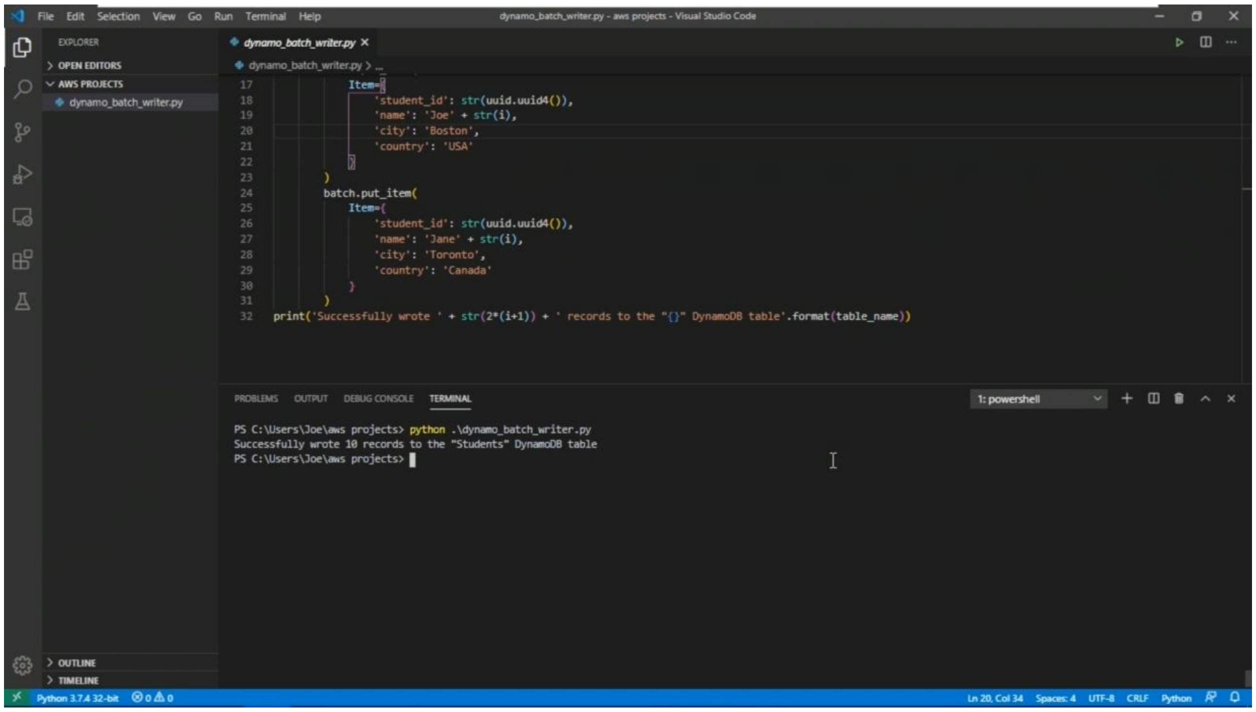
```
PS C:\Users\Joe\aws projects> python .\dynamo_query_table.py
[
   {
      "city": "Fredericton",
      "country": "Canada",
      "name": "Bob",
      "student_id": "1000"
   }
]
PS C:\Users\Joe\aws projects>
```

Using a Batch Writer against a DynamoDB Table


File Edit Selection View Go Run Terminal Help
dynamo_batch_writer.py - aws projects - Visual Studio Code
EXPLORER
OPEN EDITORS
AWS PROJECTS
dynamo_batch_writer.py
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
1: powershell
OUTLINE
TIMELINE
Python 3.7.4 32-bit
Ln 20, Col 34 Spaces: 4 UTF-8 CRLF Python

```python
import boto3
import uuid

# define variables
table_name = 'Students'

# creating a resource service client
db = boto3.resource('dynamodb')

# get a reference to the table
table = db.Table(table_name)

# write to the table using batch_writer() method
with table.batch_writer() as batch:
    for i in range(5):
        batch.put_item(
            Item={
                'student_id': str(uuid.uuid4()),
                'name': 'Joe' + str(i),
                'city': 'Boston',
```

