# Python Database Connectivity

Week 3

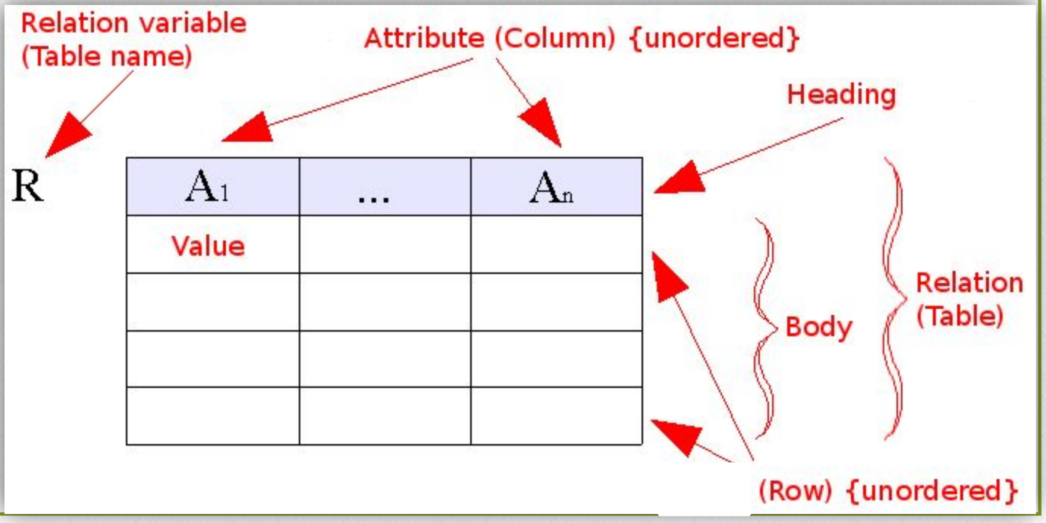
## Outline – Python Database Connectivity

- A Brief Review of Database
- Structured Query Language (SQL)
- MySQL with Python

## A Brief Review of Database

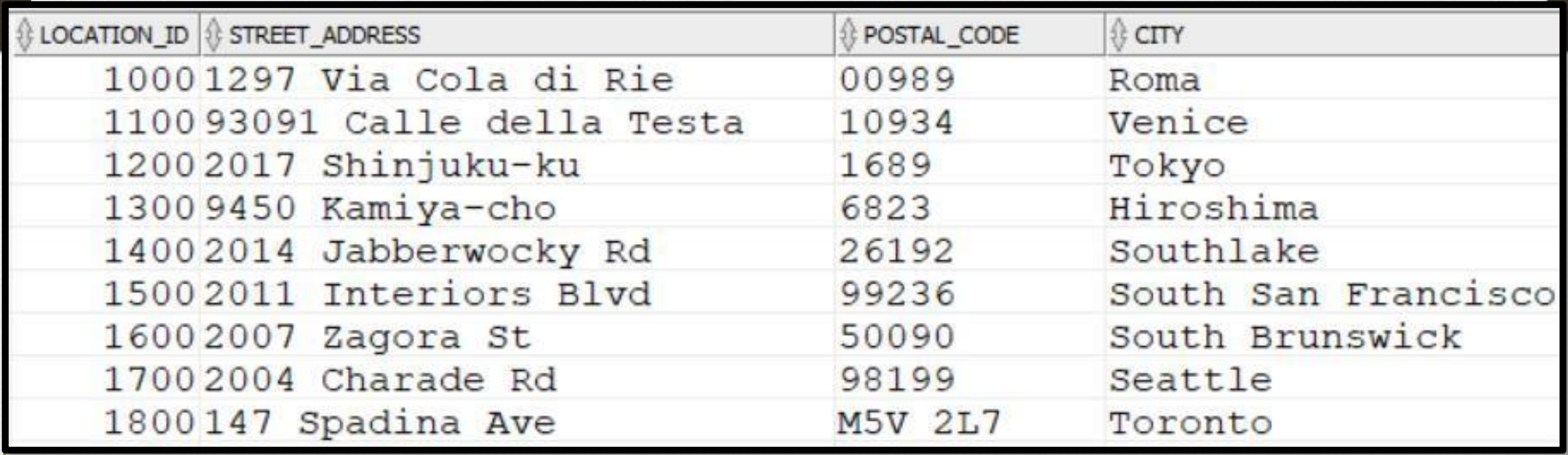
# Database

- **Database (DB):**
  - An organised collection of data
  - **Relational DBs** store data in tables
- **A table** in a database consists of:
  - rows & columns
  - Rows: records
  - Columns: attributes of the records
- **SQL** (Structured Query Language):
  - Is used for managing data in relational databases


Relation variable (Table name)
Attribute (Column) {unordered}
Heading
R
A1
...
An
Value
Body
Relation (Table)
(Row) {unordered}

# Database (cont.)

- An example of table: locations



| LOCATION_ID | STREET_ADDRESS | POSTAL_CODE | CITY |
|---|---|---|---|
| 1000 | 1297 Via Cola di Rie | 00989 | Roma |
| 1100 | 93091 Calle della Testa | 10934 | Venice |
| 1200 | 2017 Shinjuku-ku | 1689 | Tokyo |
| 1300 | 9450 Kamiya-cho | 6823 | Hiroshima |
| 1400 | 2014 Jabberwocky Rd | 26192 | Southlake |
| 1500 | 2011 Interiors Blvd | 99236 | South San Francisco |
| 1600 | 2007 Zagora St | 50090 | South Brunswick |
| 1700 | 2004 Charade Rd | 98199 | Seattle |
| 1800 | 147 Spadina Ave | M5V 2L7 | Toronto |

