Setup DB for lambda

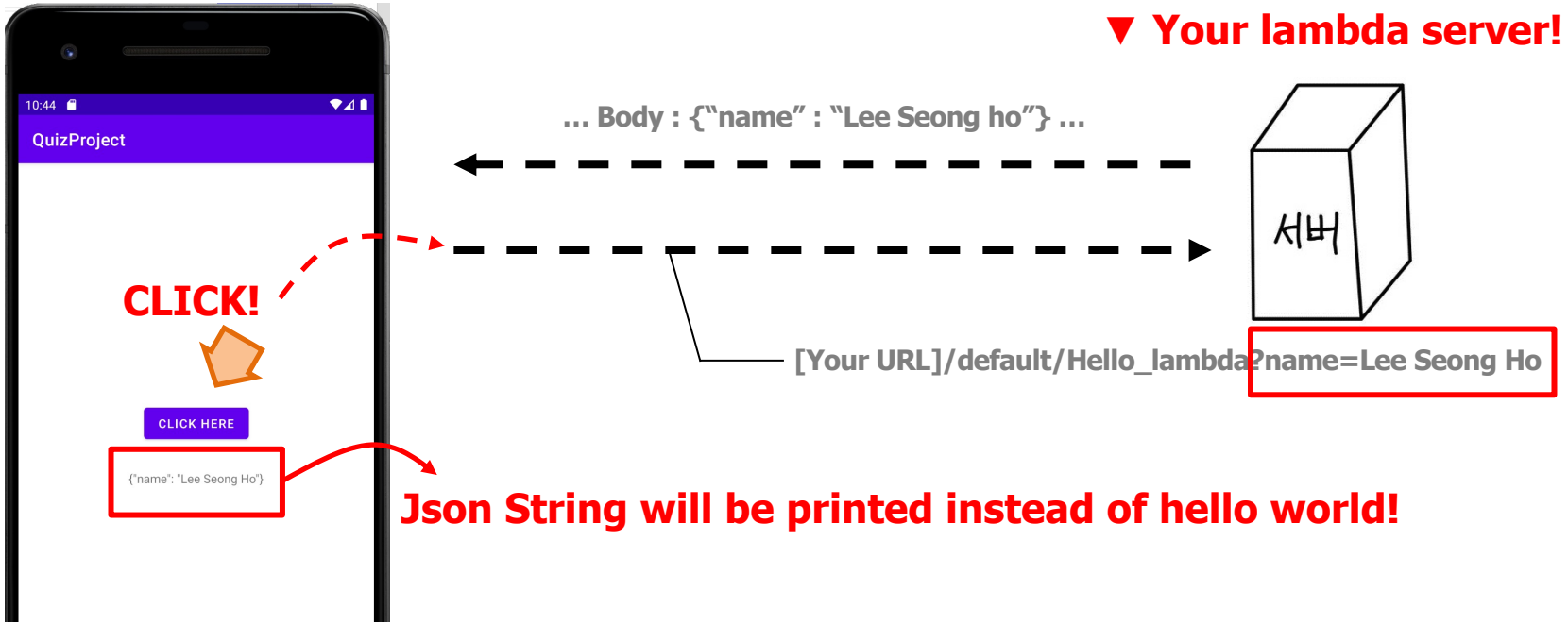
Mobile App Programming

What we learn today?

- **Let's connect AWS RDS PostgresQL to flask.**
  - Implement a web server with PostgresQL using SQLAlchemy.
  - Simply upload web server to AWS Lambda.
  - Make application contains login function.

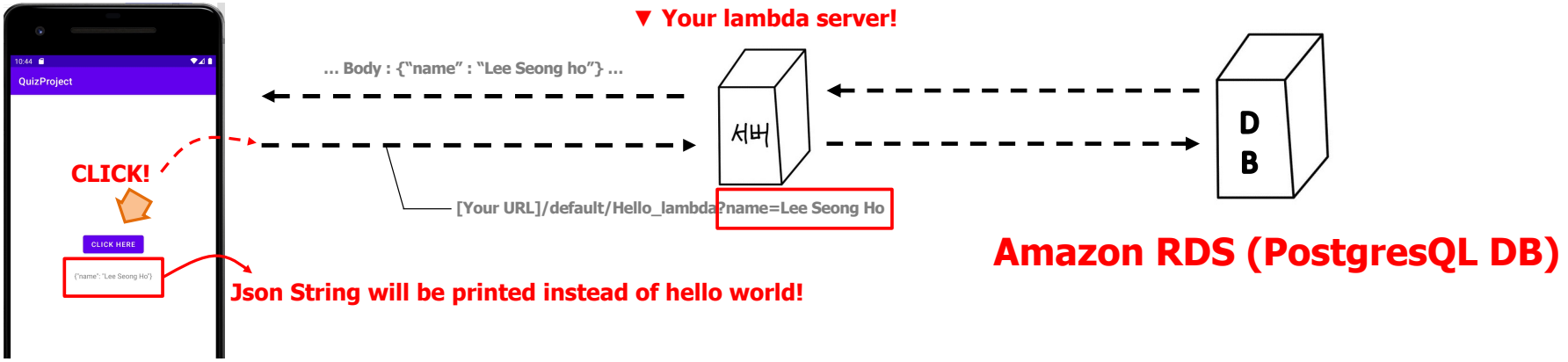
Remind

- **We upload Flask server to AWS Lambda using Zappa!**
  - We implemented a simple server using python Flask
  - We upload our flask server to AWS lambda using Zappa.



Today …

- **We connect our lambda flask with postgresQL RDS instance.**
  - We create RDS instance (PostgresQL).
  - We connect our flask server to RDS instance!



What we learn today?