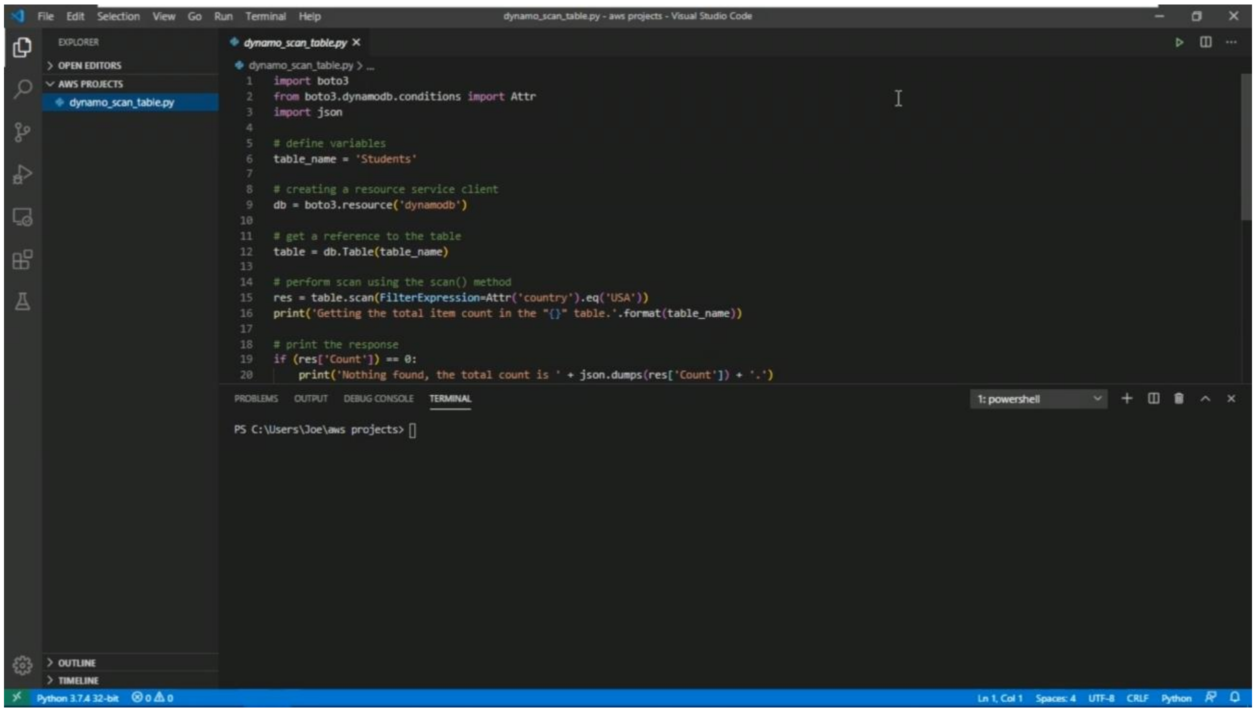
```
PS C:\Users\Joe\aws projects> 
```


File Edit Selection View Go Run Terminal Help
dynamo_batch_writer.py - aws projects - Visual Studio Code
EXPLORER
OPEN EDITORS
AWS PROJECTS
dynamo_batch_writer.py
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
1: powershell
OUTLINE
TIMELINE
Python 3.7.4 32-bit
Ln 20, Col 34 Spaces: 4 UTF-8 CRLF Python

```python
            Item={
                'student_id': str(uuid.uuid4()),
                'name': 'Joe' + str(i),
                'city': 'Boston',
                'country': 'USA'
            }
        )
        batch.put_item(
            Item={
                'student_id': str(uuid.uuid4()),
                'name': 'Jane' + str(i),
                'city': 'Toronto',
                'country': 'Canada'
            }
        )
print('Successfully wrote ' + str(2*(i+1)) + ' records to the "{}" DynamoDB table'.format(table_name))
```