# Database Management System (DBMS)

- **DBMS:**
  - a set of software programs that controls the organization, storage, management, and retrieval of data in a database.
- **Popular DBMS:**

ORACLE®

Apache Derby

SQLite

Microsoft® SQL Server

MySQL®

DB2

PostgreSQL

SYBASE®

# Structured Query Language (SQL)

- **SQL:**
  - Data Definition Language (DDL): create the database and relational structures
  - Data Manipulation Language (DML): perform insertion, modification, deletion of data from relations;
- **Examples:**
  - SELECT NAME, ID FROM STUDENT WHERE GENDER='MALE';
  - INSERT INTO STUDENT VALUES ( 6, 'MARY', 'FEMALE' );
  - DELETE FROM STUDENT WHERE NAME='JERRY';
  - DROP TABLE STUDENT;

See more: https://www.w3schools.com/sql/

## Integrate MySQL with Python

1. Download and install the free MySQL database.
   https://www.mysql.com/downloads/
2. After installing the MySQL database, open your Command prompt.
3. Download and install "MySQL Connector"
   - pip install mysql-connector-python
4. Test MySQL Connector
   - import mysql.connector
5. Create Connection

More Details:
https://dev.mysql.com/doc/connector-python/en/connector-python-installation.html

## Create Connection

```
import mysql.connector

mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  password="yourpassword"
)
```

$\xrightarrow{}$ Connect MySQL server

## Create Database

```
mycursor = mydb.cursor()

mycursor.execute("CREATE DATABASE
mydatabase")
```

## Basic SQL Statements

## Create

```
mycursor.execute("CREATE TABLE Customers
(id INT AUTO_INCREMENT PRIMARY KEY,
name VARCHAR(255),
address VARCHAR(255))")
```

## Alter

```
mycursor.execute("ALTER TABLE Customers
ADD COLUMN id INT AUTO_INCREMENT PRIMARY
KEY")
```

## Insert

```
sql = "INSERT INTO Customers (name,
address) VALUES (%s, %s)"
val = ("John", "Highway 21")
mycursor.execute(sql, val)
```

A placeholder for string values

## Insert Multiple Rows

```
sql = "INSERT INTO customers (name, address) VALUES (%s, %s)"
val = [
  ('Peter', 'Lowstreet 4'),
  ('Amy', 'Apple st 652'),
  ('Hannah', 'Mountain 21'),
  ('Michael', 'Valley 345'),
  ('Sandy', 'Ocean blvd 2'),
  ('Betty', 'Green Grass 1'),
  ('Richard', 'Sky st 331'),
  ('Susan', 'One way 98'),
  ('Vicky', 'Yellow Garden 2'),
  ('Ben', 'Park Lane 38'),
  ('William', 'Central st 954'),
  ('Chuck', 'Main Road 989'),
  ('Viola', 'Sideway 1633')
]

mycursor.executemany(sql, val)

mydb.commit()
```

**mydb.commit()**

Ensures that all these changes are made permanent. If commit() is not called, the changes will not be saved, and they will be lost when the database connection is closed.

```
import mysql.connector

mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  password="yourpassword",
  database = "yourdatabasename",
  autocommit = True
)
```

## Select

```
mycursor.execute("SELECT * FROM customers")

myresult = mycursor.fetchall()
```

retrieve all rows of a query result set and return a list of tuples.

## Selecting Columns

```
mycursor.execute("SELECT name, address FROM customers")

myresult = mycursor.fetchall()
```

## Fetchone()

```
mycursor.execute("SELECT * FROM customers")

myresult = mycursor.fetchone()
```

return the first row of the result

## Where

```
sql = "SELECT * FROM customers WHERE address
='Park Lane 38'"

mycursor.execute(sql)

myresult = mycursor.fetchall()
```

## Sort

```
sql = "SELECT * FROM customers ORDER BY name
DESC/ASC"

mycursor.execute(sql)

myresult = mycursor.fetchall()
```

## Delete

```
sql = "DELETE FROM customers WHERE address = 'Mountain 21'"

mycursor.execute(sql)
```

## Drop

```
sql = "DROP TABLE IF EXISTS customers"

mycursor.execute(sql)
```

## Update

```
sql = "UPDATE customers SET address = 'Canyon
123' WHERE address = 'Valley 345'"

mycursor.execute(sql)
```

Limit the result to only include 5 rows. Skip the first two records and starts from the third record