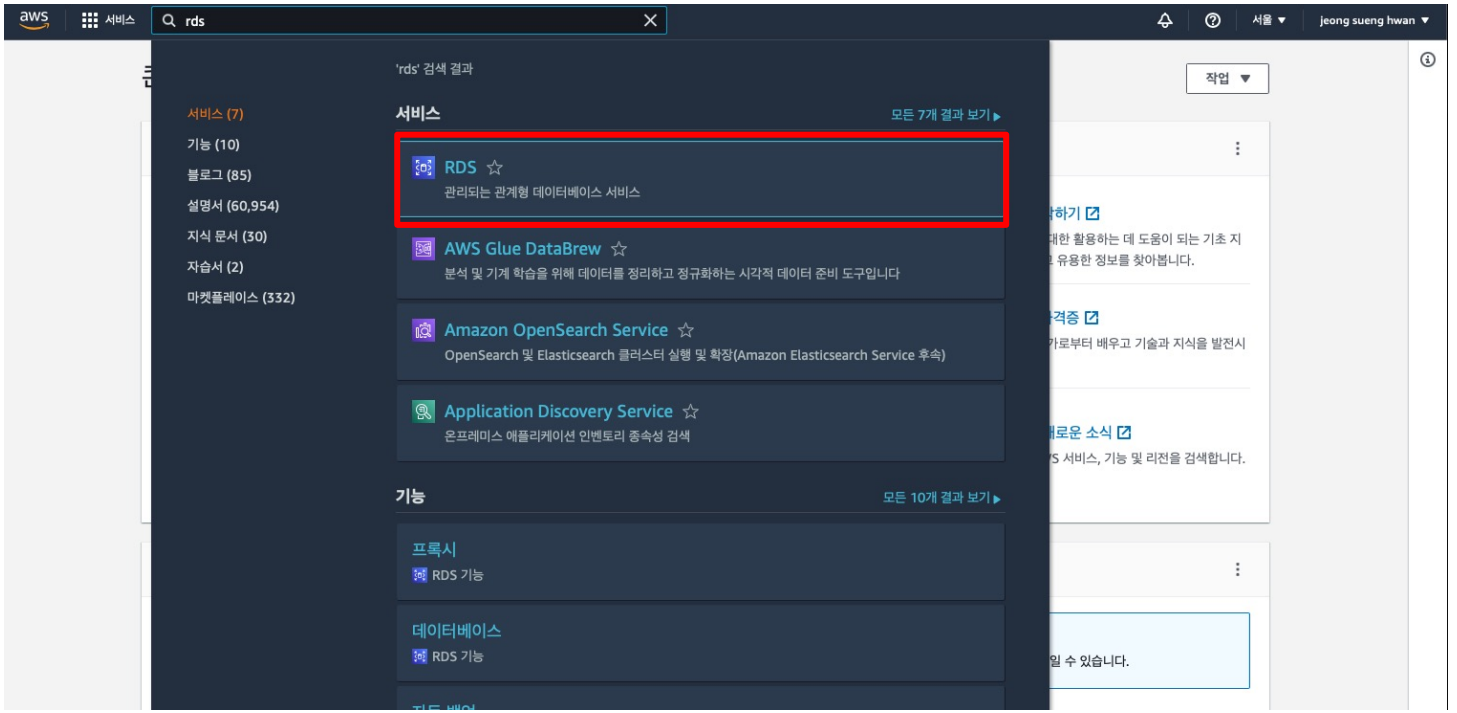
- **Process**
  1) Make a new RDS instance
  2) Setting VPC network
  3) Install SQLAlchemy and psycopg2-binary

  4) Create Engine, Session, Base
  5) Send add transaction and query transaction

  Exercise - Make an application with login function

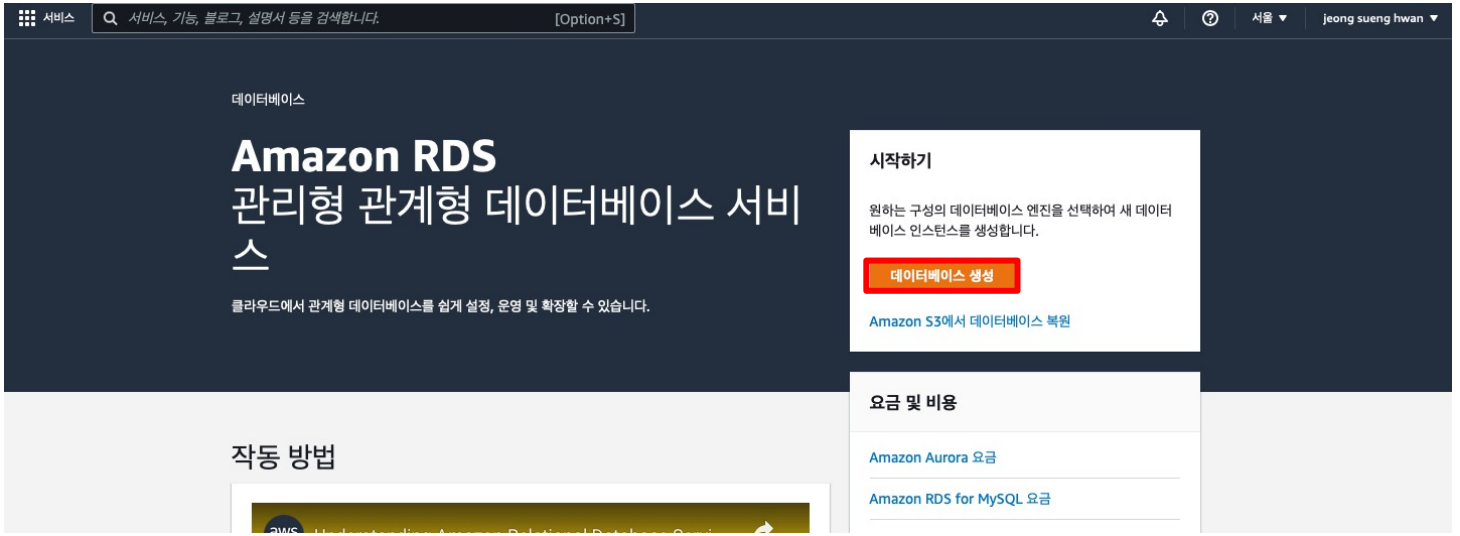
1) Make a new RDS instance

- **Go to AWS console, search RDS and select it**



1) Make a new RDS instance

- Click "Create new database"
- **Please be careful to setting your RDS.**



1) Make a new RDS instance

- **Setting a new RDS database**

RDS > 데이터베이스 생성

데이터베이스 생성

데이터베이스 생성 방식 선택 정보

- 표준 생성: 가용성, 보안, 백업 및 유지 관리에 대한 옵션을 포함하여 모든 구성 옵션을 설정합니다.
- 손쉬운 생성: 권장 모범 사례 구성을 사용합니다. 일부 구성 옵션은 데이터베이스를 생성한 후 변경할 수 있습니다.

