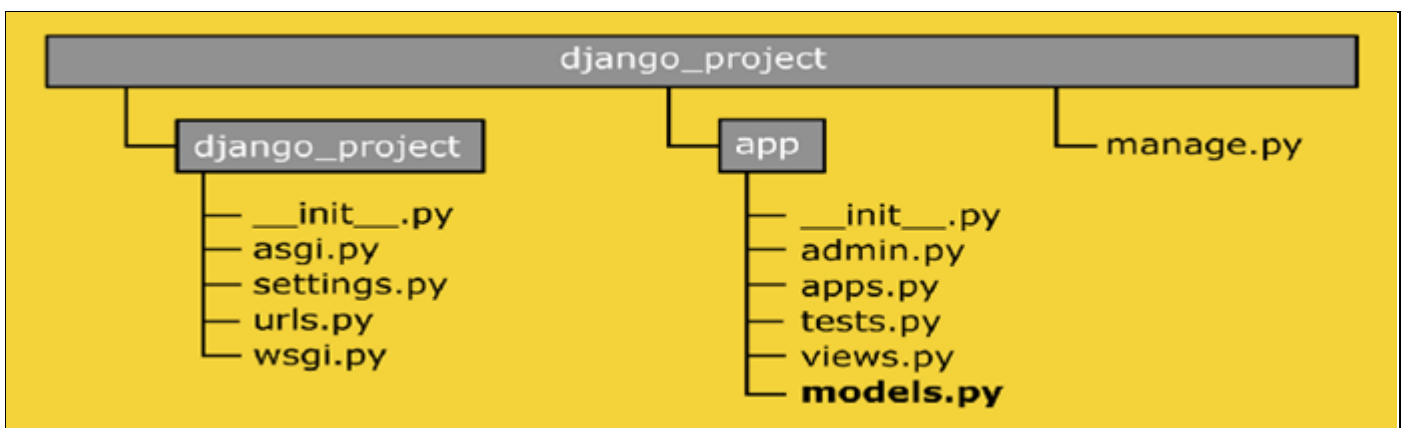
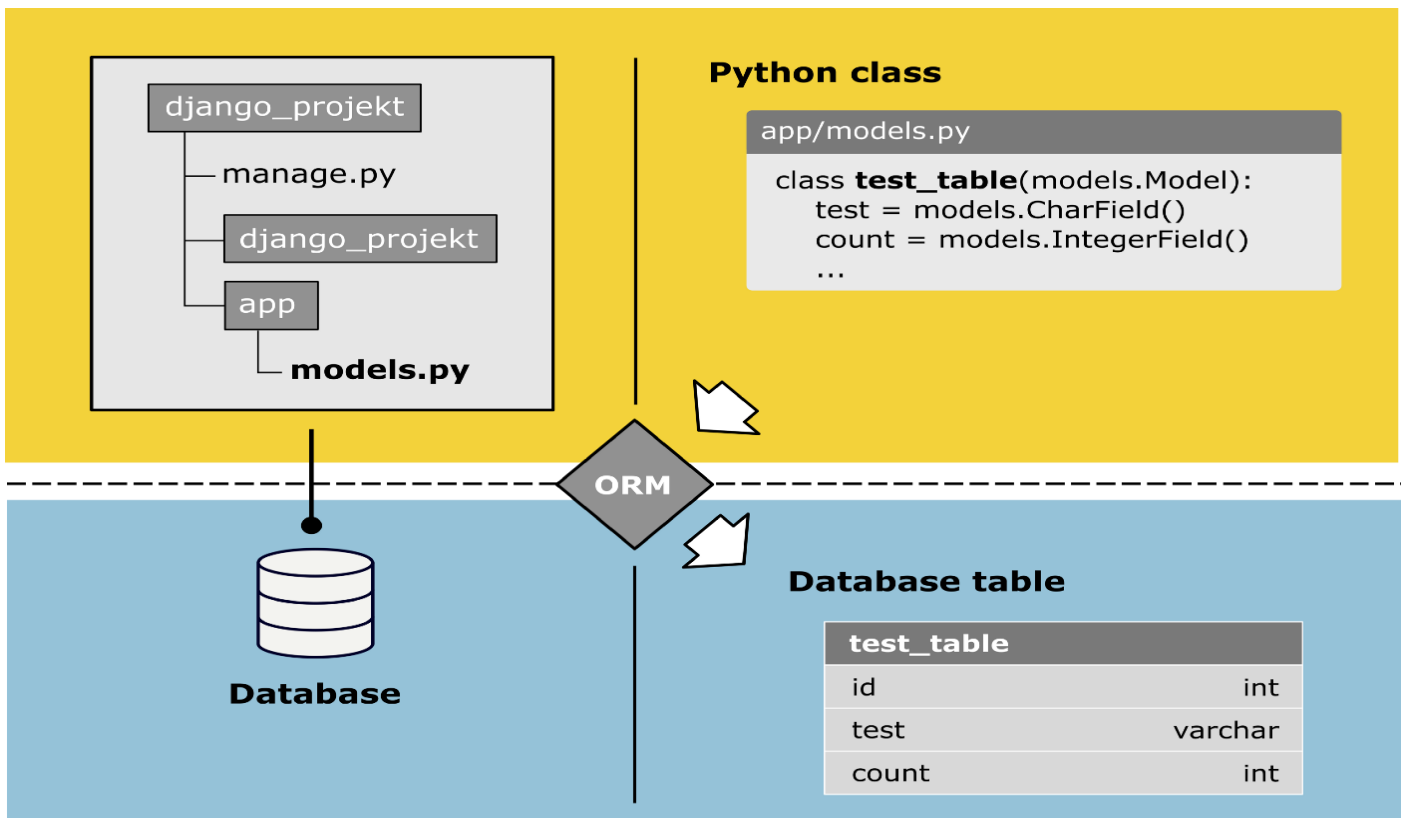
