SQL 활용 데이터 프레임 생성

> Python과 Oracle의 연동 코드

#01. 패키지 참조

`cx_oracle`와 `sqlalchemy` 패키지가 미리 설치되어 있어야 한다.

```
$ pip install cx_oracle
```

In [5]:
```
import cx_Oracle as cx                  # 대문자 주의
from pandas import DataFrame
from sqlalchemy import create_engine
from pandas import read_sql_table
```

#02. `cx_Oracle` 사용

1) 데이터베이스 접속

In [6]:
```
dbcon = cx.connect("hr",    # 사용자 이름
                   "hr",    # 비밀번호
                   "localhost:1521/xe"  # 데이터베이스 서버 주소
)
dbcon
```

Out[6]:
```
<cx_Oracle.Connection to hr@localhost:1521/xe>
```

2) 데이터 조회

기본 사용 방법

테이블의 각 record를 튜플로 표현하는 리스트 객체를 얻을 수 있다.

데이터 조회를 위한 커서 객체 생성

In [3]:
```
cursor = dbcon.cursor()
```

데이터 조회를 위한 SQL문 처리

In [4]:
```
sql = "SELECT * FROM department"
cursor.execute(sql)
result = cursor.fetchall()
result
```

Out[4]:
```
[(101, '컴퓨터공학과', '1호관'),
 (102, '멀티미디어학과', '2호관'),
 (201, '전자공학과', '3호관'),
 (202, '기계공학과', '4호관')]
```

딕셔너리 형태로 데이터 조회

cx_oracle은 별도의 딕셔너리 형태를 제공하지 않기 때문에 cursor객체의 rowfactory 프로퍼티를 직접 재정의 해야 한다.

```
cursor.rowfactory = lambda *args: dict(zip([d[0] for d in cursor.description], args))
```

> 코드 출처 : oracle.com

In [5]:
```
sql = "SELECT * FROM department ORDER BY deptno ASC"
cursor.execute(sql)

cursor.rowfactory = lambda *args: dict(zip([d[0] for d in cursor.description], args))
result = cursor.fetchall()
result
```

Out[5]:
```
[{'DEPTNO': 101, 'DNAME': '컴퓨터공학과', 'LOC': '1호관'},
 {'DEPTNO': 102, 'DNAME': '멀티미디어학과', 'LOC': '2호관'},
 {'DEPTNO': 201, 'DNAME': '전자공학과', 'LOC': '3호관'},
 {'DEPTNO': 202, 'DNAME': '기계공학과', 'LOC': '4호관'}]
```

조회결과를 데이터프레임으로 변환

In [6]:
```
df = DataFrame(result)
df
```

Out[6]:

	DEPTNO	DNAME	LOC
0	101	컴퓨터공학과	1호관
1	102	멀티미디어학과	2호관
2	201	전자공학과	3호관
3	202	기계공학과	4호관

데이터 프레임에 대한 인덱스 설정

In [7]:
```
df.set_index('DEPTNO', inplace=True)
df
```

Out[7]:

DEPTNO	DNAME	LOC
101	컴퓨터공학과	1호관
102	멀티미디어학과	2호관
201	전자공학과	3호관
202	기계공학과	4호관

3) 입력, 수정, 삭제

`INSERT`, `UPDATE`, `DELETE` 문의 수행 방식은 동일하다.

> 여기서는 데이터 조회 과정에서 생성한 `cursor` 객체를 재사용 한다.