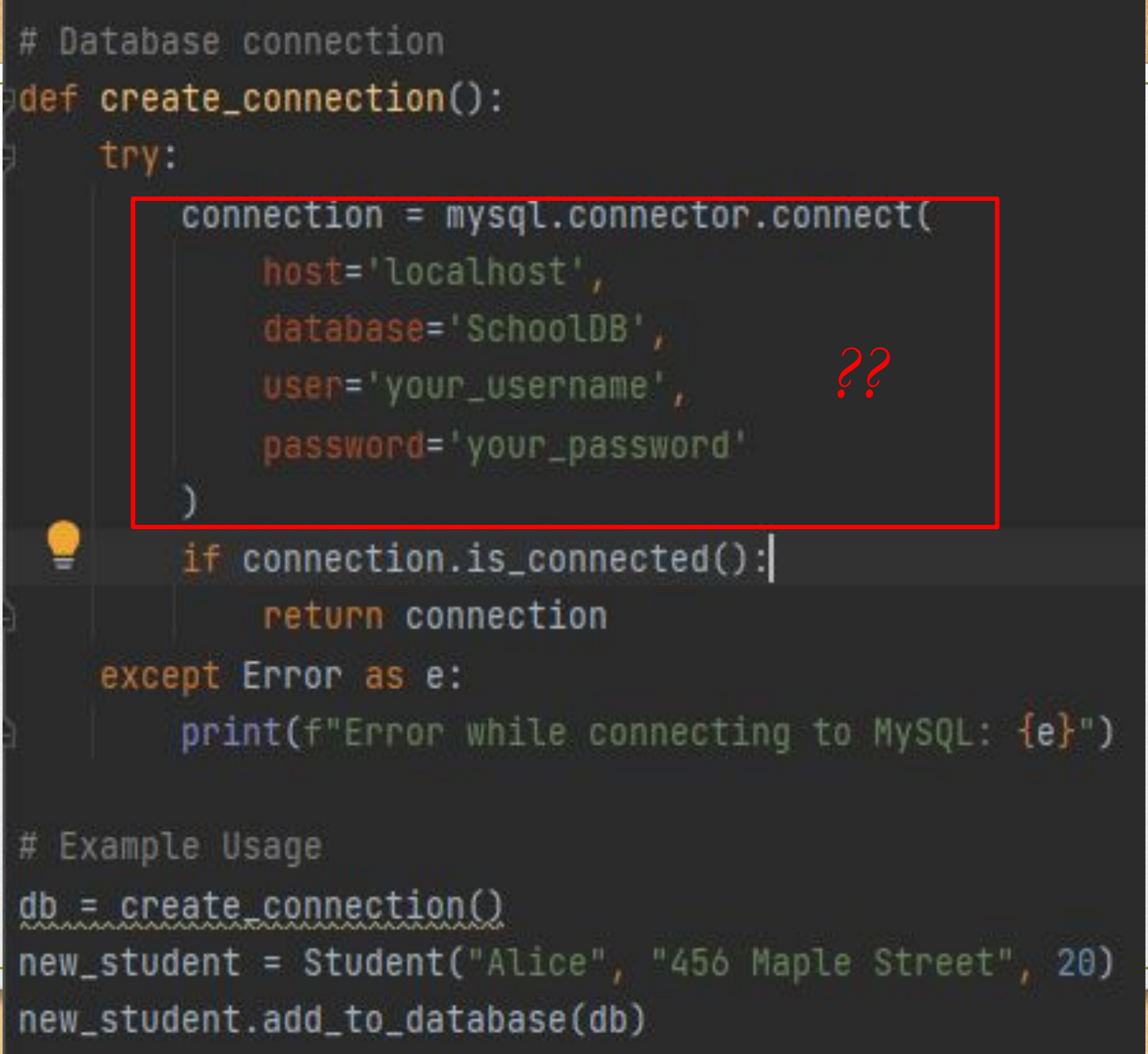
## Limit

```
mycursor.execute("SELECT * FROM customers LIMIT
5 OFFSET 2")

myresult = mycursor.fetchall()
```

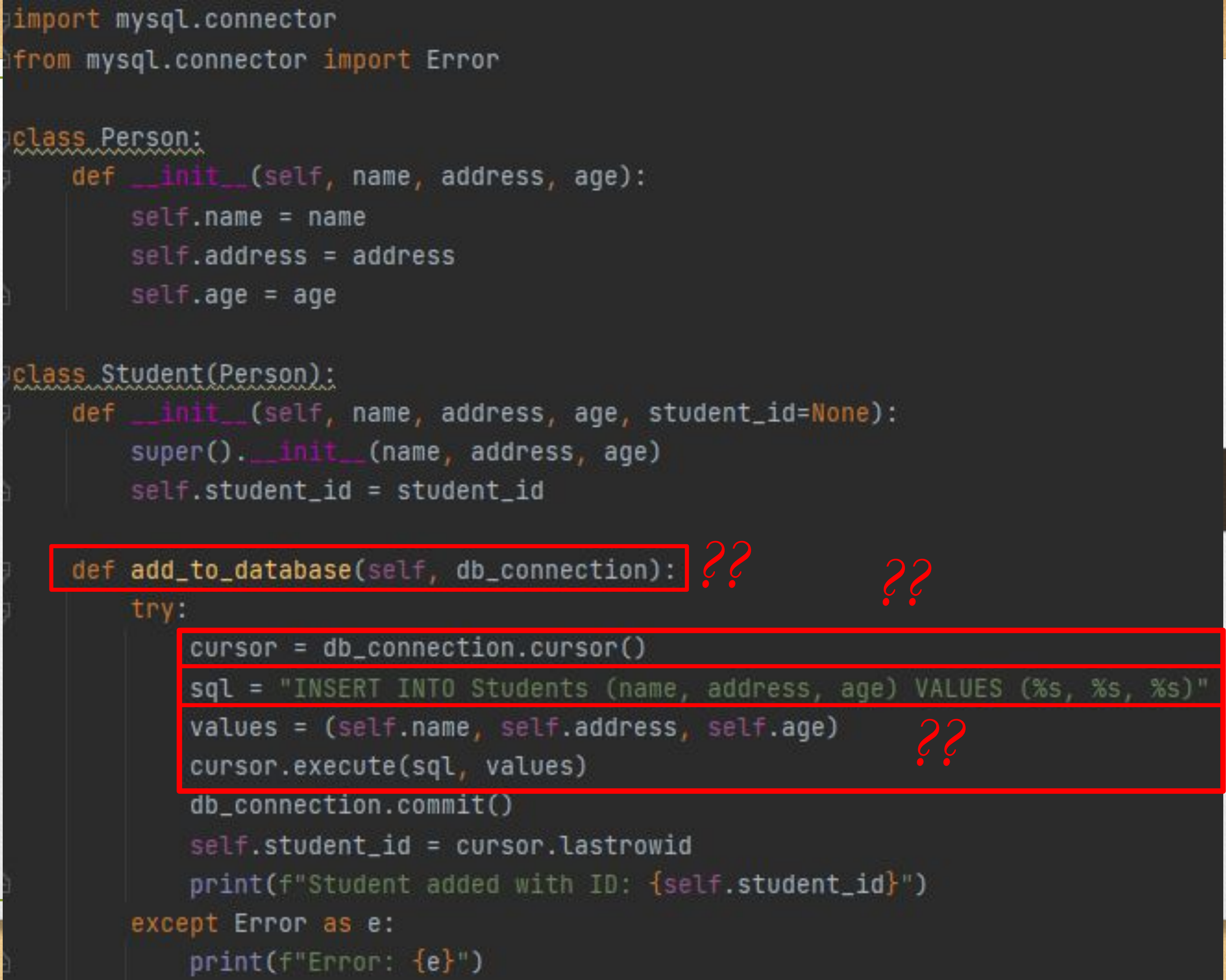
## Join

```
mycursor = mydb.cursor()

sql = "SELECT Orders.OrderID,
Customers.CustomerName, Orders.OrderDate
FROM Orders
INNER JOIN Customers ON Orders.CustomerID=Custom
ers.CustomerID;"

mycursor.execute(sql)

myresult = mycursor.fetchall()
```