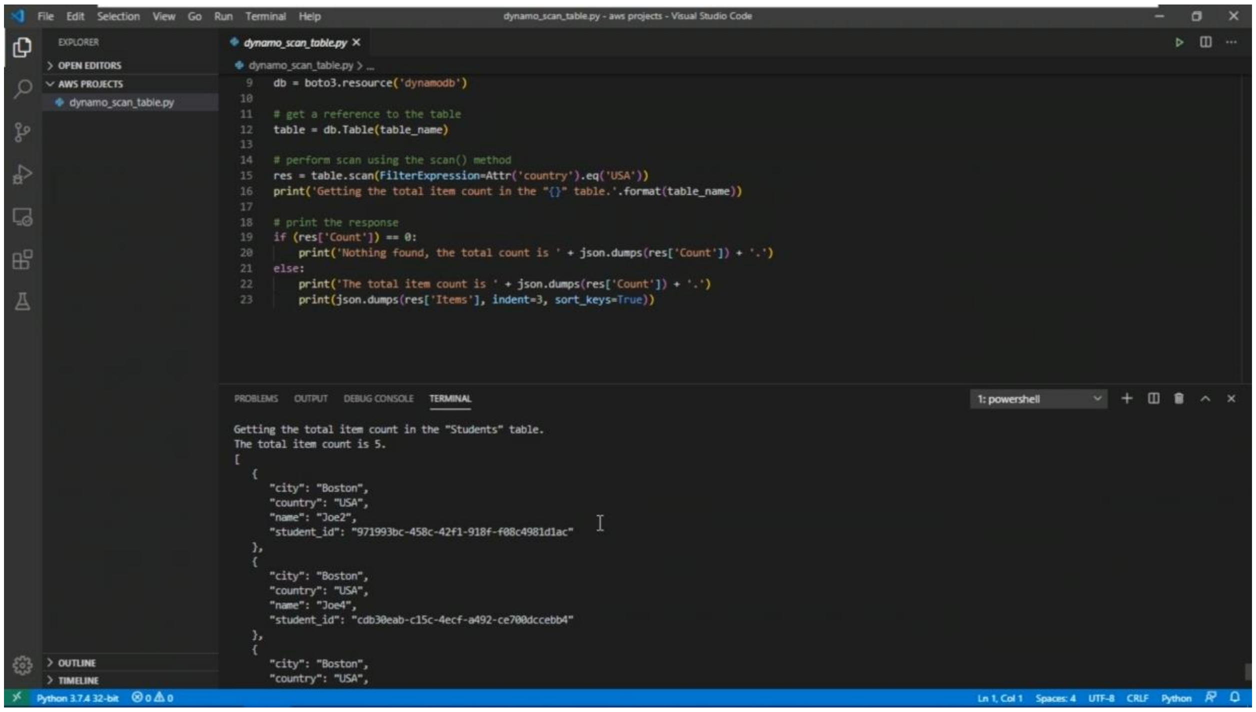
```
PS C:\Users\Joe\aws projects> python .\dynamo_batch_writer.py
Successfully wrote 10 records to the "Students" DynamoDB table
PS C:\Users\Joe\aws projects> 
```

Scanning a DynamoDB Table



```python
import boto3
from boto3.dynamodb.conditions import Attr
import json

# define variables
table_name = 'Students'

# creating a resource service client
db = boto3.resource('dynamodb')

# get a reference to the table
table = db.Table(table_name)

# perform scan using the scan() method
res = table.scan(FilterExpression=Attr('country').eq('USA'))
print('Getting the total item count in the "{}" table.'.format(table_name))

# print the response
if (res['Count']) == 0:
    print('Nothing found, the total count is ' + json.dumps(res['Count']) + '.')
```

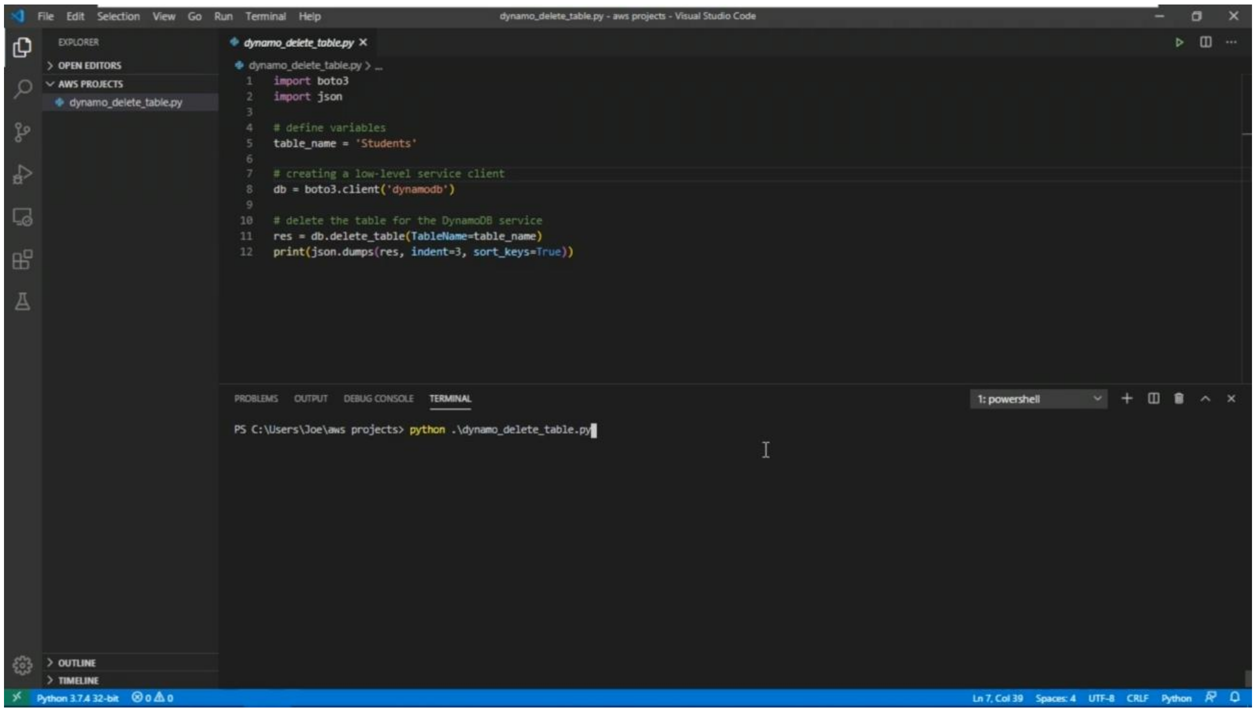
```
PS C:\Users\Joe\aws projects>
```



```python
db = boto3.resource('dynamodb')

# get a reference to the table
table = db.Table(table_name)

# perform scan using the scan() method
res = table.scan(FilterExpression=Attr('country').eq('USA'))
print('Getting the total item count in the "{}" table.'.format(table_name))

# print the response
if (res['Count']) == 0:
    print('Nothing found, the total count is ' + json.dumps(res['Count']) + '.')
else:
    print('The total item count is ' + json.dumps(res['Count']) + '.')
    print(json.dumps(res['Items'], indent=3, sort_keys=True))
```