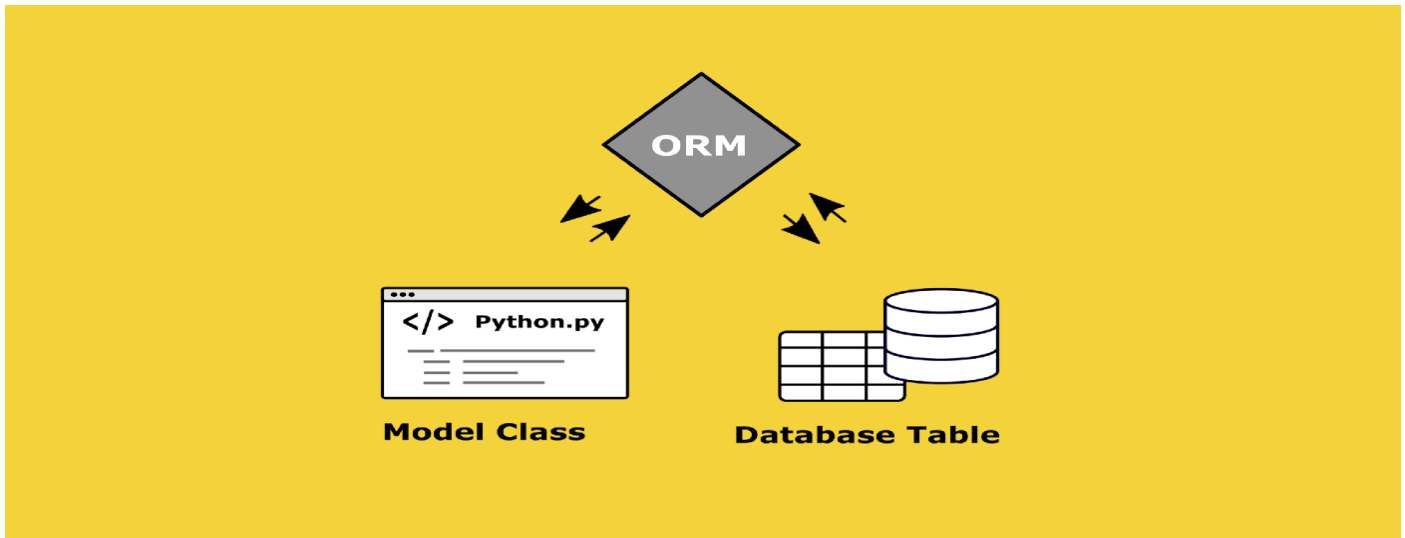
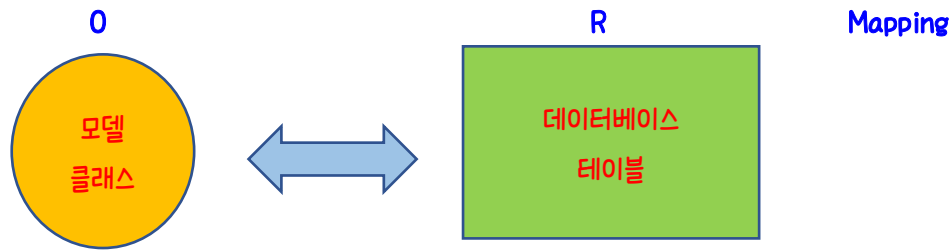


