sql_orm.py

```
import sqlite3
import hashlib
import secrets
import string

class Users_db:
    def __init__(self):
        self.conn = None
        self.cursor = None

    def open_DB(self):
        self.conn = sqlite3.connect('users.db')
        self.cursor = self.conn.cursor()

    def close_DB(self):
        self.conn.close()

    def commit(self):
        self.conn.commit()

    def create_table(self):
        self.open_DB()
        self.cursor.execute('''DROP TABLE Users''')
        self.cursor.execute('''CREATE TABLE Users (username VARCHAR(20)
        UNIQUE NOT NULL, password VARCHAR(64), salt VARCHAR(5));''')
        self.commit()
        self.close_DB()

    def insert_new_user(self, username: str, password: str) -> bool:
        self.open_DB()
        self.cursor.execute("SELECT username FROM Users WHERE username
        = ?", (username,))
        res = self.cursor.fetchone()
        if res: # Exists
            self.close_DB()
            return False

        salt = ''.join(secrets.choice(string.ascii_letters) for _ in
        range(5))
        hash_pass =
        hashlib.sha256((password+salt).encode()).hexdigest()
        self.cursor.execute("INSERT INTO Users (username, password,
        salt) VALUES (?, ?, ?);",
                            (username, hash_pass, salt))
        self.commit()
        self.close_DB()
        return True

    def remove_user(self, username) -> bool:
        self.open_DB()
        self.cursor.execute("SELECT username FROM Users WHERE username
        = ?", (username,))
        res = self.cursor.fetchone()
        if res == None:
            print("User Doesn't Exist, can't remove")
            self.close_DB()
            return False

        self.cursor.execute("DELETE FROM Users WHERE username=?;",
                            (username,))
        self.commit()
        self.close_DB()
        return True

    def user_exists(self, username, password) -> bool:
        self.open_DB()
        self.cursor.execute("SELECT salt FROM Users WHERE username=?;",
        (username,))
        salt = self.cursor.fetchone()
        if salt == None:
            self.close_DB()
            return False

        salt = salt[0] # db returns tuple of return values, we need the
        str itself

        hash_pass = hashlib.sha256((password +
        salt).encode()).hexdigest()

        self.cursor.execute(
            "SELECT username FROM Users WHERE username=? AND
            password=?;",
            (username, hash_pass))

        res = self.cursor.fetchone()
        self.close_DB()

        if res:
            return True
        return False

    def username_exists(self, username) -> bool:
        self.open_DB()
        self.cursor.execute("SELECT username FROM Users WHERE
        username=?;", (username,))
        res = self.cursor.fetchone()
        self.close_DB()
        if res:
            return True
        return False

    def upgrade_to_pro(self, username):
        self.open_DB()
        self.cursor.execute("UPDATE Users SET is_pro = ? WHERE username
        = ?;", (True, username))
        self.commit()
        self.close_DB()

    def get_email_from_username(self, username):
        self.open_DB()
        self.cursor.execute("SELECT email FROM Users WHERE username=?",
        (username,))
        res = self.cursor.fetchone()
        self.close_DB()
        if not res:
            return ''
        return res[0]

    def set_new_password(self, username, password):
        self.open_DB()
        hash_pass = hashlib.sha256(password.encode()).hexdigest()
        self.cursor.execute("UPDATE Users SET password = ? WHERE
        username = ?;", (hash_pass, username))
        self.commit()
        self.close_DB()
```