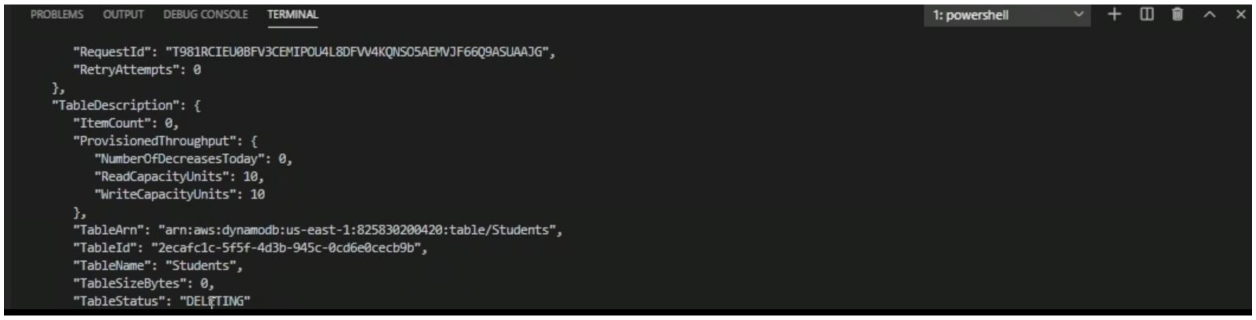
```
Getting the total item count in the "Students" table.
The total item count is 5.
[
   {
      "city": "Boston",
      "country": "USA",
      "name": "Joe2",
      "student_id": "971993bc-458c-42f1-918f-f08c4981d1ac"
   },
   {
      "city": "Boston",
      "country": "USA",
      "name": "Joe4",
      "student_id": "cdb30eab-c15c-4ecf-a492-ce700dccebb4"
   },
   {
      "city": "Boston",
      "country": "USA",
```

Deleting a DynamoDB Table



```python
import boto3
import json

# define variables
table_name = 'Students'

# creating a low-level service client
db = boto3.client('dynamodb')

# delete the table for the DynamoDB service
res = db.delete_table(TableName=table_name)
print(json.dumps(res, indent=3, sort_keys=True))
```

```
PS C:\Users\Joe\aws projects> python .\dynamo_delete_table.py
```



```
      "RequestId": "T981RCIEU0BFV3CEMIPOU4L8DFVV4KQNSO5AEMVJF66Q9ASUAAJG",
      "RetryAttempts": 0
   },
   "TableDescription": {
      "ItemCount": 0,
      "ProvisionedThroughput": {
         "NumberOfDecreasesToday": 0,
         "ReadCapacityUnits": 10,
         "WriteCapacityUnits": 10
      },
      "TableArn": "arn:aws:dynamodb:us-east-1:825830200420:table/Students",
      "TableId": "2ecafc1c-5f5f-4d3b-945c-0cd6e0cecb9b",
      "TableName": "Students",
      "TableSizeBytes": 0,
      "TableStatus": "DELETING"
```