엔진 옵션

엔진 유형 정보

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

버전

PostgreSQL 13.4-R1

템플릿

해당 사용 사례를 충족하는 샘플 템플릿을 선택하세요.

Must select free-tier!!!

- 프로덕션: 고가용성 및 빠르고 일관된 성능을 위해 기본값을 사용하세요.
- 개발/테스트: 이 인스턴스는 프로덕션 환경 외부에서 개발 용도로 마련되었습니다.
- 프리 티어: RDS 프리 티어를 사용하여 새로운 애플리케이션을 개발하거나, 기존 애플리케이션을 테스트하거나 Amazon RDS에서 실무 경험을 쌓을 수 있습니다. 정보

설정

DB 인스턴스 식별자 정보

DB 인스턴스 이름을 입력하세요. 이름은 현재 AWS 리전에서 AWS 계정이 소유하는 모든 DB 인스턴스에 대해 고유해야 합니다.

map-database

DB 인스턴스 식별자는 대소문자를 구분하지 않지만 'mydbinstance'와 같이 모두 소문자로 저장됩니다. 제약: 1자~60자의 영숫자 또는 하이픈으로 구성되어야 합니다. 첫 번째 문자는 글자이어야 합니다. 하이픈 2개가 연속될 수 없습니다. 끝에 하이픈이 올 수 없습니다.

▼ 자격 증명 설정

마스터 사용자 이름 정보

DB 인스턴스의 마스터 사용자에 로그인 ID를 입력하세요.

postgres

1~16자의 영숫자. 첫 번째 문자는 글자여야 합니다.

☐ 암호 자동 생성

Amazon RDS에서 사용자를 대신하여 암호를 생성하거나 사용자가 직접 암호를 지정할 수 있습니다.

마스터 암호 정보

••••••••

제약 조건: 8자 이상의 인쇄 가능한 ASCII 문자. 다음은 포함할 수 없습니다. /(슬래시), '(작은따옴표), "(큰따옴표) 및 @(앳 기호).

암호 확인 정보

••••••••

Please don't forget master username and password

인스턴스 구성

아래의 DB 인스턴스 구성 옵션은 위에서 선택한 엔진에서 지원하는 옵션으로 제한됩니다.

DB 인스턴스 클래스 정보

- 스탠다드 클래스(m 클래스 포함)
- 메모리 최적화 클래스(r 및 x 클래스 포함)
- 버스터블 클래스(t 클래스 포함)