## Examples



```python
# Database connection
def create_connection():
    try:
        connection = mysql.connector.connect(
            host='localhost',
            database='SchoolDB',
            user='your_username',
            password='your_password'
        )
        if connection.is_connected():
            return connection
    except Error as e:
        print(f"Error while connecting to MySQL: {e}")

# Example Usage
db = create_connection()
new_student = Student("Alice", "456 Maple Street", 20)
new_student.add_to_database(db)
```



```python
import mysql.connector
from mysql.connector import Error


class Person:
    def __init__(self, name, address, age):
        self.name = name
        self.address = address
        self.age = age


class Student(Person):
    def __init__(self, name, address, age, student_id=None):
        super().__init__(name, address, age)
        self.student_id = student_id

    def add_to_database(self, db_connection):
        try:
            cursor = db_connection.cursor()
            sql = "INSERT INTO Students (name, address, age) VALUES (%s, %s, %s)"
            values = (self.name, self.address, self.age)
            cursor.execute(sql, values)
            db_connection.commit()
            self.student_id = cursor.lastrowid
            print(f"Student added with ID: {self.student_id}")
        except Error as e:
            print(f"Error: {e}")
```

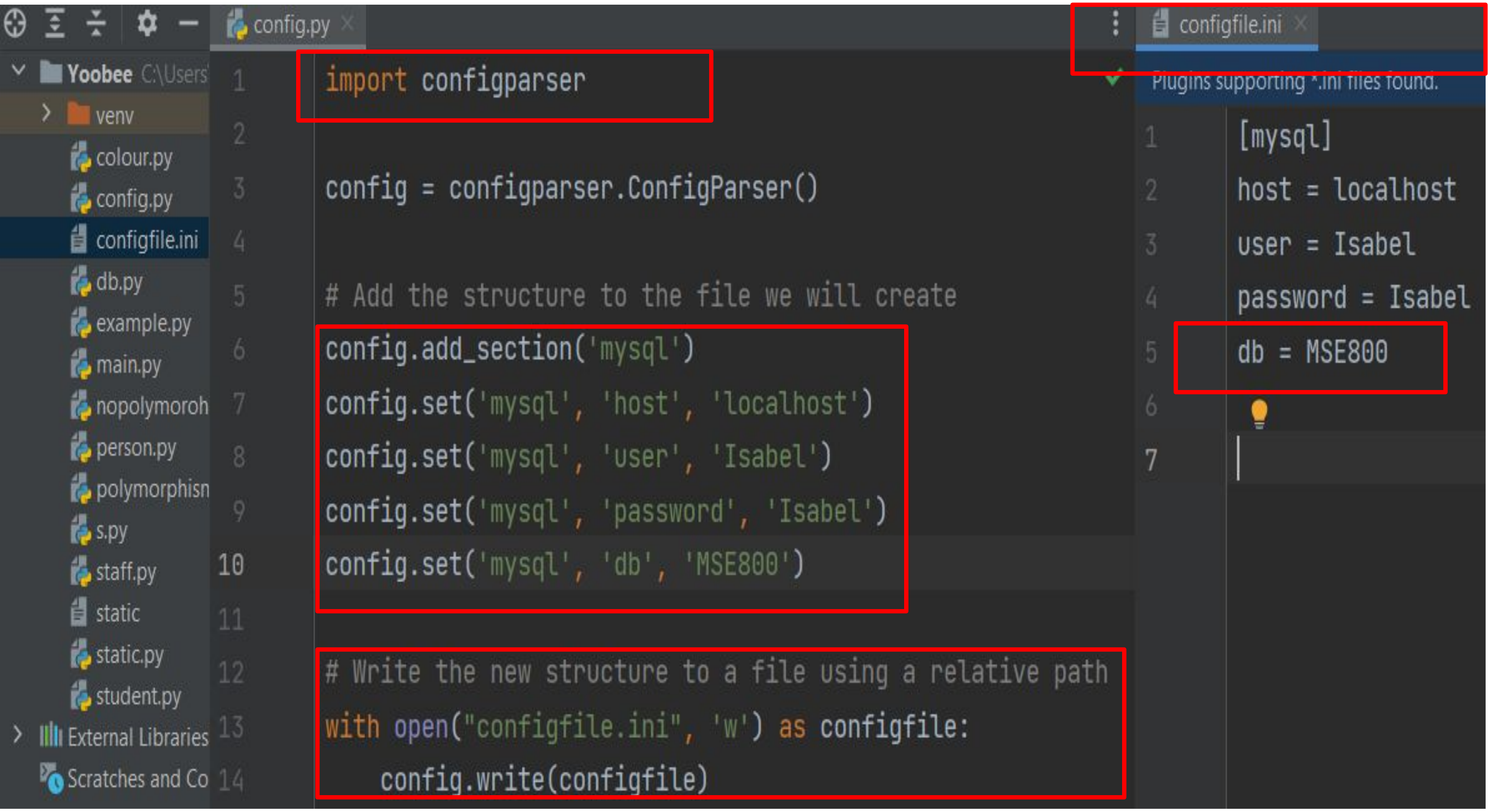
## Problems

1. Sensitive information such as usernames and passwords are hardcoded directly in the source code.
