ΠΕΡΙΕΧΟΜΕΝΑ:

1. Εισαγωγή - Ενημέρωση - Διαγραφή Δεδομένων (CRUD)
2. Κλήση αποθηκευμένων διαδικασιών
3. Εργασία με BLOBs
4. Εργασία με Ημερομηνίες
5. Μία Άσκηση - Πολλά Προβλήματα

ΠΡΟΑΠΑΙΤΟΥΜΕΝΑ:

1. Python Modules: Μάθημα 3.1 - datetime

Ευάγγελος Μ.

Σμαραγδένιος Χορηγός Μαθήματος

Σπύρος Α.

Ασημένιος Χορηγός Μαθήματος

Εισαγωγή δεδομένων στη βάση

- Γράφουμε ένα INSERT query (βλ. SQL - Μάθημα 1.11) και το αποθηκεύουμε σε μία συμβολοσειρά.
- Τρέχουμε το query μέσω ενός **cursor**, με τη μέθοδο **execute**.
- Αν υπάρχει πρωτεύον κλειδί με autoincrement, η τιμή του (της νέας εγγραφής) αποθηκεύεται στο **cursor.lastrowid**.
- Προσοχή! Ο **connector** πρέπει να κάνει commit (μέθοδος **commit()**) στο τέλος, αλλιώς δεν θα αποθηκευτεί η εισαγωγή της νέας εγγραφής.

Παράδειγμα 1: insert

```
def insert_query(connection, query):
    cursor = connection.cursor()
    cursor.execute(query)
    lastrowid = None
    if cursor.lastrowid:
        lastrowid = cursor.lastrowid
    cursor.close()
    return lastrowid
```

```
query = """INSERT INTO city (Name, countryCode, District, Population)
           VALUES('Chania','GRC', 'Crete', 100000)"""
id = insert_query(connector,query)
print("Row inserted with id: " + str(id))
connector.commit()
```

Διαγραφή - Ενημέρωση Δεδομένων

- Εντελώς αντίστοιχα γράφουμε ένα DELETE ή UPDATE query (βλ. SQL - Μάθημα 1.12) και το αποθηκεύουμε σε μία συμβολοσειρά.
- Ακολουθούμε την ίδια διαδικασία:
  - **cursor.execute(query)**
  - **connector.commit()**

Παράδειγμα 2: upd_del

```
def update_query(connection, query):
    cursor = connection.cursor()
    cursor.execute(query)
    cursor.close()

def delete_query(connection, query):
    cursor = connection.cursor()
    cursor.execute(query)
    cursor.close()
```

```
query = "UPDATE city SET name='Χανιά' WHERE name='Chania"

update_query(connector,query)
connector.commit()

query = "DELETE FROM city WHERE name='Χανιά'"

delete_query(connector, query)
connector.commit()
```

Για να εκτελέσουμε μία αποθηκευμένη διαδικασία (stored procedure)

- Κατασκευάζουμε έναν cursor
- Εκτελούμε την stored procedure με την **callproc()** της cursor:
  - 1ο όρισμα: Το όνομα της stored procedure
  - 2ο όρισμα: Μία λίστα με τα ορίσματα της stored procedure.
- Αν επιστρέφει αποτελέσματα:
  - Η **stored_results()** του cursor επιστρέφει iterator
  - Με τη **fetchall()** παίρνουμε το result set.

Παράδειγμα 3: MySQL: stored.sql

```
CREATE PROCEDURE country_cities(
                IN in_country_name CHAR(52)
)
BEGIN
   SELECT ct.name, ct.district, ct.population
   FROM country cn JOIN city ct
        ON cn.code = ct.countrycode
   WHERE cn.name = in_country_name;
END
```

Παράδειγμα 3: python: stored_proc

```
def call_stored(connection, stored_proc_name, list_args):
   cursor = connection.cursor(dictionary=True)
   cursor.callproc(stored_proc_name, list_args)

   results = []
   for result in cursor.stored_results():
      results += result.fetchall()

   cursor.close()
   return results
```

```
results = call_stored(connector, 'country_cities', ['Greece'])

for result in results:
   print(result)
```