db.t3.micro

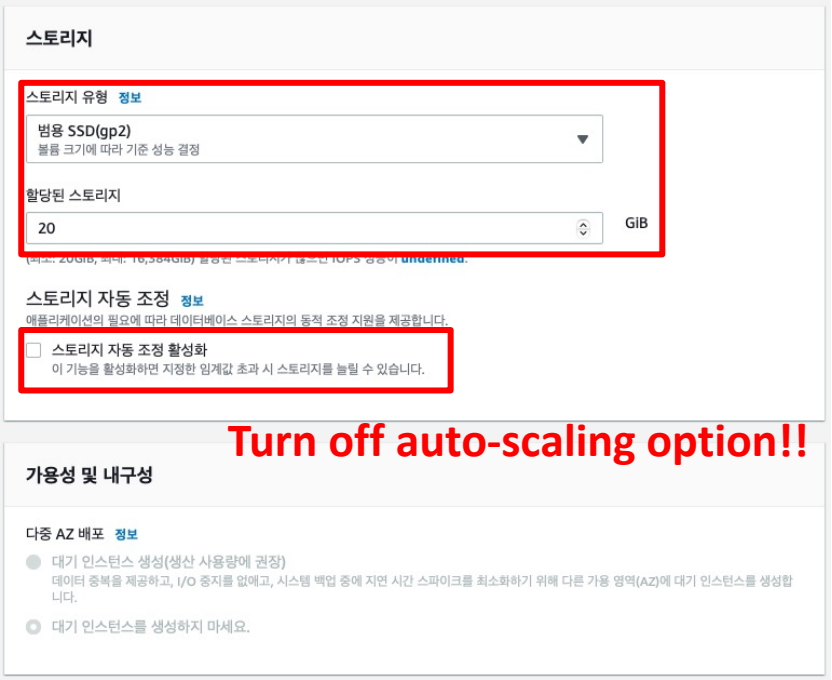
2 vCPUs 1 GiB RAM 네트워크: 2.085Mbps

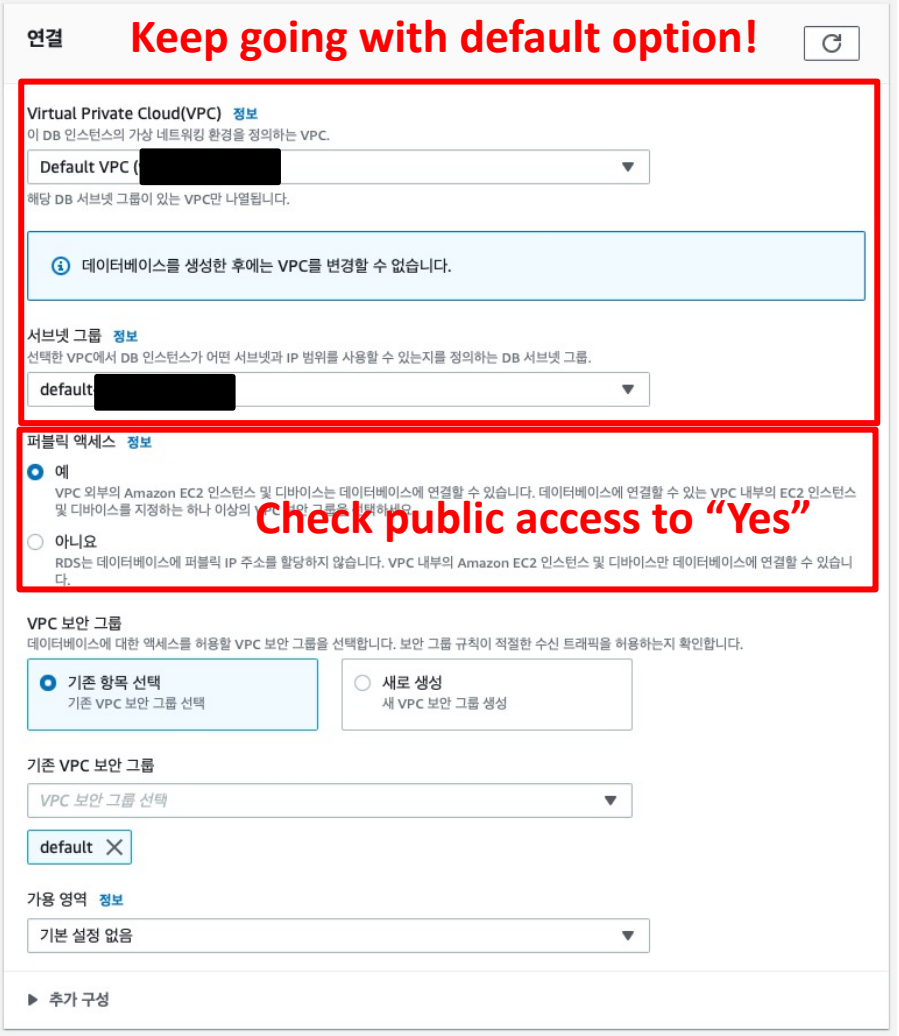
db.t3.micro

이전 세대 클래스 포함

1) Make a new RDS instance

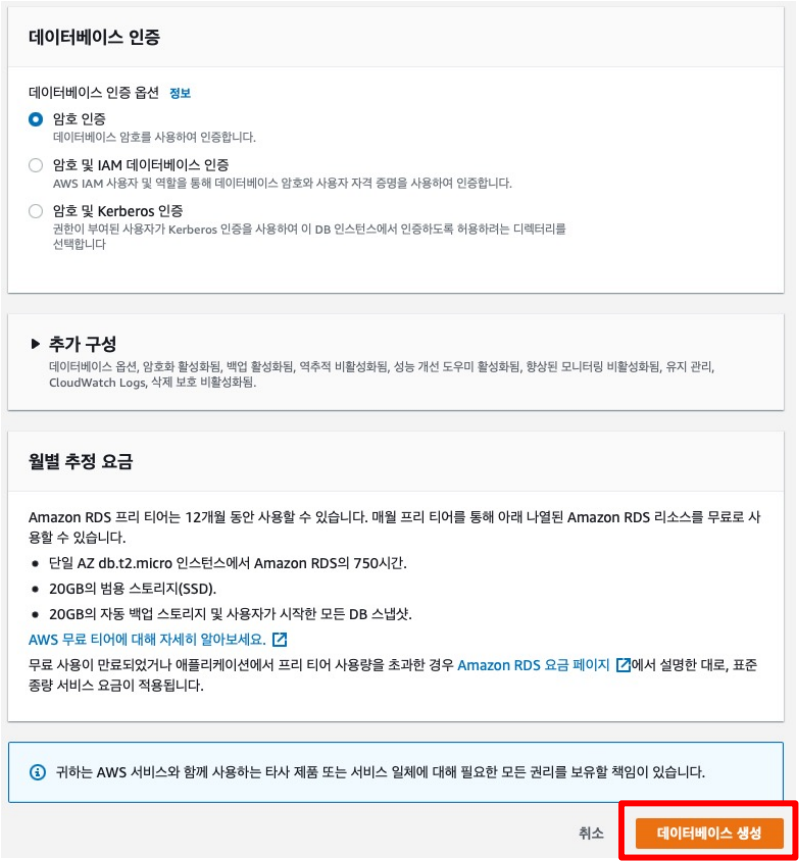
- **Setting a new RDS database**





1) Make a new RDS instance

- **Setting a new RDS database**



Create a PostgresQL Database!

1) Make a new RDS instance

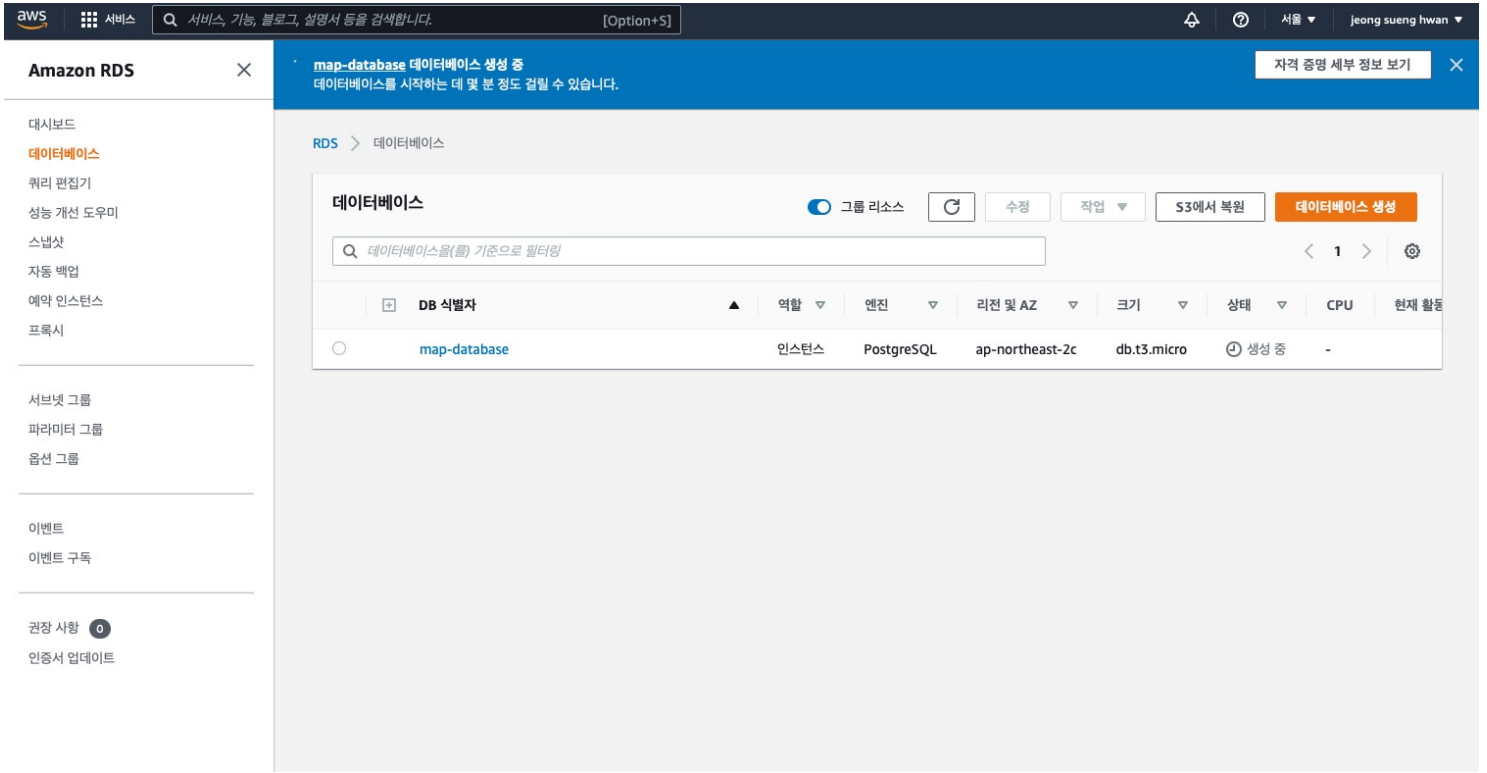
- **Be careful!!!!**
  - Free-tier RDS instance is only when **db.t3.micro** class with **20GB** storage.
  - In addition, if you use RDS instances more than **750 hours/month**, you must pay for overuse.
    - Ex) 1 RDS instance during 1 month