2. No database creation and existence checks are done in the application
    - Each time you deploy to a new environment, you need to manually create the database and tables. This adds a step to the deployment process
    - Programs that directly assume that the database and tables already exist and are available may crash immediately or generate a runtime error
3. Python and SQL statements show together, reduce readability.
4. Format

## Solution for first problem:

Store the database connection information in a configfile.ini file and then use Python to read the required data from this file.

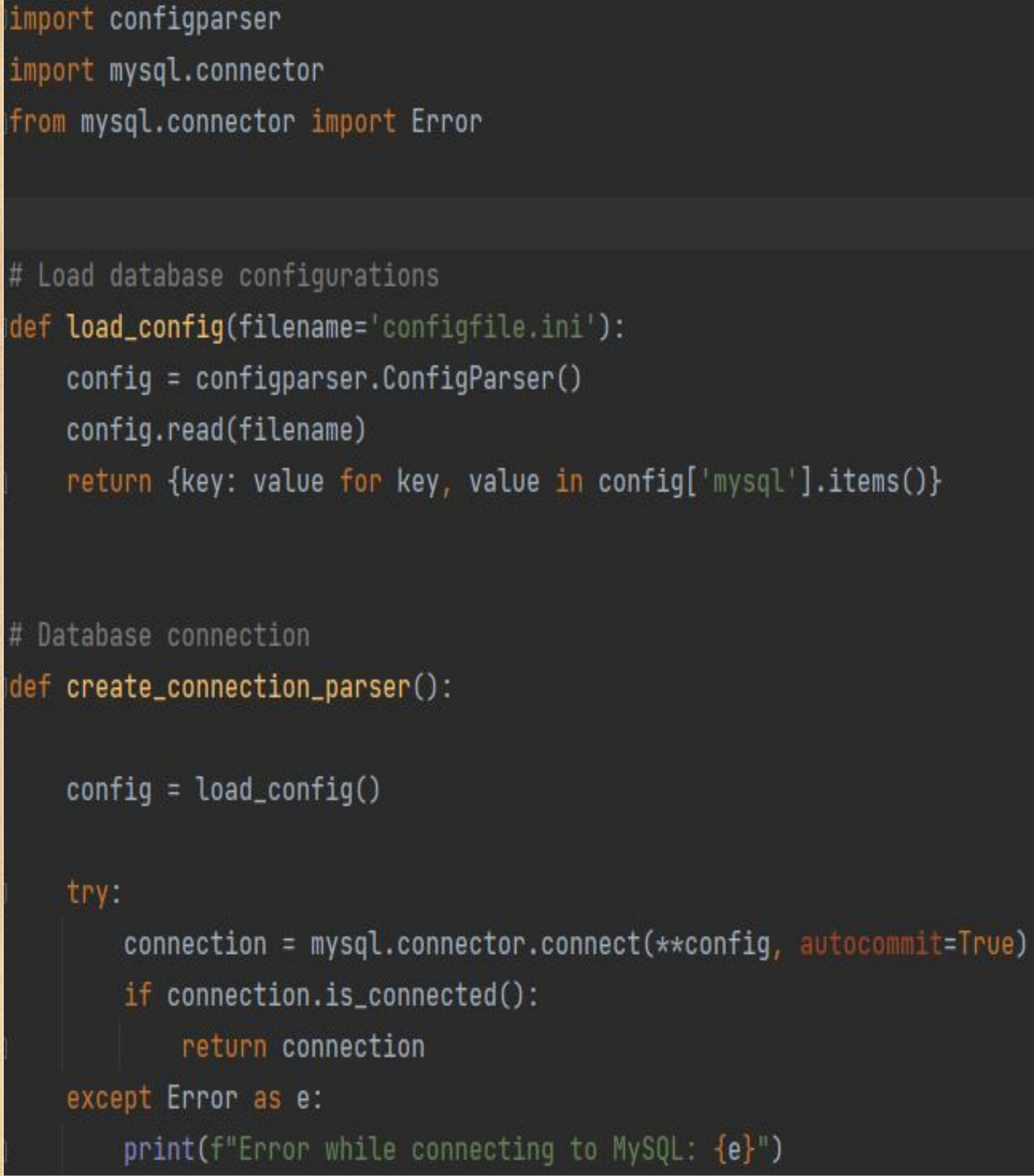
- In Python, we have a module called configparser.py which allows us to easily create, read and update INI files. Install it, using pip install configparser
- Each INI file you create will consist of sections within which data is stored using key-value pairs in much the same way as a **Python dictionary** datatype.



```python
import configparser

config = configparser.ConfigParser()

# Add the structure to the file we will create
config.add_section('mysql')
config.set('mysql', 'host', 'localhost')
config.set('mysql', 'user', 'Isabel')
config.set('mysql', 'password', 'Isabel')
config.set('mysql', 'db', 'MSE800')

# Write the new structure to a file using a relative path
with open("configfile.ini", 'w') as configfile:
    config.write(configfile)
```

```ini
[mysql]
host = localhost
user = Isabel
password = Isabel
db = MSE800
```