Django 모델 (Model)



Django에서 Model은 데이터 서비스를 제공한다. Django의 Model은 각 Django App안에 기본적으로 생성되는 models.py 모듈 안에 정의하게 된다. models.py 모듈 안에 하나 이상의 모델 클래스를 정의할 수 있으며, 하나의 모델 클래스는 데이터베이스에서 하나의 테이블에 해당된다.



Django 모델은 "`django.db.models.Model`"의 자식 클래스이며, 모델의 필드는 클래스의 속성(Attribute)로 표현되고 테이블의 컬럼에 해당한다. (Primary Key(기본키)가 지정되지 않으면, 모델에 Primary Key 역할을 하는 id 필드가 자동으로 추가되며 DB 테이블 생성시 자동으로 값이 1씩 증가되는 id 컬럼이 생성된다) Django 에서 필드는 모델을 생성할 때 필수적인 요소이며 필드를 clean, save, delete 등과 같이 모델 API 와 동일한 이름으로 생성하지 않도록 주의해야 한다.

```
class 모델이름(models.Model):
    필드이름1 = models.필드타입(필드옵션)
    필드이름2 = models.필드타입(필드옵션)
```

모델 클래스에 필드를 정의하기 위해 인스턴스 변수가 아닌 클래스 변수로 정의하며, 변수에는 테이블의 컬럼의 메타 데이터를 정의한다. 필드를 정의하는 각각의 클래스 변수는 `models.CharField()`, `models.IntegerField()`, `models.DateTimeField()`, `models.TextField()` 등의 각 필드 타입에 맞는 Field 클래스 객체를 생성하여 할당한다. Field 클래스는 여러 종류가 있는데, 생성자 함수 호출시 필요한 옵션들을 지정할 수 있다.

- 필드 타입

모델의 필드에는 다음과 같이 다양한 타입들이 있다. 모든 필드 타입 클래스들은 "Field" 클래스의 자손 클래스들이다.

Field Type	설명
CharField	제한된 문자열 필드 타입. 최대 길이를 <code>max_length</code> 옵션에 지정해야 한다. 문자열의 특별한 용도에 따라 CharField의 파생클래스로서, 이메일 주소를 체크를 하는 EmailField, IP 주소를 체크를 하는 GenericIPAddressField, 콤마로 정수를 분리한 CommaSeparatedIntegerField, 특정 폴더의 파일 패스를 표현하는 FilePathField, URL을 표현하는 URLField 등이 있다.
TextField	대용량 문자열을 갖는 필드
IntegerField	32 비트 정수형 필드. 정수 사이즈에 따라 BigIntegerField, SmallIntegerField 을 사용할 수도 있다.
BooleanField	true/false 필드. Null 을 허용하기 위해서는 NullBooleanField를 사용한다.
DateTimeField	날짜와 시간을 갖는 필드. 날짜만 가질 경우는 DateField, 시간만 가질 경우는 TimeField를 사용한다. <code>auto_now_add</code> (생성)과 <code>auto_now</code> (수정)을 true로 설정하면 생성 또는 수정시 기본 타임존 시간으로 변경된다.
DecimalField	소수점을 갖는 decimal 필드

BinaryField	바이너리 데이터를 저장하는 필드
FileField	파일 업로드 필드
ImageField	FileField의 파생클래스로서 이미지 파일인지 체크한다.

위와 같은 필드 타입 클래스 이외에, Django 프레임워크는 테이블 간 혹은 필드 간 **관계(Relationship)**를 표현하기 위해 **ForeignKey, ManyToManyField, OneToOneField** 클래스를 또한 제공하고 있다. 특히 **ForeignKey**는 모델 클래스간의 Many-To-One (혹은 One-To-Many) 관계를 표현하기 위해 흔히 사용된다.

- 필드 옵션

모델의 필드는 필드 타입에 따라 여러 옵션(혹은 Argument)를 가질 수 있다. 예를 들어, CharField는 문자열 최대 길이를 의미하는 max_length 라는 옵션을 갖는다. 필드 옵션은 일반적으로 생성자에서 아규먼트로 지정한다. 다음은 모든 필드 타입에 적용 가능한 옵션들 중 자주 사용되는 몇 가지를 요약한 것이다.