    => 24 hours * 31 = 744 hours/month -> free!
    - Ex) 2 RDS instance during 20 days

    => 24 hours * 20 * 2 = 960 hours/month -> pay for 210 hours/month!
  - Therefore, you must keep only **1 RDS free-tier instance**!!
  - When you do your project, remove other one.

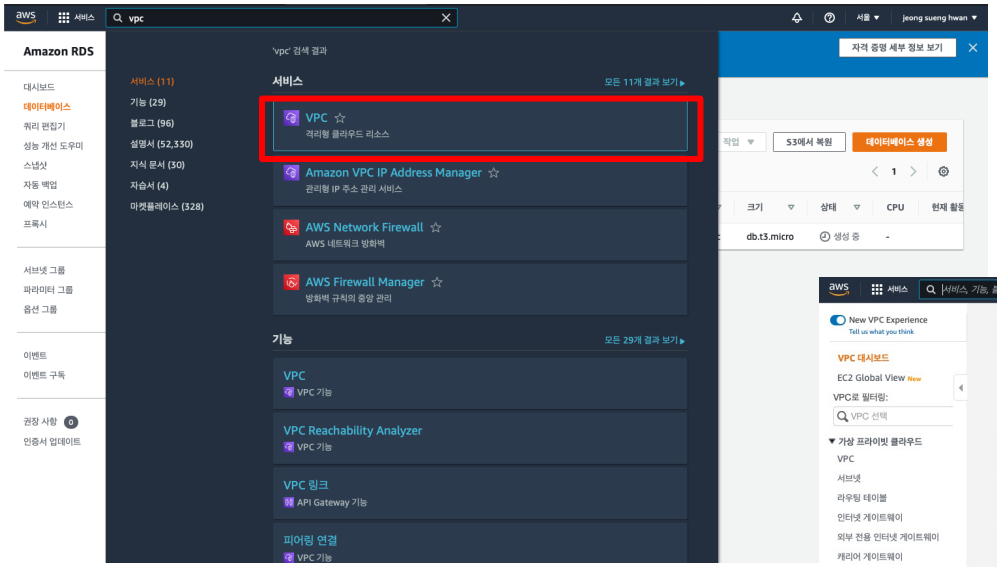
1) Make a new RDS instance

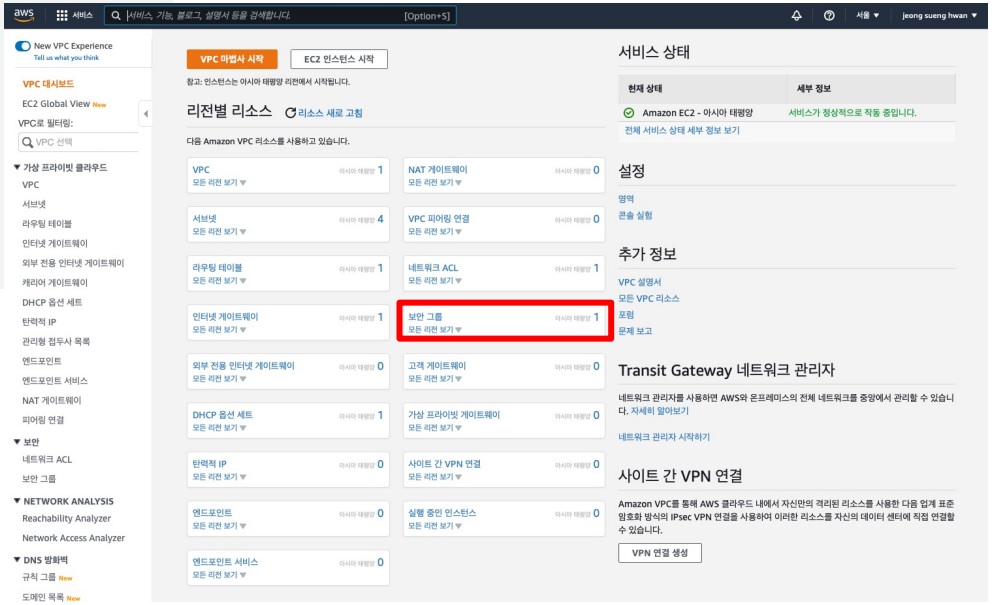
- **New database is created successfully!**



