Database Systems

Application Development

H. Turgut Uyar   Şule Öğüdücü

2002-2015

License

Topics

Introduction

- how to carry out data statements in application code?
- connect to the database server
- provide credentials
- carry out operations
- adapt results
- disconnect

Goals

- code shouldn’t be tied to a specific product
- easy to port to another product

- abstraction layers cause performance issues
- for example, ODBC is standard but slow

- languages define standard interfaces for drivers to implement
- Java: JDBC, Python: DBAPI

Python DBAPI

- import driver module
- rename for easier porting to other drivers

example

```
import psycopg2 as dbapi2
# import sqlite3 as dbapi2
```

Connection

- connection info: username, password, host, port, database name

- data source name (DSN):
  `user=..  password=..  host=..  port=..  dbname=..`
- uniform resource identifier (URI):
  `protocol://user:password@host:port/dbname`

examples

```
user='vagrant' password='vagrant' host='localhost'
    port=5432 dbname='itucsdb'

postgres://vagrant:vagrant@localhost:5432/itucsdb
```

Connection Example

```
dsn = """user='vagrant' password='vagrant'
         host='localhost' port=5432 dbname='itucsdb'"""
connection = dbapi2.connect(dsn)

# database operations

connection.close()
```

Update Operations

- for update operations (insert, delete, update, create, drop, . . .)

- create a cursor on the connection
- execute statement(s) on the cursor
- commit pending changes on the connection
- close the cursor

Update Operation Example

```
connection = dbapi2.connect(dsn)
cursor = connection.cursor()
statement = """CREATE TABLE PERSON (
    ID SERIAL PRIMARY KEY,
    NAME VARCHAR(40) UNIQUE NOT NULL
)"""
cursor.execute(statement)
connection.commit()
cursor.close()
connection.close()
```

Retrieve Operations

- for retrieve operations (select)

- create a cursor on the connection
- execute statement on the cursor
- iterate over rows on the cursor (every row is a tuple)
- close the cursor

Retrieve Operation Example

```
connection = dbapi2.connect(dsn)
cursor = connection.cursor()
statement = """SELECT TITLE, SCORE FROM MOVIE
                WHERE (YR = 1999)"""
cursor.execute(statement)
for row in cursor:
    title, score = row
    print('{}: {}'.format(title, score))
cursor.close()
connection.close()
```

Retrieve Operation Examples

- simpler code with tuple assignment

```
for row in cursor:
    title, score = row
    print('{}: {}'.format(title, score))

for title, score in cursor:
    print('{}: {}'.format(title, score))
```

Retrieve Operation Examples

- movies and their directors

```
statement = """SELECT TITLE, NAME
                 FROM MOVIE JOIN PERSON
                   ON (MOVIE.DIRECTORID = PERSON.ID)"""
cursor.execute(statement)
for title, name in cursor:
    print('{}: {}'.format(title, name))
```

Error Handling

- catch database related exceptions
- rollback operation on error (`except`)
- close all opened resources (`finally`)

Template

```
try:
    connection = dbapi2.connect(dsn)
    cursor = connection.cursor()
    cursor.execute(statement)
    connection.commit()
    cursor.close()
except dbapi2.DatabaseError:
    connection.rollback()
finally:
    connection.close()
```

Connection Context Managers

- in some drivers, connections are context managers: `with`
- automatic commit (try), rollback (except), close (finally)
- template:

```
with dbapi2.connect(dsn) as connection:
    cursor = connection.cursor()
    cursor.execute(statement)
    cursor.close()
```

Connection Context Manager Example

```
with dbapi2.connect(dsn) as connection:
    cursor = connection.cursor()
    statement = """CREATE TABLE MOVIE (
        ID SERIAL PRIMARY KEY,
        TITLE VARCHAR(80),
        YR NUMERIC(4),
        SCORE FLOAT,
        VOTES INTEGER DEFAULT 0,
        DIRECTORID INTEGER REFERENCES PERSON (ID)
    )"""
    cursor.execute(statement)
    cursor.close()
```

Cursor Context Managers

- in some drivers, cursors are also context managers
- automatic close
- template:

```
with dbapi2.connect(dsn) as connection:
    with connection.cursor() as cursor:
        cursor.execute(statement)
```