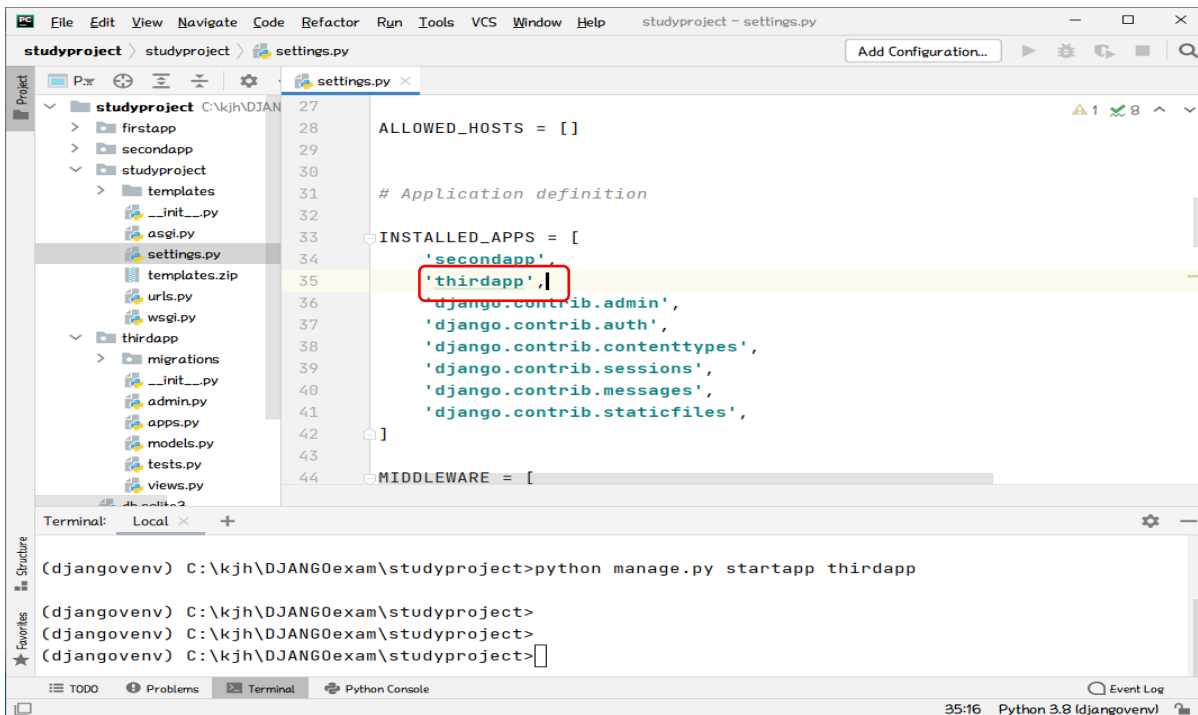
필드 옵션	설명
null (Field.null)	null=True 이면, Empty 값을 DB에 NULL로 저장한다. DB 테이블에서 Null이 허용되는 필드가 된다. 예) models.IntegerField(null=True)
blank (Field.blank)	blank=False 이면, Required 필드가 된다. blank=True 이면, Optional 필드이다. 예) models.DateTimeField(blank=True)
primary_key (Field.primary_key)	해당 필드가 Primary Key임을 표시한다. 예: models.CharField(max_length=10, primary_key=True) 모델 클래스에 Primary Key 로 설정되는 필드가 없으면 id 라는 필드가 자동 생성된다.
unique (Field.unique)	해당 필드가 테이블에서 Unique함을 표시한다. 해당 컬럼에 대해 Unique Index를 생성한다. 예) models.IntegerField(unique=True)
default (Field.default)	필드의 디폴트값을 지정한다. 예: models.CharField(max_length=2, default="WA")

- DB Migration

Django에서 Model 클래스를 생성하고 난 후, 해당 모델에 상응하는 DB 테이블을 데이터베이스에서 생성할 수 있다. Python 모델 클래스의 수정 (및 생성)을 DB에 적용하는 과정을 **Migration**이라 부른다. 이는 Django가 기본적으로 제공하는 ORM(Object-Relational Mapping) 서비스를 통해 진행된다.