2) Setting VPC network

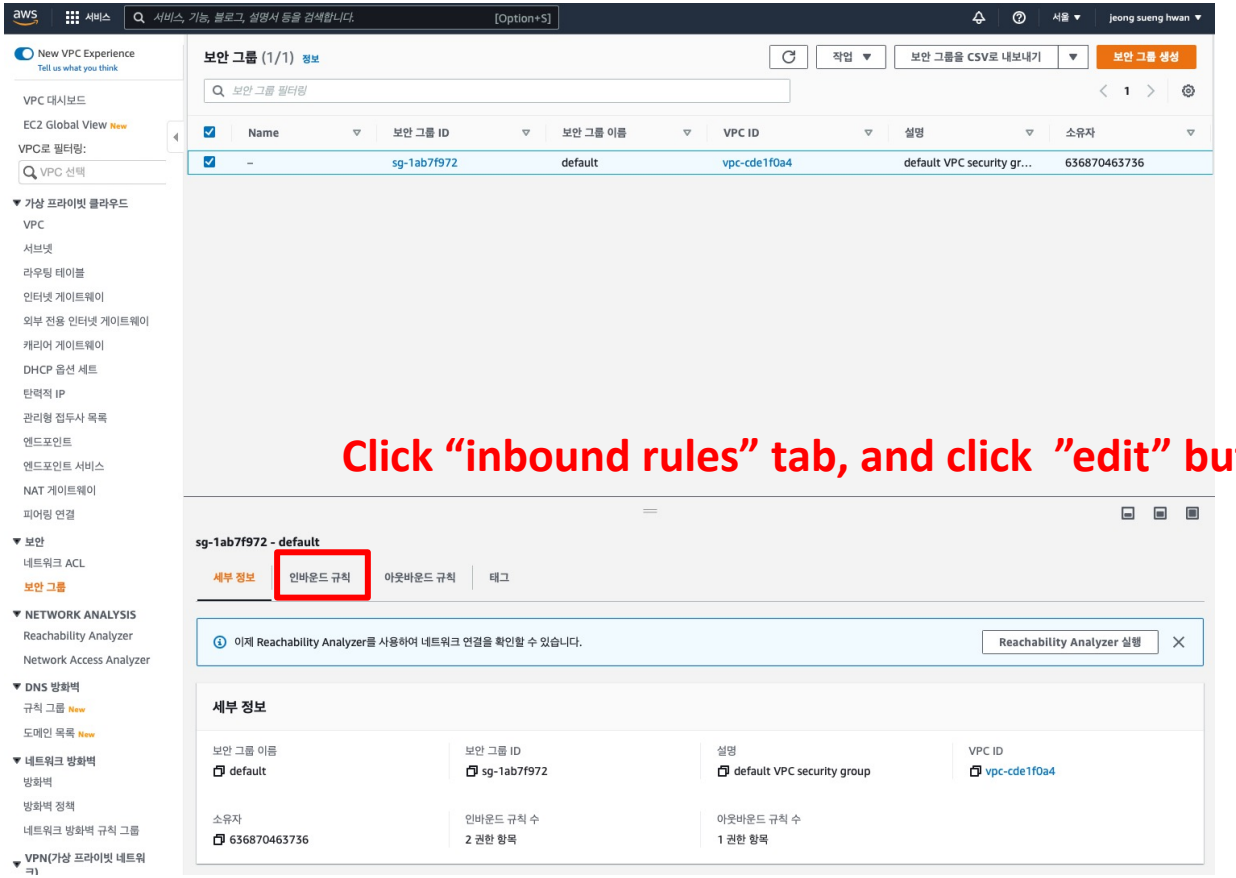
- **Search “VPC” and select it. Click “Security Group”.**





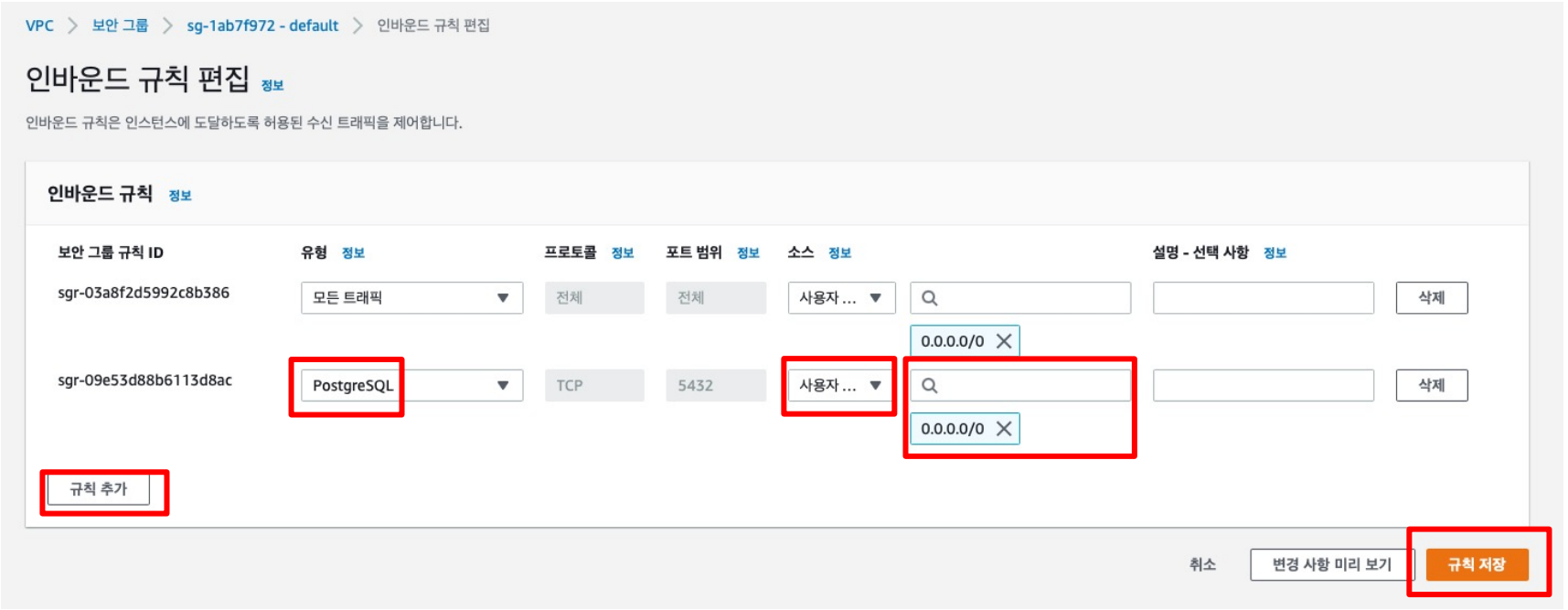
2) Setting VPC network

- **Search “VPC” and select it. Click “Security Group”.**



2) Setting VPC network

- **Add new rule**
  - **Type : PostgreSQL, Source : “사용자 지정” – 0.0.0.0/0**



Save it!