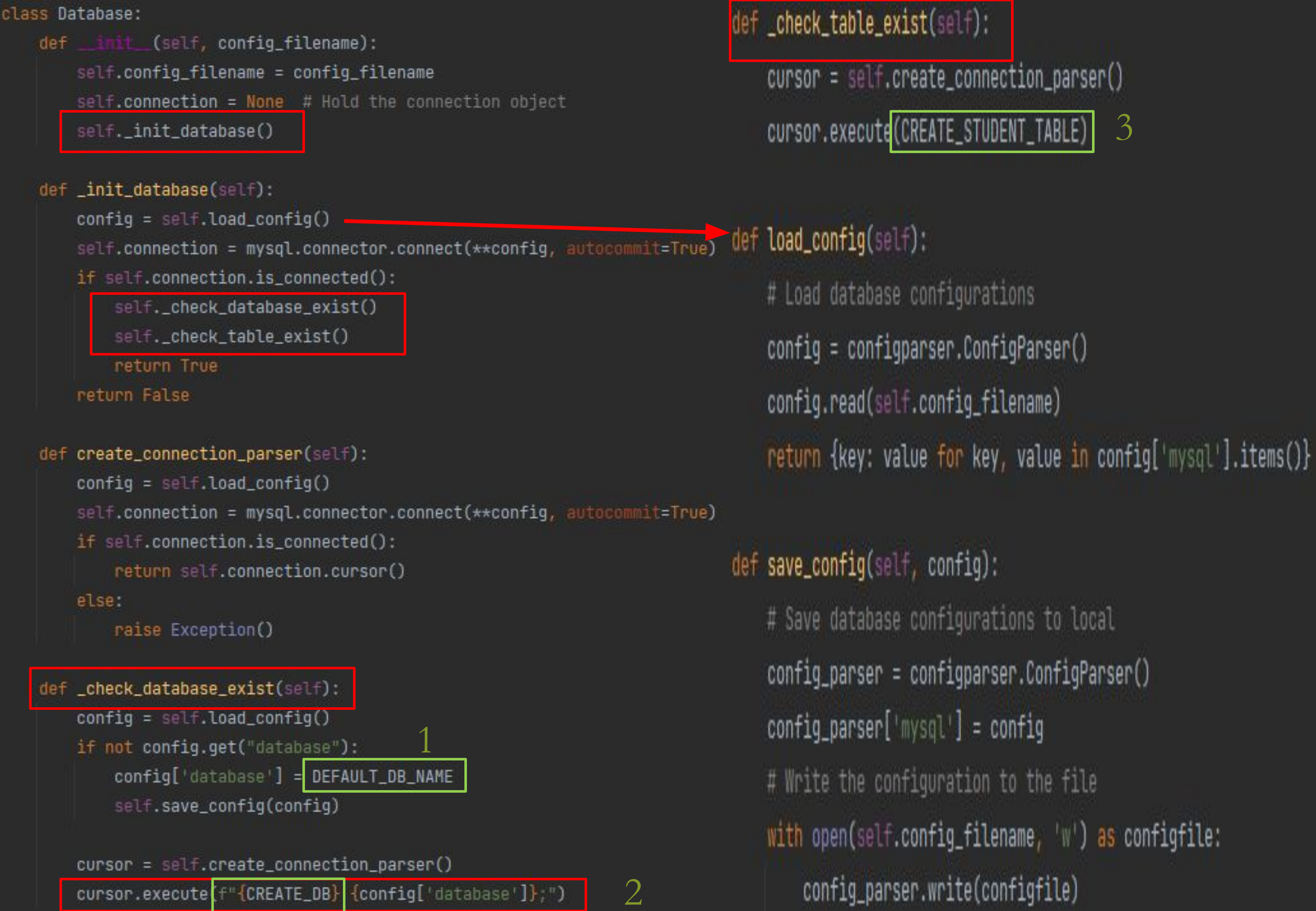
## Read data from INI file



```python
import configparser
import mysql.connector
from mysql.connector import Error


# Load database configurations
def load_config(filename='configfile.ini'):
    config = configparser.ConfigParser()
    config.read(filename)
    return {key: value for key, value in config['mysql'].items()}


# Database connection
def create_connection_parser():

    config = load_config()

    try:
        connection = mysql.connector.connect(**config, autocommit=True)
        if connection.is_connected():
            return connection
    except Error as e:
        print(f"Error while connecting to MySQL: {e}")
```