Cursor Context Manager Example

```
with dbapi2.connect(dsn) as connection:
    with connection.cursor() as cursor:
        statement = """CREATE TABLE CASTING (
            MOVIEID INTEGER REFERENCES MOVIE (ID),
            ACTORID INTEGER REFERENCES PERSON (ID),
            ORD INTEGER,
            PRIMARY KEY (MOVIEID, ACTORID)
        )"""
        cursor.execute(statement)
```

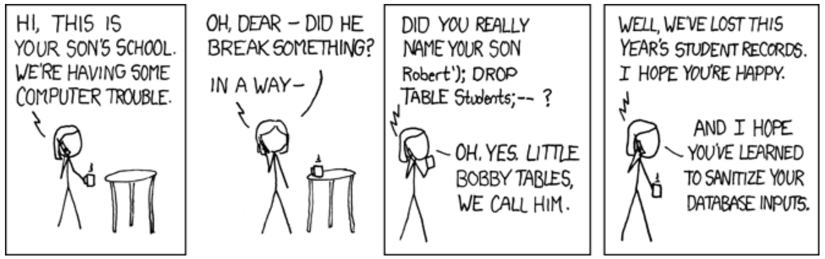
Statements

- unsafe to use string formatting for constructing statements
- never trust inputs from outside sources
- SQL injection attacks

bad example

```
name = input('What is your name? ')
statement = """INSERT INTO Students (Name)
                VALUES ('%s')""" % name
cursor.execute(statement)
```

SQL Injection Example


HI, THIS IS YOUR SON'S SCHOOL. WE'RE HAVING SOME COMPUTER TROUBLE.
OH, DEAR – DID HE BREAK SOMETHING?
IN A WAY–
DID YOU REALLY NAME YOUR SON Robert'); DROP TABLE Students;-- ?
OH, YES. LITTLE BOBBY TABLES, WE CALL HIM.
WELL, WE'VE LOST THIS YEAR'S STUDENT RECORDS. I HOPE YOU'RE HAPPY.
AND I HOPE YOU'VE LEARNED TO SANITIZE YOUR DATABASE INPUTS.

```
INSERT INTO Students (Name)
  VALUES ('Robert'); DROP TABLE Students;-- ')
INSERT INTO Students (Name)
  VALUES ('Robert'); DROP TABLE Students;-- ')
```

http://xkcd.com/327/

Placeholders

- placeholders for values
- different drivers use different formats: %s, ?, ...
- provide actual parameters as tuples or dictionaries

Placeholder Examples

- using tuples:

```
statement = """INSERT INTO MOVIE (TITLE, YR)
                 VALUES (%s, %s)"""
cursor.execute(statement, (title, year))
```

- using dictionaries:

```
statement = """INSERT INTO MOVIE (TITLE, YR)
                 VALUES (%(title)s, %(year)s)"""
cursor.execute(statement, {'year': year,
                           'title': title})
```

Fetching Results

- fetching results instead of iterating over cursor:
  `.fetchall()`
  `.fetchone()`

Fetch Example

- people and movies they directed

```
statement = """SELECT ID, NAME FROM PERSON"""
cursor.execute(statement)
people = cursor.fetchall()
for person_id, name in people:
    statement = """SELECT TITLE FROM MOVIE
                    WHERE (DIRECTORID = %s)"""
    cursor.execute(statement, (person_id,))
    directed = cursor.fetchall()
    print('{}:'.format(name))
    for (title,) in directed:
        print('  {}'.format(title))
```

References

Supplementary Reading

- Python Database API Specification v2.0:
  `https://www.python.org/dev/peps/pep-0249/`

Problem

- mismatch between data model and software model
- data is relational: relation, tuple, foreign key, ...
- software is object-oriented: object, reference, ...