3) Install SQLAlchemy and psycopg2-binary

- **Activate Virtual environment and install belows.**

In Windows

```
$ .\[가상환경이름]\Scripts\activate.bat
$ pip install SQLAlchemy
$ pip install psycopg2-binary
```

In Mac

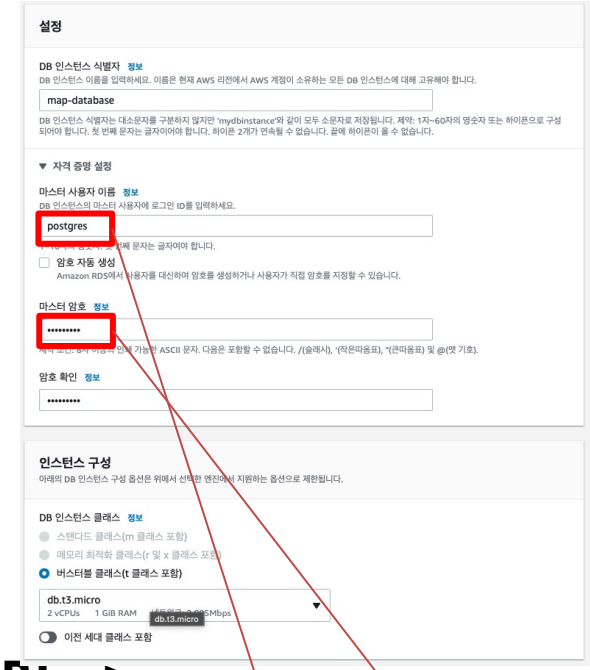
```
$ source [가상환경이름]/bin/activate
$ pip install SQLAlchemy
$ pip install psycopg2-binary
```

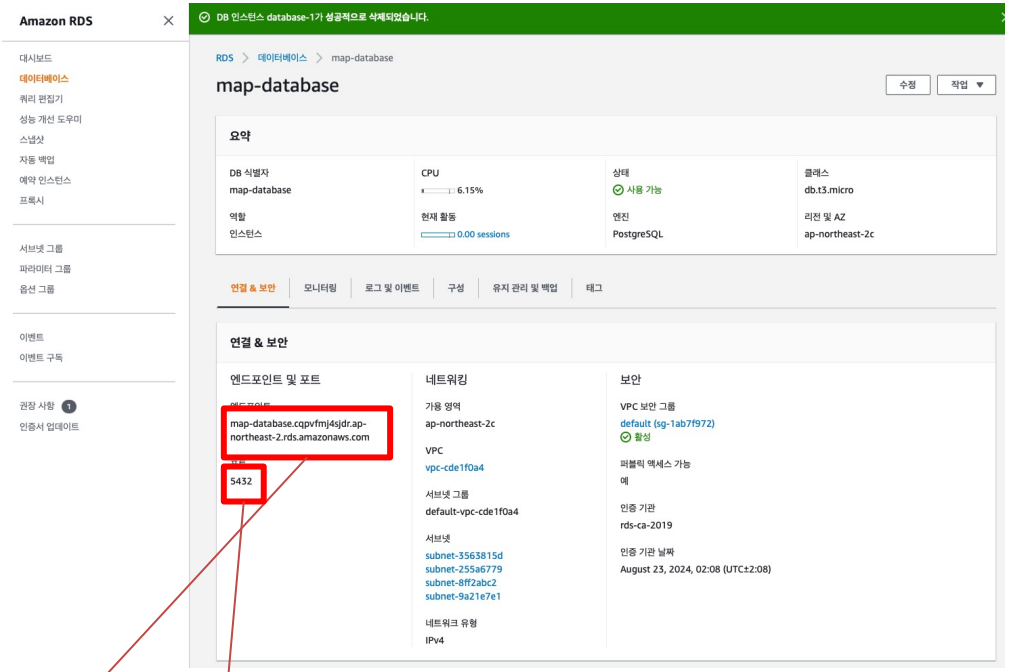
In Linux (include WSL)

```
$ source [가상환경이름]/bin/activate
$ pip install SQLAlchemy
$ pip install psycopg2-binary
```

3) Install SQLAlchemy and postgresQL

- **Make URI based on your RDS setting!**





URI =>

postgresql://{USER}:{PASSWORD}@{ENDPOINT URL}:{PORT}/{DB_NAME}

Default DB_NAME is “postgres"

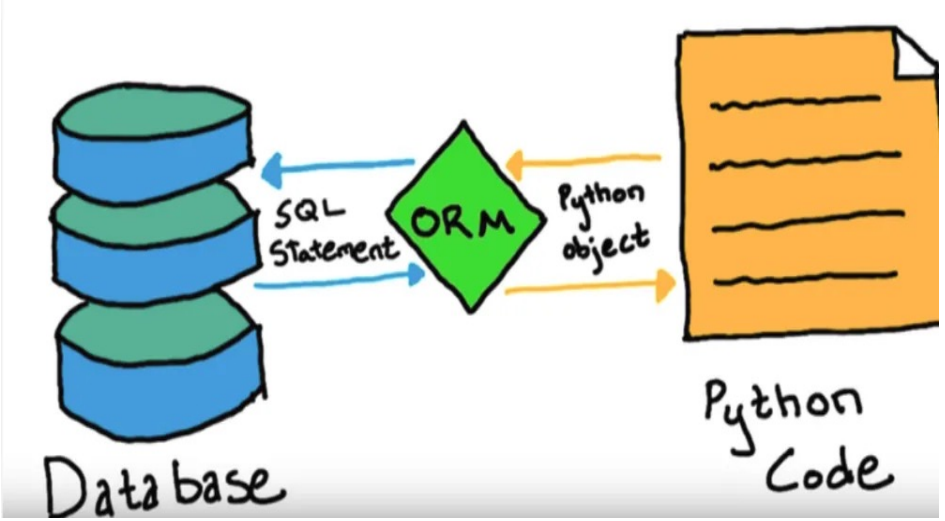
3) Install SQLAlchemy and postgresQL

- **Connect to your RDS instance using below code!**

```
from sqlalchemy import create_engine
from sqlalchemy.orm import scoped_session, sessionmaker
from sqlalchemy.ext.declarative import declarative_base

USER = "postgres"
PW = ”[redacted]'
URL = "map-database[redacted].ap-northeast-2.rds.amazonaws.com"
PORT = "5432"
DB = "postgres"

engine = create_engine("postgresql://{}:{}@{}:{}/{}".format(USER, PW, URL,PORT, DB))
db_session = scoped_session(sessionmaker(autocommit=False, autoflush=False, bind=engine))

Base = declarative_base()
Base.query = db_session.query_property()

Base.metadata.create_all(bind=engine)
```

4) Create Engine, Session, Base

- **SQLAlchemy is ORM(Object Relation Mapping) library.**
  - SQLAlchemy is the Python SQL toolkit and Object Relational Mapper that gives application developers the full power and flexibility of SQL.
  - It provides a full suite of well-known enterprise-level persistence patterns, designed for efficient and high-performing database access.