1. Which relational database engines are available through Amazon RDS?
- **MySQL**
- **PostgreSQL**

2. When deleting an RDS instance, which waiter should be used in Python code to wait for the database instance to be identified as deleted by Amazon RDS?
- **db_instance_deleted**

3. What tool does AWS Data Migration Service (DMS) use for heterogenous migrations to ensure compatible schema in the target database?
- **Schema Conversion Tool (SCT)**

4. You've installed boto3 and PyMySQL using Python's pip package manager. What single command can be used to verify details about the versions of these packages installed?
- **pip show boto3 pymysql**

5. While using the create_table() method to create a table in Amazon DynamoDB, what information must be specified in the call to create_table() for a provisioned table?
- **Table name**
- **Primary key data types**
- **Throughput settings**
- **Primary key attributes**

6. Complete the code snippet to query a table in Amazon RDS.

```
# query the table within a try block
try:
  sql = "SELECT * FROM `Students` "
  cursor.execute(sql)
  res = cursor.<missing code>
  print('Querying table...')
  print(res)
except mysqldb.Error as e:
  print('Error: {}'.format(e))
```

```
    print('Something went wrong!')
  finally:
    db.close()
```
- **fetchall()**

7. Complete the code snippet to generate a cursor that will be used to create a table in an Amazon RDS MySQL database.

```
# connect to the target database
db = mysqldb.connect(
  host=rds_endpoint, user=user_name,
  password=user_password, database=db_name)
cursor = db.<missing code>()
# create the table within a try block
try:
  cursor.execute(
    "CREATE TABLE Students " +
    "(st_id INT NOT NULL AUTO_INCREMENT, name VARCHAR(20) NOT NULL," +
    "PRIMARY KEY (` student_id` ))")
  print('Table successfully created!')
except mysqldb.Error as e:
  print('Error: {}'.format(e))
finally:
  db.close()
```
- **cursor**

8. Which nonrelational databases are available from AWS?
- **Amazon DocumentDB**
- **Amazon DynamoDB**

9. Complete the code snippet to generate the inbound rules for a Security Group on AWS Cloud using the AWS SDK for Python.

```
# add rules for the security group
res = cl.<missing code>(
  CidrIp=ip_cidr,
  FromPort=3306,
  GroupName=sg_name,
  ToPort=3306,
  IpProtocol='tcp')
print("Security Group successfully created!")
```

- **authorize_security_group_ingress**

10. Which statement is used to import the Key class so that a query condition can be related to the key of an item.

- **from boto3.dynamodb.conditions import Key**

11. Which types of NoSQL databases are available from AWS?

- **Columnar**
- **Documents**

12. Which statement is used to import the appropriate class so that we can add conditions to a scanning operation against a DynamoDB table?

- **from boto3.dynamodb.conditions import Attr**

13. What are the two main strategies of caching?

- **Write through**
- **Lazy loading**

14. Complete the code snippet to create a resource service client that can be used on the DynamoDB service to add items to a table.

```
import boto3
import json
```

```
# define variables
table_name = 'Students'

# create a resource service client
db = boto3.resource('dynamodb')
```

- **resource('dynamodb')**

15. Complete the code snippet to delete a DynamoDB table using the delete_table() method.

```
import boto3
import json

# define variables
table_name = 'Students'

# creating a low-level service client
db = <missing code>

# delete the table for the DynamoDB service
res = db.delete_table(TableName=table_name)
print(json.dumps(res, indent=3, sort_keys=True))
```

- **boto3.client('dynamodb')**

16. Which characteristics accurately describe Amazon DynamoDB?

- **Features automatic scaling**
- **Fully managed database service**

17. What is the only DB instance size available if you want to use RDS free tier?

- **db.t2.micro**

18. Complete the code snippet to generate items in a DynamoDB table with the batch_writer() method.

```
# write to the table using batch_writer() method
```

```
with table.batch_writer() as batch:
  for i in range(5):
    batch.<missing code>(
      Item={
        'student_id': str(uuid.uuid4()),
        'name': 'Joe' + str(i),
        'city': 'Boston',
        'country': 'USA'
      }
    )
```

- **put_item**

19. You've installed MySQL Workbench 8.0 CE and you are configuring a new connection. In which field on the Setup New Connection dialog do you specify the Amazon RDS instance endpoint?

- **Hostname**

20. Which statements accurately describe relational database characteristics?

- **Relational databases incorporate integrity constraints**
- **Relational databases use the concept of transactions**