3. Εργασία με BLOBs

Για να εισάγουμε BLOB (Binary Large Objects)

- (μέσω αυτών αποθηκεύουμε στη βάση οποιοδήποτε τύπο αρχείων (π.χ. εικόνες, βίντεο κ.λπ.)
- Ανοίγουμε το αρχείο και αποθηκεύουμε τα περιεχόμενα του σε δυαδική μορφή (βλ. Python - μάθημα 15) σε κάποια μεταβλητή.
- Ανοίγουμε έναν cursor
- Χρησιμοποιούμε μία ειδική μορφή της cursor.execute():

```
cursor.execute(query, insert_tuple)
```

  - Το query είναι τυπικό insert query όπου όμως, αντί για τιμές, θέτουμε τον προσδιοριστή %s
  - Το insert_tuple είναι tuple με τις διαδοχικές τιμές που θα αντικαταστήσουν τους προσδιοριστές.

Σημείωση:

- Αυτή η μορφή της execute() μπορεί να χρησιμοποιηθεί σε οποιοδήποτε query, ανεξάρτητα αν περιέχονται BLOBs ή όχι

Παράδειγμα 4: MySQL: create_table_images.sql, Python: blobs

```
DROP TABLE IF EXISTS images;

CREATE TABLE images (
        image_id INT PRIMARY KEY AUTO_INCREMENT,
        image BLOB,
        descr VARCHAR(100)
);
```

```
def insert_with_blob(connection, filename, descr):
    with open(filename, "rb") as f:
        contents = f.read()
    query = "INSERT INTO images (image, descr) VALUES(%s, %s)"
    insert_tuple = (contents, descr)
    cursor = connection.cursor()
    cursor.execute(query, insert_tuple)
    cursor.close()
```

```
insert_with_blob(connector, "bs.png", "beautiful soup logo")
connector.commit()
```

Για να διαβάσουμε BLOB

- Ακολουθούμε τη συνηθισμένη διαδικασία όπως σε ένα τυπικό SELECT. Το επιστρεφόμενο αποτέλεσμα θα είναι ακολουθία από bytes (και μπορούμε π.χ. να την αποθηκεύσουμε σε ένα αρχείο)

```
def read_blob(connection, id):
    results = query_full_results(connection, "SELECT * FROM images
                                              WHERE image_id=" + str(id))
    print(results)
    return results[0]["image"]
```

```
res = read_blob(connector,1)
with open("copy.png", "wb") as f:
    f.write(res)
```

Στα παραδείγματα, έχουμε δει μέχρι τώρα:

- Οι **ακέραιοι** της MySQL μετατρέπονται σε **int** στην Python
- Οι τύποι **float**, **double** της MySQL μετατρέπονται σε **float** στην Python
- Οι τύπος **DECIMAL**(M,N) της MySQL μετατρέπεται σε **Decimal** στην Python
- Οι συμβολοσειρές της MySQL αποτυπώνονται στον τύπο **str** στην Python.

Για τους τύπους δεδομένων ημερομηνίας/ώρας της MySQL:

- Ο τύπος DATE μετατρέπεται σε date
- Ο τύπος TIME μετατρέπεται σε timedelta
- Ο τύπος DATETIME μετατρέπεται σε datetime
- Ο τύπος TIMESTAMP μετατρέπεται σε datetime

Παράδειγμα 5: MySQL: dates.sql, Python: dates

```
DROP TABLE IF EXISTS dates;

CREATE TABLE dates (
        vdate DATE,
        vtime TIME,
        vdatetime DATETIME,
        vtimestamp TIMESTAMP DEFAULT NOW());

INSERT INTO dates
VALUES(CURRENT_DATE(), CURRENT_TIME(), NOW());
```

```
def insert_with_tuple(connection, query, insert_tuple):
    cursor = connection.cursor()
    cursor.execute(query, insert_tuple)
    cursor.close()
```

```
query = "SELECT * FROM dates"
results = query_full_results(connector, query)

vdate = results[0]["vdate"]
print("A date: " + vdate.strftime("%d.%m.%Y"))

td = results[0]["vtime"]
vtime = time(td.seconds//3600, (td.seconds%3600)//60, td.seconds%60)
print("A time: " + str(vtime))

vdatetime = results[0]["vdatetime"]
print("A datetime: " + vdatetime.strftime("%d.%m.%Y %H:%M:%S"))

vtimestamp = results[0]["vtimestamp"]
print("A timestamp: " + vtimestamp.strftime("%d.%m.%Y %H:%M:%S"))

query = "INSERT INTO dates(vdate, vtime, vdatetime) VALUES (%s, %s, %s)"
tuple_values = (datetime.now().date(), datetime.now().time(),
datetime.now())
insert_with_tuple(connector, query, tuple_values)
connector.commit()
```