## Solution for second and fourth problems:

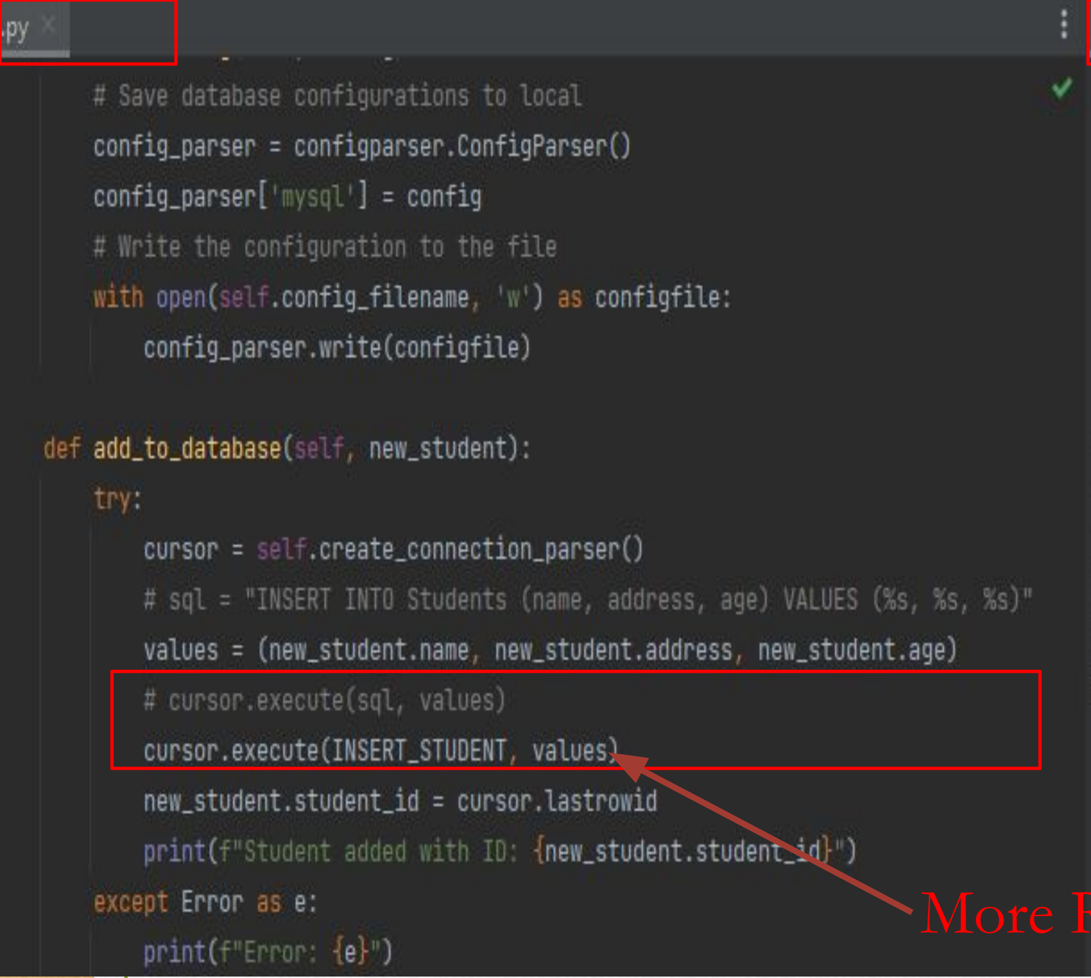
- Check the existence of databases and tables on initialization. If doesn’t exist, create one!