Mismatch Example

- adding an actor to a movie: SQL definitions

```
CREATE TABLE MOVIE (ID INTEGER PRIMARY KEY,
    TITLE VARCHAR(80) NOT NULL)

CREATE TABLE PERSON (ID INTEGER PRIMARY KEY,
    NAME VARCHAR(40) NOT NULL)

CREATE TABLE CASTING (
    MOVIEID INTEGER REFERENCES MOVIE (ID),
    ACTORID INTEGER REFERENCES PERSON (ID),
    PRIMARY KEY (MOVIEID, ACTORID)
)
```

Mismatch Example

- adding an actor to a movie: SQL operations

```
INSERT INTO MOVIE (ID, TITLE)
  VALUES (110, 'Sleepy Hollow')

INSERT INTO PERSON (ID, NAME)
  VALUES (26, 'Johnny Depp')

INSERT INTO CASTING (MOVIEID, ACTORID)
  VALUES (110, 26)
```

Mismatch Example

- adding an actor to a movie: Python definitions

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

class Movie:
    def __init__(self, title):
        self.title = title
        self.cast = []

    def add_actor(self, person):
        self.cast.append(person)
```

Mismatch Example

- adding an actor to a movie: Python operations

```
movie = Movie('Sleepy Hollow')
actor = Person('Johnny Depp')
movie.add_actor(actor)
```

Object/Relational Mapping

- map software components to database components
- translate the object interface into SQL statements

model	SQL	software
relation	table	class
tuple	row	object (instance)
attribute	column	attribute

SQLAlchemy

- abstraction over SQL expressions
- object-relational mapper

- regular Python class
- SQL table description
- mapper maps class to table

Connection Example

```
from sqlalchemy import create_engine

uri = 'postgres://vagrant:vagrant@localhost:5432/itucsdb'
engine = create_engine(uri, echo=True)

from sqlalchemy import MetaData

metadata = MetaData()
```

Class Example

```
class Movie:
    def __init__(self, title, year=None,
                 score=None, votes=None):
        self.title = title
        self.yr = year
        self.score = score
        self.votes = votes
```

Table Example

```
from sqlalchemy import Column, Table
from sqlalchemy import Float, Integer, String

movie_table = Table(
    'Movie', metadata,
    Column('id', Integer, primary_key=True),
    Column('title', String(80), nullable=False),
    Column('yr', Integer),
    Column('score', Float)
    Column('votes', Integer)
)
```

Mapper Example

```
from sqlalchemy.orm import mapper

mapper(Movie, movie_table)
```

Creating Tables

```
metadata.create_all(bind=engine)
```

```
CREATE TABLE "Movie" (
    id SERIAL NOT NULL,
    title VARCHAR(80) NOT NULL,
    yr INTEGER,
    score FLOAT,
    votes INTEGER,
    PRIMARY KEY (id)
)
```

Sessions

- data operations are handled in sessions
- end with commit or rollback
- session keeps track of modified and new objects

Session Example

```python
from sqlalchemy.orm import sessionmaker

Session = sessionmaker(bind=engine)
session = Session()
```

Session Example: Insert

```python
movie = Movie('Casablanca', year=1942)
session.add(movie)
session.commit()
```

```sql
INSERT INTO "Movie" (title, yr, score, votes)
  VALUES (%(title)s, %(yr)s, %(score)s, %(votes)s)
  RETURNING "Movie".id

{'yr': 1942, 'title': 'Casablanca', 'score': None,
 'votes': None}

# autogenerated id is assumed to be 1
```

Session Example: Update

```python
movie.votes = 23283
session.commit()
```

```sql
UPDATE "Movie" SET votes=%(votes)s
 WHERE "Movie".id = %(Movie_id)s

{'Movie_id': 1, 'votes': 23283}
```

Session Example: Delete

```python
session.delete(movie)
session.commit()
```

```sql
DELETE FROM "Movie"
 WHERE "Movie".id = %(id)s

{'id': 1}
```

Query Examples

```
session.query(Movie)
```

```
SELECT "Movie".id AS "Movie_id",
       "Movie".title AS "Movie_title",
       "Movie".yr AS "Movie_yr",
       "Movie".score AS "Movie_score",
       "Movie".votes AS "Movie_votes"
  FROM "Movie"
```

Query Examples: Selecting Columns

```
session.query(Movie.title, Movie.score)
```

```
SELECT "Movie".title AS "Movie_title",
       "Movie".score AS "Movie_score"
  FROM "Movie"
```

SQLAlchemy Example: Ordering

