

DB 응용프로그래밍

: mySQL 및 python 기반

DB 사용자 관리

- 사용자 생성

- MySQL 의 root 사용자를 모든 용도에 사용하지 말고, 적절한 권한을 가지는 사용자를 생성하여 사용하는 게 바람직

- 예)

`create user 'db2020'@'%' identified by 'db2020';`

Handwritten annotations:
- A blue circle around the '@' symbol with an arrow pointing to it and the text "만들 접근 가능" (Can create and access).
- A blue arrow pointing to the 'db2020' before the '@' with the label "userid".
- A blue arrow pointing to the 'db2020' after the '@' with the label "pw".

- 권한 부여

- 특정 데이터베이스, 테이블 등에 읽기/쓰기 등의 권한을 부여할 수 있음

- 예)

`grant all privileges on university.* to 'db2020'@'%';`

DB 응용프로그램 API 호출 순서

1. DB connection 설정 DBMS/DB
2. Cursor 생성
3. SQL 문 실행
4. SQL 검색결과 가져오기
5. Cursor 닫기
6. DB connection 닫기

A small db program

```
import pymysql
conn = pymysql.connect(host='localhost', user='db2020',
                        password='db2020', db='university')

curs = conn.cursor(pymysql.cursors.DictCursor)

sql = "select * from student"
curs.execute(sql)

row = curs.fetchone()
while ( row ):
    print(row)
    row = curs.fetchone()

curs.close()
conn.close()
```

DB connection 설정

```
#Pymysql를 import  
import pymysql
```

```
#DB connection 설정  
conn = pymysql.connect(  
    host='localhost',  
    user='db2020',  
    password='db2020 ' ,  
    db='university')
```

Cursor 생성

- **cursor**: 하나의 DB connection에 대하여 독립적으로 SQL 문을 실행할 수 있는 작업환경을 제공하는 객체.
 - 하나의 connection에 동시에 한개의 cursor만 생성할 수 있음
- cursor를 통해서 SQL 문을 실행할 수 있으며, 응용 프로그램이 실행결과를 튜플 단위로 접근할 수 있도록 해 줌.

```
curs = conn.cursor()
```

SQL 문 실행

#SQL 문 정의

```
sql = "select * from student"
```

#SQL 문 실행

```
curs.execute(sql)
```

SQL 검색결과 가져오기

- **방법 1:** 검색결과와 모든 tuple을 한번에 가져오기

```
rows = curs.fetchall()  
print(rows)
```

- fetchall()을 호출하면 mysql에서 생성한 검색결과와 모든 튜플을 응용프로그램 메모리 공간으로 모두 복사함.
- 검색결과와 사이즈가 너무 클 경우, 즉, 튜플이 과다하게 많을 경우, 메모리 복사 오버헤드 발생, 응용 프로그램 메모리 부족 등의 문제가 발생할 수 있음.

SQL 검색결과 가져오기

- **방법 2:** 검색결과의 tuple을 루프를 돌면서 하나씩 차례대로 가져오기

```
row = curs.fetchone()  
while row:  
    print(row)  
    row = curs.fetchone()
```

- fetchone()이 호출될때 마다, 해당 튜플만을 응용 프로그램 메모리 공간으로 가져오기 때문에, 응용 프로그램 메모리 관리가 효율적임.
- 검색결과의 튜플의 갯수가 현저히 작을때에만 fetchall()사용하고, 대부분의 경우에는 fetchone()을 사용하는 습관을 가지는 게 바람직함.

응용프로그램

mySQL

cursor

: SQL 실행 결과 보관

fetchone()

또는

fetchall()

⋮

The diagram illustrates the interaction between an application and a MySQL cursor. A vertical blue line separates the application side from the MySQL side. On the application side, the text 'fetchone()' is followed by '또는' (or) and 'fetchall()'. A vertical ellipsis '⋮' is positioned between these two. On the MySQL side, a cursor is defined as 'SQL 실행 결과 보관' (storage of SQL execution results). Below this is a table representing the result set, with 10 rows and 3 columns. The rows alternate between light gray and dark gray shading. A red curved line on the left side of the table indicates the current position of the cursor, starting at the top and moving down to the sixth row. Blue arrows point from the application side to the table: one from 'fetchone()' to the first row, one from '또는' to the second row, and one from 'fetchall()' to the sixth row. A vertical blue line separates the application side from the MySQL side.