ΜΑΘΗΜΑ 2: CRUD, Τύποι Δεδομένων και Επεκτάσεις 5. Μία Άσκηση - Πολλά Προβλήματα

Άσκηση 1 (βλ. υποχρεωτικά και βίντεο)

Κατασκευάστε πρόγραμμα με τις εξής επιλογές:

1. Εισαγωγή χώρας
2. Διάβασμα χώρας
3. Ενημέρωση χώρας
4. Διαγραφή χώρας

Προβληματισμοί:

- Η κλάση περιτυλιχτής της χώρας της προτεινόμενης επίλυσης, περιέχει «ανιαρό» κώδικα. Σκεφθείτε μια μεγάλη βάση δεδομένων με πολλούς πίνακες, ότι θα είχε πολύ παρόμοιο κώδικα σε πολλούς πίνακες.
- Η περισσότερη δουλειά γίνεται για την επικοινωνία της βάσης δεδομένων με το πρόγραμμα (μετατροπές τύπων δεδομένων, διαχείριση NULL κ.λπ.)

Επεκτάσεις - προσθήκες στην προτεινόμενη επίλυση:

- Δεν γίνεται **συστηματικός έλεγχος εισόδου** (όπως είχαμε δει στα μαθήματα της βασικής σειράς), π.χ. με συναρτήσεις τύπου getInteger()
- Ο έλεγχος είναι απαιτητός, αλλιώς υπάρχει κίνδυνος για **κακόβουλες χρήσεις** του προγράμματος (π.χ. SQL injection)
- Δεν γίνεται έλεγχος στο πεδίο Continent (το οποίο οφείλει να παίρνει συγκεκριμένες τιμές από το enum τύπο δεδομένων της βάσης)
- Μεγάλα περιθώρια για **συντακτικές αστοχίες**, πράγματα που δεν έχουμε προβλέψει κ.ο.κ.

- Είναι προφανές, ότι ο κώδικας διαχείρισης μιας βάσης, θα έχει δουλευτεί πάρα πολύ από την προγραμματιστική κοινότητα.
- Πράγματι, ο προβληματισμός αυτός έχει οδηγήσει σε πακέτα λογισμικού που κάνουν με μεγάλη συστηματικότητα τη διαχείριση της βάσης και επί των θεμάτων αυτών, αλλά και άλλων που προκύπτουν στην πράξη.
- Τα πακέτα αυτά **(ORM - object relational mapping)** εστιάζουν στο εύκολο δέσιμο των στοιχείων της βάσης με αντικείμενα της γλώσσας προγραμματισμού
  - και συνακόλουθα είναι εφικτό να γράφουμε πιο εύκολα κώδικα διαχείρισης και χρήσης της Βάσης Δεδομένων
- **Στην Python τα πιο δημοφιλή ORM είναι τα:**
  - **SQLAlchemy** (προτεινόμενη επόμενη μίνι-σειρά προς μελέτη)
  - PeeWee, Django ORM, SQL Object, Tortoise ORM

Βιβλιογραφία μίνι σειράς Python + MySQL:

- "MySQL Connector/Python Revealed: SQL and NoSQL Data Storage Using MySQL for Python Programmers", Jesper W. Krogh, 2018
- https://dev.mysql.com/doc/connector-python/en/
- https://www.mysqltutorial.org/getting-started-mysql-python-connector/
- https://realpython.com/python-mysql/
- www.stackoverflow.com