데이터 입력

In [8]:
```
sql = "SELECT seq_department.nextval FROM dual"
cursor.execute(sql)
result = cursor.fetchall()
print(result)
```

```
seq = result[0][0]
print("새로운 시퀀스 번호: %d" % seq)
```

```
[(309,)]
새로운 시퀀스 번호: 309
```

In [9]:
```
sql = "INSERT INTO department (deptno, dname, loc) VALUES (:1, :2, :3)"
print(sql)

cursor.execute(sql, [seq, '컴퓨터공학과', '5호관'])
print("%s개의 행이 저장됨" % cursor.rowcount)

# 처리 결과를 실제로 반영함
dbcon.commit()

# 되돌리기
# --> 이미 commit()한 내역은 적용안됨
#dbcon.rollback()
```

```
INSERT INTO department (deptno, dname, loc) VALUES (:1, :2, :3)
1개의 행이 저장됨
```

데이터 수정

In [10]:
```
sql = "UPDATE department SET dname=:1, loc=:2 WHERE deptno=:3"
print(sql)

rows = cursor.execute(sql, ['컴퓨터과학과', '6호관', seq])
print("%s개의 행이 갱신됨" % cursor.rowcount)

dbcon.commit()
```

```
UPDATE department SET dname=:1, loc=:2 WHERE deptno=:3
1개의 행이 갱신됨
```

데이터 삭제

In [11]:
```
sql = "DELETE FROM department WHERE deptno > 202"
print(sql)

rows = cursor.execute(sql)
print("%s개의 행이 삭제됨" % cursor.rowcount)

dbcon.commit()
```

```
DELETE FROM department WHERE deptno > 202
1개의 행이 삭제됨
```

데이터베이스 접속 해제

In [12]:
```
cursor.close()
dbcon.close()
```

#02. SQLAlchemy 사용

1) 데이터베이스 접속

접속 문자열 생성

`oracle+cx_oracle://계정이름:비밀번호@접속주소/SID`

In [13]:
```
conStr = "oracle+cx_oracle://hr:hr@localhost:1521/xe"
```

데이터베이스 접속하기

In [14]:
```
engine = create_engine(conStr)
conn = engine.connect()
```

2) 데이터 조회하기

특정 테이블의 모든 데이터 조회

In [15]:
```
df = read_sql_table('department', con=conn)
df
```

Out[15]:

	deptno	dname	loc
0	101	컴퓨터공학과	1호관
1	102	멀티미디어학과	2호관
2	201	전자공학과	3호관
3	202	기계공학과	4호관

인덱스를 지정한 조회

`read_sql_table` 함수를 사용할 경우 WHERE절은 사용할 수 없다.

In [16]:
```
df = read_sql_table('department', index_col='deptno', con=conn)
df
```

Out[16]:

deptno	dname	loc
101	컴퓨터공학과	1호관
102	멀티미디어학과	2호관
201	전자공학과	3호관
202	기계공학과	4호관

특정 컬럼만 가져오기

In [17]:
```
df = read_sql_table('department', index_col='deptno', columns=['dname'], con=conn)
df
```

Out[17]:

deptno	dname
101	컴퓨터공학과
102	멀티미디어학과
201	전자공학과
202	기계공학과

3) 데이터 내보내기

- name='테이블명' 이름으로 기존 테이블이 있으면 해당 테이블의 컬럼명에 맞게 데이터를 넣을 수 있음

- if_exists='append' 옵션이 있으면, 기존 테이블에 데이터를 추가로 넣음
- if_exists='fail' 옵션이 있으면, 기존 테이블이 있을 경우, 아무일도 하지 않음
- if_exists='replace' 옵션이 있으면, 기존 테이블이 있을 경우, 기존 테이블을 삭제하고, 다시 테이블을 만들어서, 새로 데이터를 넣음

이미 만들어진 테이블이 없으면, name='테이블명' 이름으로 테이블을 자동으로 만들고, 데이터를 넣을 수 있음

테이블이 자동으로 만들어지므로, 테이블 구조가 최적화되지 않아 자동으로 테이블 만드는 것은 추천하지 않음

In [18]:
```
df.to_sql(name='new_table', con=conn, if_exists='append', index=False)
conn.commit()
```

4) 데이터베이스 접속 해제

DB 관련 작업이 종료되면 반드시 접속 객체를 반납해야 한다.

In [19]:
```
conn.close()
```