1. Python und SQL

1.1. Einleitung

Python ist über Libraries zu allen gängigen DBMS[1] kompatibel.
Eine spezielle Anbindung ist zu SQLite vorhanden, da die notwendige Bibliothek bereits im Sprachumfang von Python enthalten ist.

1.2. SQLAlchemy

SQLAlchemy bildet eine Abstraktionsebene sowie die einfache Austauschbarkeit des eingesetzen DBMS.
SQLAlchemy[2] ist ein OpenSource (MIT-Lizenz) SQL-Toolkit und ein ORM für Python. Im Unterschied von Ruby on Rails unterstützt SQLAlchemy das Data Mapper Pattern (ähnlich wie Hibernate für Java).
SQLAlchemy kennt Transaktionen und das ACID-Prinzip[3] womit sämtliche Änderung an einer Datenbank bzw. Tabelle innerhalb einer datenbank-spezifischen Session erfolgen müssen. Diese kontrolliert den Lebenslauf jedes Objektes innerhalb der Session. Für die genaue Dokumentation aller Funktionalitäten von SQLAlchemy[4] wird auf die entsprechende Seite verwiesen.

1.2.1. Python als Ausgangspunkt

```
# base class of all declarative classes
Base = declarative_base()

# table person
class Person(Base):
    __tablename__ = 'person'
    # pk
    id = Column(Integer, primary_key=True)
    # attribute
    name = Column(String)

# table address
class Address(Base):
    __tablename__ = 'address'
    # pk
    id = Column(Integer, primary_key=True)
    # attribute
    address = Column(String)
    # attribute and  foreign key
    person_id = Column(Integer, ForeignKey(Person.id))
    person = relationship(Person)

# create_engine
engine = create_engine('sqlite:///')

# create session
s = Session(engine)
```

[1] Database Management System
[2] Siehe auch http://pythoncentral.io/sqlalchemy-vs-orms/
[3] ACID (Atomicity, Consistency, Isolation, Durability)
[4] http://www.sqlalchemy.org/

```
Base.metadata.create_all(engine)

# create an instance of person (record in table person)
p = Person(name='mayer')
# add it to the session (SQL INSERT statement)
s.add(p)
# create an instance of address (record in table address)
a = Address(address='Baumgasse 4', person=p)
# add it to the session (SQL INSERT statement)
s.add(a)

# get the first record of table person
p = s.query(Person).one()

# print query
print ("%r, %r" % (p.id, p.name))
# 1, 'mayer'

# get the corresponding record of table address
a = s.query(Address).filter(Address.person == p).one()
print ("%r, %r" % (a.id, a.address))
# 1, 'Baumgasse 4'

# commit the current transaction
s.commit()
# close the current transaction
s.close()
```

1.2.2. DBMS als Ausgangspunkt

```
db = 'mysql+mysqldb://wienwahl:wienwahl@localhost/wienwahl?charset=utf8'
engine = create_engine(db)
conn = engine.connect()

# create declarative base class
Base = automap_base()
# create declarative classes from dbms
Base.prepare(engine, reflect=True)
# create references for all reflected tables
wahl = Base.classes.wahl

s = Session(engine)
try:
# create an instance of wahl (record in table wahl)
      w = wahl(wtermin = 'GR21', mandate = 1000)
      # add it to the session (SQL INSERT statement)
      s.add(w)
    s.commit()
except:
    s.rollback()
    raise
finally:
    s.close()
```