* 응용프로그램 메모리관리는 fetchone()이 항상 유리

cursor 및 connection 닫기

#cursor 닫기

`curs.close()`

#db connection 닫기

`conn.close()`

A small db program

```
import pymysql
conn = pymysql.connect(host='localhost', user='db2020',
                       password='db2020', db='university')

curs = conn.cursor(pymysql.cursors.DictCursor)

sql = "select * from student"
curs.execute(sql)

row = curs.fetchone()
while ( row ):
    print(row)
    row = curs.fetchone()

curs.close()
conn.close()
```

SQL 검색결과에서 각 튜플의 애트리뷰트 리뷰트 접근하기

- cursor 생성시 `pymysql.cursors.DictCursor` 값을 설정해 주면 튜플들의 애트리뷰트를 이름으로 접근이 가능함

....

```
curs = conn.cursor(pymysql.cursors.DictCursor)
```

```
sql = "select sno, sname from student"  
curs.execute(sql)
```

```
row = curs.fetchone()  
while row:  
    print("학번:%d, 이름:%s" %(row['sno'], row['sname']))  
    row = curs.fetchone()
```

....

튜플 삽입

....

```
curs = conn.cursor()
```

```
sql = "insert into student (sno, sname, dept) values (%s, %s, %s)"
```

```
a = (3000, '김선경', '컴퓨터')
```

```
curs.execute(sql, a)
```

```
conn.commit()
```

....

- insert, delete, update 문 이후에는 connection 을 commit() 해야, 데이터베이스에 반영이 됨.

동시에 복수의 튜플 삽입

....

```
curs = conn.cursor()
```

```
sql = "insert into student (sno, sname, dept) values (%s, %s, %s)"
```

```
a = (3000, '김선경', '컴퓨터')
```

```
b = (4000, '황산성', '산업공학')
```

```
c = (5000, '김호일', '경영학')
```

```
student_list = [a, b, c]
```

```
curs.executemany(sql, student_list)
```

```
conn.commit()
```

....

튜플 삭제

....

```
curs = conn.cursor()
```

```
sno = 1000
```

```
sql = "delete from student where sno = %s" %(sno)
```

```
curs.execute(sql)
```

```
conn.commit()
```

....

튜플 업데이트

....

```
curs = conn.cursor()
```

```
dept = 'CS'
```

```
sno = 800
```

```
sql = "update student set dept = '%s' where sno = %d" %(dept,  
sno)
```

```
curs.execute(sql)
```

```
conn.commit()
```

....

동시에 여러개의 connection 및 cursor 생성하는 예

- 하나의 connection으로 동시에 하나의 cursor만을 생성하는 게 바람직함.
- 하나의 cursor는 한번에 하나의 SQL문을 실행할 수 있음.
- 동시에 여러 SQL문을 실행하고자 할때는 다중 connection 및 cursor를 생성할 수 있음.
- 한문장의 SQL로 표현하기 어렵거나 비효율적인 경우, 복수의 SQL문을 중첩으로 사용할 수 있음

2단계 SQL문

- 예

“컴퓨터과 학생들 중에서 2과목 이상을 수강하는 학생들의 기말고사 점수를 10점씩 올려라”

- 전략: 2단계의 중첩 SQL 문 수행

- 1단계: 2과목 이상 듣는 학생들의 학번을 검색한다

```
select s.sno  
from student s, enrol e  
where s.sno = e.sno and s.dept = '컴퓨터'  
group by e.sno  
having count(*) >=2
```

- 2단계: 각 학생에 대하여 기말고사 점수를 10점씩 올린다.

```
update enrol  
set final = final + 10  
where sno = %d
```

복수의 connection/cursor 예

```
import pymysql
```

```
conn1 = pymysql.connect(host='localhost', user='db201902',  
                        password='db2020', db='university')
```

```
curs1 = conn1.cursor(pymysql.cursors.DictCursor)
```

```
conn2 = pymysql.connect(host='localhost', user='db201902',  
                        password='db2020', db='university')
```

```
curs2 = conn2.cursor(pymysql.cursors.DictCursor)
```

```
sql1 = """ select s.sno  
          from student s, enrol e  
          where s.sno = e.sno and s.dept = '컴퓨터'  
          group by e.sno  
          having count(*) >=2"""
```

```
curs1.execute(sql1)
```

복수의 connection/cursor 예

```
row = curs1.fetchone()
```

```
while ( row ):
```

```
    sql2 = """ update enrol
```

```
        set final = final + 10
```

```
        where sno = %d""" %(row['sno'])
```

```
    cur2.execute(sql2)
```

```
    conn2.commit()
```

```
    row = curs1.fetchone()
```

```
curs1.close()
```

```
curs2.close()
```

```
conn1.close()
```

```
conn2.close()
```