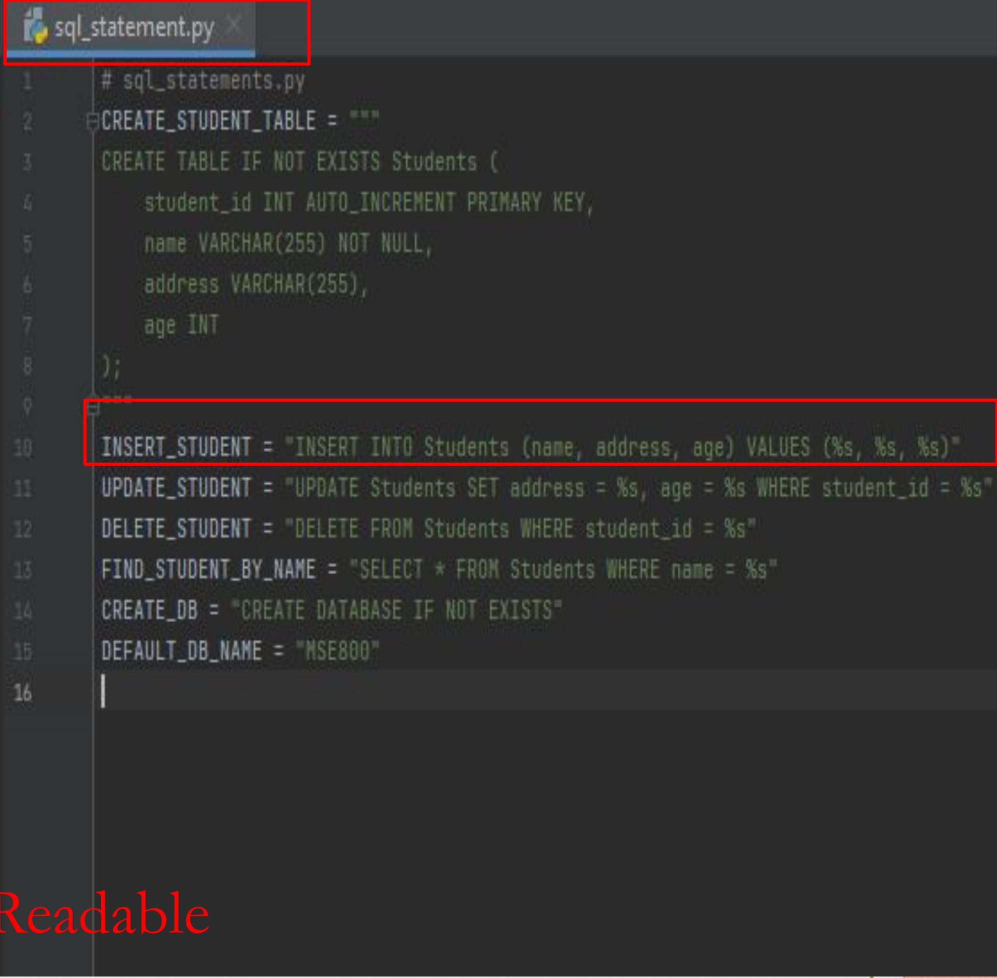
```python
class Database:
    def __init__(self, config_filename):
        self.config_filename = config_filename
        self.connection = None  # Hold the connection object
        self._init_database()

    def _init_database(self):
        config = self.load_config()
        self.connection = mysql.connector.connect(**config, autocommit=True)
        if self.connection.is_connected():
            self._check_database_exist()
            self._check_table_exist()
            return True
        return False

    def create_connection_parser(self):
        config = self.load_config()
        self.connection = mysql.connector.connect(**config, autocommit=True)
        if self.connection.is_connected():
            return self.connection.cursor()
        else:
            raise Exception()

    def _check_database_exist(self):
        config = self.load_config()
        if not config.get("database"):                  1
            config['database'] = DEFAULT_DB_NAME
            self.save_config(config)

        cursor = self.create_connection_parser()
        cursor.execute(f"{CREATE_DB} {config['database']};")      2
```

```python
    def _check_table_exist(self):
        cursor = self.create_connection_parser()
        cursor.execute(CREATE_STUDENT_TABLE)     3

    def load_config(self):
        # Load database configurations
        config = configparser.ConfigParser()
        config.read(self.config_filename)
        return {key: value for key, value in config['mysql'].items()}

    def save_config(self, config):
        # Save database configurations to local
        config_parser = configparser.ConfigParser()
        config_parser['mysql'] = config
        # Write the configuration to the file
        with open(self.config_filename, 'w') as configfile:
            config_parser.write(configfile)
```

## Solution for third problem:

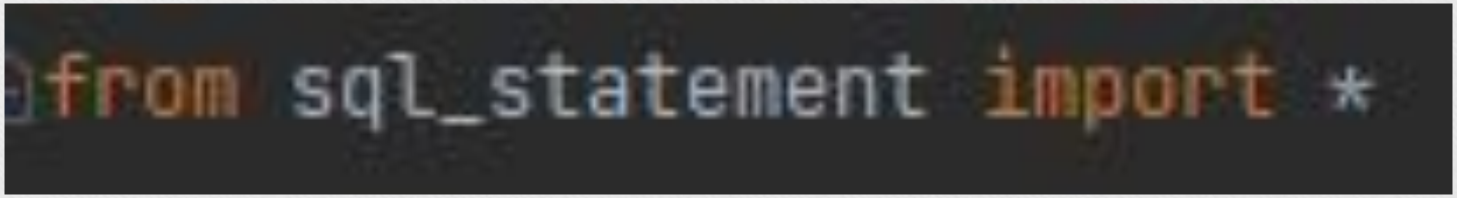


```
.py

    # Save database configurations to local
    config_parser = configparser.ConfigParser()
    config_parser['mysql'] = config
    # Write the configuration to the file
    with open(self.config_filename, 'w') as configfile:
        config_parser.write(configfile)

def add_to_database(self, new_student):
    try:
        cursor = self.create_connection_parser()
        # sql = "INSERT INTO Students (name, address, age) VALUES (%s, %s, %s)"
        values = (new_student.name, new_student.address, new_student.age)
        # cursor.execute(sql, values)
        cursor.execute(INSERT_STUDENT, values)
        new_student.student_id = cursor.lastrowid
        print(f"Student added with ID: {new_student.student_id}")
    except Error as e:
        print(f"Error: {e}")
```



```
sql_statement.py

# sql_statements.py
CREATE_STUDENT_TABLE = """
CREATE TABLE IF NOT EXISTS Students (
    student_id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    address VARCHAR(255),
    age INT
);
"""
INSERT_STUDENT = "INSERT INTO Students (name, address, age) VALUES (%s, %s, %s)"
UPDATE_STUDENT = "UPDATE Students SET address = %s, age = %s WHERE student_id = %s"
DELETE_STUDENT = "DELETE FROM Students WHERE student_id = %s"
FIND_STUDENT_BY_NAME = "SELECT * FROM Students WHERE name = %s"
CREATE_DB = "CREATE DATABASE IF NOT EXISTS"
DEFAULT_DB_NAME = "MSE800"
```

More Readable



```
from sql_statement import *
```

## Exercises

## Exercise： DataBase

Assume you are a database administrator for a YooBee, and you need to develop a simple Python application to manage student information. This application should be able to perform the following functions:

1. **Add a New Student**: Add a new student's name, address, and age to the database.
2. **Update Student Information**: Update the address and age of a student based on the student's ID.
3. **Delete a Student**: Remove a student record based on the student's ID.
4. **Query Student Information**: Retrieve all information for a student based on the student's name.

Thank you