Django 모델 클래스로부터 테이블을 생성하기 위해서는 크게 Migration을 준비하는 과정과 이를 적용하는 과정으로 나뉘는데, 구체적으로는 다음과 같은 절차를 따른다.(DB Migration을 수행하려면 해당 앱이 settings.py 파일 안의 **INSTALLED_APPS** 리스트에 등록되어 있어야 한다.)

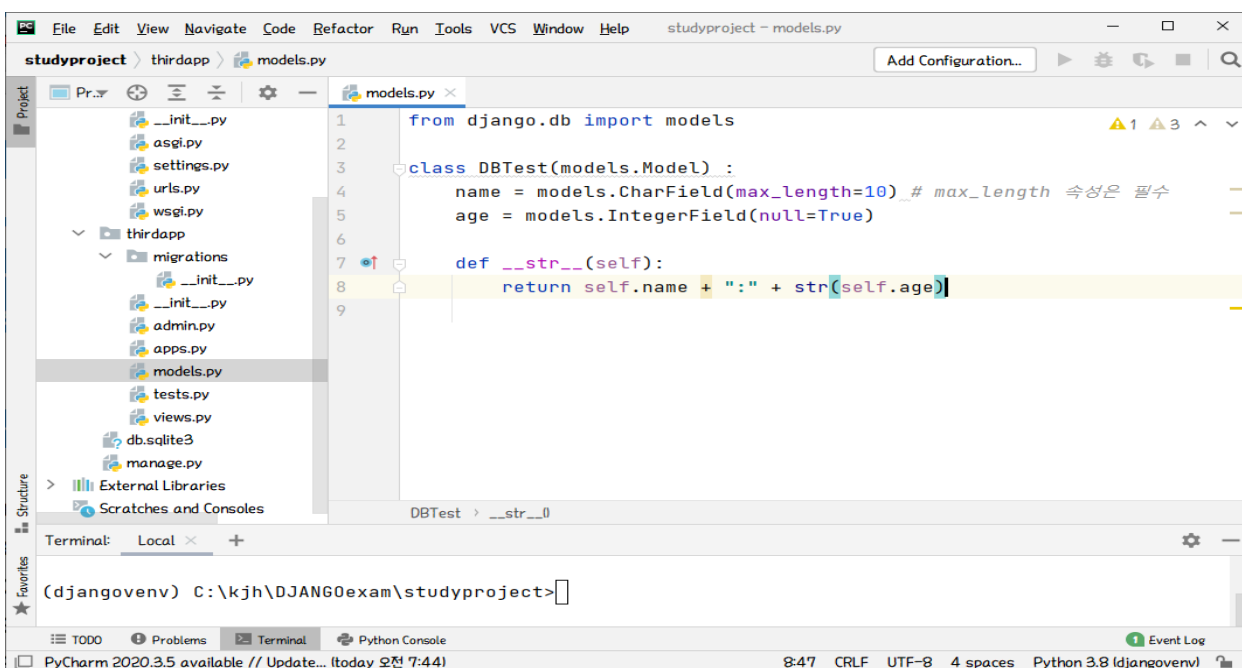
thirdapp을 생성(**python manage.py startapp thirdapp**)하고 thirdapp을 등록한다. (반드시 생성하고 나서 등록한다.)