```
session.query(Movie).order_by(Movie.yr)
```

```
SELECT "Movie".id AS "Movie_id",
       "Movie".title AS "Movie_title",
       "Movie".yr AS "Movie_yr",
       "Movie".score AS "Movie_score",
       "Movie".votes AS "Movie_votes"
  FROM "Movie"
  ORDER BY "Movie".yr
```

SQLAlchemy Example: Selecting Rows

```
session.query(Movie).filter_by(yr=1999)
```

```
SELECT "Movie".id AS "Movie_id",
       "Movie".title AS "Movie_title",
       "Movie".yr AS "Movie_yr",
       "Movie".score AS "Movie_score",
       "Movie".votes AS "Movie_votes"
  FROM "Movie"
 WHERE "Movie".yr = %(yr_1)s

{'yr_1': 1999}
```

SQLAlchemy Example: Selecting Rows by Predicate

```
session.query(Movie).filter(Movie.yr < 1999)
```

```
SELECT "Movie".id AS "Movie_id",
       "Movie".title AS "Movie_title",
       "Movie".yr AS "Movie_yr",
       "Movie".score AS "Movie_score",
       "Movie".votes AS "Movie_votes"
  FROM "Movie"
 WHERE "Movie".yr < %(yr_1)s

{'yr_1': 1999}
```

Foreign Keys

- add foreign key columns to table definitions
- add a “relationship” property to the mapper
- property name becomes attribute from source to target
- backref parameter becomes attribute from target to source

Foreign Key Example

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

person_table = Table(
    'Person', metadata,
    Column('id', Integer, primary_key=True),
    Column('name', String(40), nullable=False,
           unique=True)
)

mapper(Person, person_table)
```

Foreign Key Example

```
from sqlalchemy import ForeignKey

movie_table = Table(
    'Movie', metadata,
    Column('id', Integer, primary_key=True),
    Column('title', String(80)),
    Column('yr', Integer),
    Column('score', Float),
    Column('votes', Integer),
    Column('directorid', Integer, ForeignKey('Person.id'))
)
```

Foreign Key Example

```
from sqlalchemy.orm import relationship

mapper(Movie, movie_table,
       properties={
           'director':
               relationship(Person,
                            backref='directed')
       })
```

Foreign Key Example

```
movie = session.query(Movie) \
               .filter_by(title='Ed Wood').first()
```

```
SELECT "Movie".id AS "Movie_id",
       "Movie".title AS "Movie_title",
       ...
       "Movie".directorid AS "Movie_directorid"
  FROM "Movie"
 WHERE "Movie".title = %(title_1)s

{'title_1': 'Ed Wood'}

# returned directorid is assumed to be 8
```

Foreign Key Example

```
person = movie.director
print(person.name)
```

```
SELECT "Person".id AS "Person_id",
       "Person".name AS "Person_name"
  FROM "Person"
 WHERE "Person".id = %(param_1)s

{'param_1': 8}
```

Backref Example

```
for movie in person.directed:
    print(movie.title)
```

```
SELECT "Movie".id AS "Movie_id",
       "Movie".title AS "Movie_title",
       ...
       "Movie".directorid AS "Movie_directorid"
  FROM "Movie"
 WHERE %(param_1)s = "Movie".directorid

{'param_1': 8}
```

Foreign Key Example

- over a secondary table

```
casting_table = Table(
    'Casting', metadata,
    Column('movieid', Integer, ForeignKey('Movie.id'),
           primary_key=True),
    Column('actorid', Integer, ForeignKey('Person.id'),
           primary_key=True),
    Column('ord', Integer)
)
```

Foreign Key Example

```
mapper(Movie, movie_table,
       properties={
           'director':
               relationship(Person,
                            backref='directed'),
           'cast':
               relationship(Person,
                            backref='acted',
                            secondary=casting_table)
       })
```

Foreign Key Example

```
for movie in session.query(Movie):
    print('{}:'.format(movie.title))
    for person in movie.cast:
        print('  {}'.format(person.name))

for person in session.query(Person):
    print('{}:'.format(person.name))
    for movie in person.acted:
        print('  {}'.format(movie.title))
```

References

Supplementary Reading

- SQLAlchemy Documentation:
  `http://docs.sqlalchemy.org/`