Connecting to the Database Using Python

To get the Connection instance, you should create an object using:

```
conn = Connector.DBConnector()
```

(after importing "import Utility.DBConnector as Connector" as in Example.py).

To submit a query to your database, simply perform:

```
conn.execute("query here")
```

This function will return the number of affected rows and a ResultSet object.
You can find the full implementation of ResultSet in DBConnector.py.
Here are a few examples of how to use it:

```
conn = Connector.DBConnector()
query = "SELECT * FROM customers"
# execute the query, result will be a ResultSet object
rows_affected, result = conn.execute(query)

# get the first tuple in the result as a dictionary, the keys are the column
# names, the value are the values of the tuple
tuple = result[0]

# get the id column from the first tuple in the result
customer_id = result[0]['id']

# get the id column as a list
id_column = result['id']

# get the id of the first tuple
tuple_id = column[0]

# iterate over the rows of the result set. tuple is a dictionary, same as the
# example above
for tuple in result:
    # do something
    pass
```

This will return a tuple of (number of rows affected, results in case of SELECT).
Make sure to close your session using:

```
conn.close()
```

SQL Exceptions

When preparing or executing a query, an SQL Exception might be thrown. It is thus needed to use the try/catch (try/except in python) mechanism to handle the exception. For your convenience, the DatabaseException enum type has been provided to you. It captures the error codes that can be returned by the database due to error or inappropriate use. The codes are listed here:
NOT_NULL_VIOLATION (23502), *FOREIGN_KEY_VIOLATION*(23503), *UNIQUE_VIOLATION*(23505), *CHECK_VIOLIATION* (23514);

To check the returned error code, the following code should be used inside the except block: (here we check whether the error code *CHECK_VIOLIATION* has been returned)

```
except DatabaseException.CHECK_VIOLATION as e:
  # Do stuff
  print(e)
```

Notice you can print more details about your errors using print(e).

Dates in SQL

PostgreSQL provides several data types for storing date and time information. One of the most versatile is TIMESTAMP, which stores both the date and time.

Creating a TIMESTAMP that represents the 6th of May 2023 at 14:30:00

This line returns a table with a single row and a single column that has type timestamp and a value of the date 6/5/2023.

```
SELECT TIMESTAMP '2023-05-06 14:30:00'
```

The returned table:

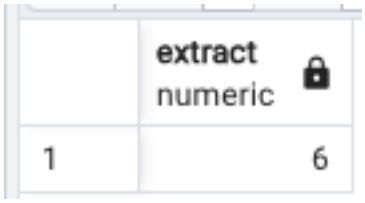
	timestamp timestamp without time zone
1	2023-05-06 14:30:00

You can extract individual components from a timestamp using the EXTRACT function.

This line returns a table with a single row and a single column that has type numeric(also called decimal) and a value of the day from the given timestamp:

```
SELECT EXTRACT(DAY FROM TIMESTAMP '2023-05-06 14:30:00');
```

The returned table:



	extract numeric
1	6

The next two lines will return similar tables with values of month/year/hour/minute/second

```
SELECT EXTRACT(MONTH FROM TIMESTAMP '2023-05-06 14:30:00');
SELECT EXTRACT(YEAR FROM TIMESTAMP '2023-05-06 14:30:00');
SELECT EXTRACT(HOUR FROM TIMESTAMP '2023-05-06 14:30:00');
SELECT EXTRACT(MINUTE FROM TIMESTAMP '2023-05-06 14:30:00');
SELECT EXTRACT(SECOND FROM TIMESTAMP '2023-05-06 14:30:00');
```

Dates in Python

This assignment will be done in python. Because you will be working with timestamps, you will need to use the datetime object from the datetime module in python.

How to use date:

Import:

```
from datetime import datetime
```

Basic constructor:

```
# format: datetime(year: int, month: int, day: int, hour: int, minute: int, second: int)
# example: 24/12/2023 14:30:00
dt = datetime(2023, 12, 24, 14, 30, 0)
```

Basic getters:

```
print(dt.year)   # prints "2023"
print(dt.month)  # prints "12"
print(dt.day)    # prints "24"
print(dt.hour)   # prints "14"
print(dt.minute) # prints "30"
print(dt.second) # prints "0"
```

date.strftime:

A function for getting a string of the datetime object in a specified format. The function uses codes for different representations of year, month, day and time.

A link to a list of all codes: Python strftime reference cheatsheet

Usage example:

```
print(dt.strftime('%Y-%m-%d %H:%M:%S'))  # prints "2023-12-24 14:30:00"
print(dt.strftime('%A %B %d %Y %I:%M %p'))  # prints "Sunday December 24 2023 02:30 PM"
```

datetime.strptime:
This function is the inverse of strftime. It takes a format and a string and converts it into a datetime object. It uses the same codes as strftime.
Usage example:

```
t = datetime.strptime('2023-12-24 14:30:00', '%Y-%m-%d %H:%M:%S')
print(t)  # prints "2023-12-24 14:30:00"
print(type(t))  # prints "<class 'datetime.datetime'>"
```

Further reading: https://docs.python.org/3.11/library/datetime.html

Tips

1. Create auxiliary functions that convert a record of ResultSet to an instance of the corresponding business object.
2. Use the enum type DatabaseException. It is highly recommended to use the exceptions mechanism to validate Input, rather than use Python's “if else”.
3. Devise a convenient database design for you to work with.
4. Before you start programming, think which Views you should define to avoid code duplication and make your queries readable and maintainable. (Think which sub-queries appear in multiple queries).
5. Use the constraints mechanisms taught in class to maintain a consistent database. Use the enum type DatabaseException in case of violation of the given constraints.
6. Remember - you are also graded on your database design (tables, views).
7. Please review and run Example.py for additional information and implementation methods.
8. We highly recommend to use views.