4) Create Engine, Session, Base

server.py (RDS part)

```
from sqlalchemy import create_engine
from sqlalchemy.orm import scoped_session, sessionmaker
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy import Column, Integer, String

USER = "postgres"
PW = "[redacted]
URL = "map-database[redacted]ap-northeast-2.rds.amazonaws.com"
PORT = "5432"
DB = "postgres"

engine = create_engine("postgresql://{}:{}@{}:{}/{}".format(USER, PW, URL,PORT, DB))
db_session = scoped_session(sessionmaker(autocommit=False, autoflush=False, bind=engine))

Base = declarative_base()
Base.query = db_session.query_property()

class User(Base):
    __tablename__ = 'users'
    id = Column(Integer, primary_key=True)
    name = Column(String(50), unique=True)
    passwd = Column(String(120), unique=False)

    def __init__(self, name=None, passwd=None):
        self.name = name
        self.passwd = passwd

    def __repr__(self):
        return f'<User {self.name!r}>'

# Base.metadata.drop_all(bind=engine)
Base.metadata.create_all(bind=engine)
```

Added!

1) Connect to our DB instance

2) Make session to communicate with RDS instance!

3) Declare Base class (A base class stores a catalog of classes and mapped tables in the system)

4) Declare Data Class!

5) Create all tables (drop_all( ) method delete all tables)

5) Send add transaction and query transaction

server.py (Flask part)

```
... Write below of RDS part code ...

from flask import Flask
from flask import request
from flask import jsonify
from werkzeug.serving import WSGIRequestHandler
import json

import json
WSGIRequestHandler.protocol_version = "HTTP/1.1"

app = Flask(__name__)

@app.route("/adduser", methods=['POST'])
def add_user():
    content = request.get_json(silent=True)
    name = content["name"]
    passwd = content["passwd"]
    if db_session.query(User).filter_by(name=name).first() is None:
        u = User(name=name, passwd=passwd)
        db_session.add(u)
        db_session.commit()
        return jsonify(success=True)
    else:
        return jsonify(success=False)

if __name__ == "__main__":
    app.run(host='localhost', port=8888)
```

Send query in ”User” table,
Filter with ”name=name” condition

Make new ROW(data) and send add transaction using {session}.add( ) method

You must call {session}.commit( ) method after send transaction

5) Send add transaction and query transaction

server.py (Flask part)

... Add new routing method below of before codes ...

```
@app.route("/login", methods=['POST'])
def login():
    content = request.get_json(silent=True)
    name = content["name"]
    passwd = content["passwd"]

    check = False
    result = db_session.query(User).all()
    for i in result:
        if i.name == name and i.passwd == passwd:
            check = True
    return jsonify(success=check)
```

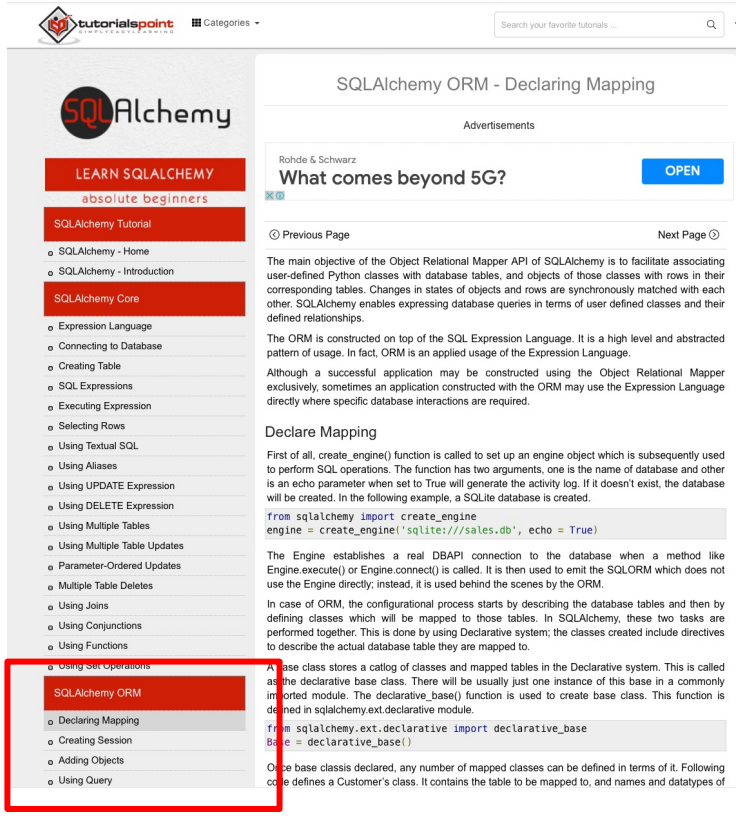
Send query in "User" table, to get all ROWs of table. all( ) method return list of all satisfied ROWs.

Check data attributes of each ROW.

5) Send add transaction and query transaction

If you want to know about updating ROW, deleting ROW, or JOIN operation, Please check below links "SQLAlchemy ORM" tab!

https://www.tutorialspoint.com/sqlalchemy/sqlalchemy_orm_declaring_mapping.htm



Exercise – Login Function Application

- Using “zappa deploy dev”, upload your flask server to AWS Lambda!
- Send Screenshot of your RDS instance page and two Screenshots of reqbin webpage with “adduser” and ”login” requests.
- Zip your python code with below screenshots and Submit it on ICAMPUS.