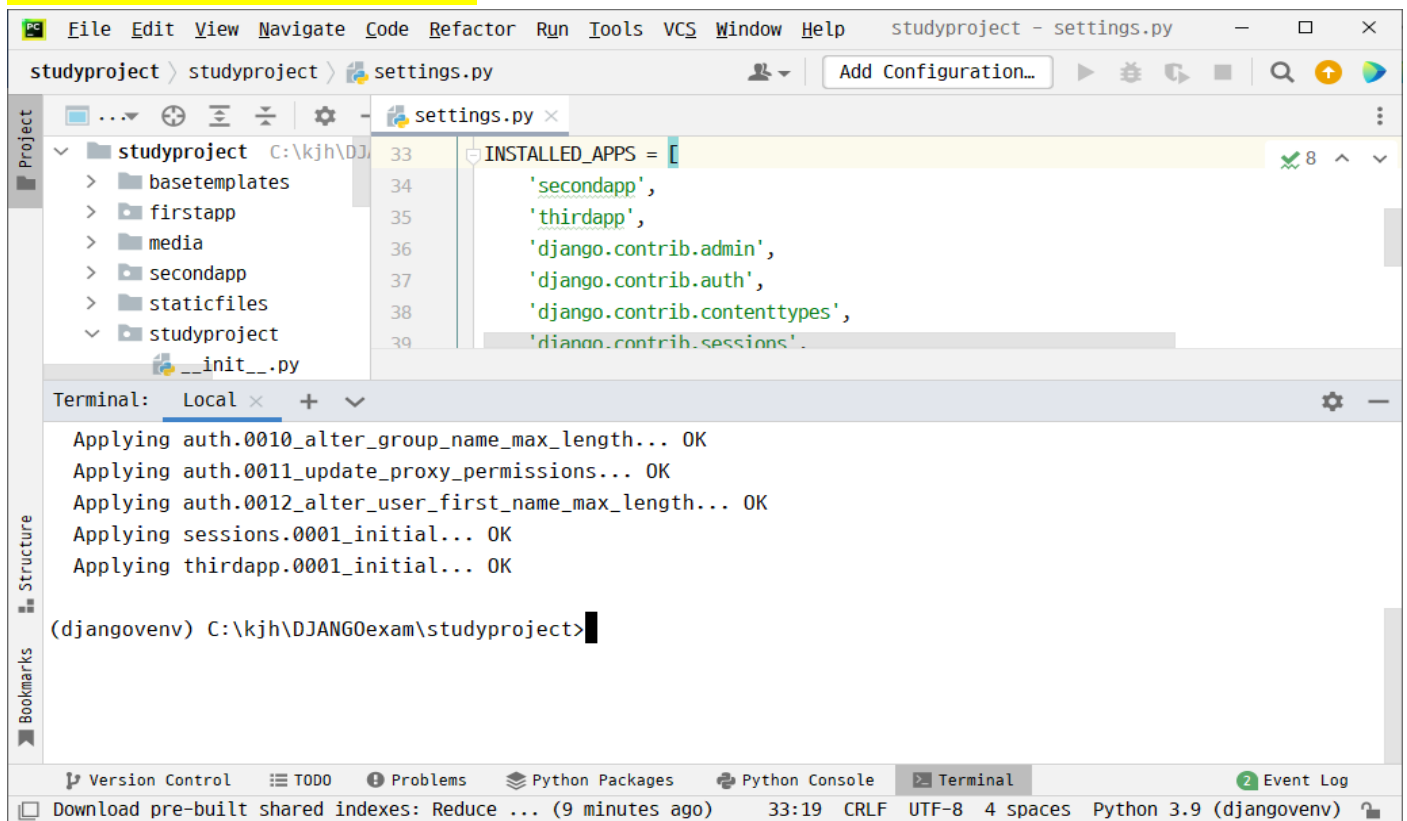
thirdapp 폴더의 models.py 를 오픈하여 다음 내용을 작성한다.

```
from django.db import models

class DBTest(models.Model):
    name = models.CharField(max_length=10) # max_length 속성은 필수
    age = models.IntegerField(null=True)
    def __str__(self):
        return self.name + ":" + str(self.age)
```



```
python manage.py makemigrations thirdapp
```



- Django 모델 API

앞의 Django 모델에서처럼 모델 클래스를 정의하게 되면, Django는 데이터를 추가/갱신하고 읽어 들일 수 있는 다양한 데이터베이스 API 들을 자동으로 제공한다. 이러한 기능은 Django가 ORM 서비스를 기본적으로 제공함에 따른 것으로 데이터베이스를 편리하게 핸들링할 수 있게 도와준다.

[INSERT]

데이터를 삽입하기 위해서는 먼저 테이블에 해당하는 모델(Model Class)로부터 객체를 생성하고, 그 객체의 `save()` 메서드를 호출한다. `save()` 메서드가 호출되면, SQL의 INSERT이 생성되고 실행되어 테이블에 데이터가 추가된다.

[SELECT]

Django는 디폴트로 모든 Django 모델 클래스에 대해 `"objects"` 라는 `Manager (django.db.models.Manager)` 객체를 자동으로 추가한다. Django 에서 제공하는 이 `Manager`를 통해 특정 데이터를 필터링 할 수도 있고 정렬할 수도 있으며 기타 여러 기능들을 사용할 수 있다. 데이터를 읽어오기 위해서는 Django 모델의 `Manager` 즉 `"모델클래스.objects"` 를 사용한다.

Django Model API에는 기본적으로 제공하는 여러 쿼리 메서드들이 있는데, 자주 사용되는 주요 메서드 몇 가지만 살펴본다.

<code>all()</code>	테이블 데이터를 전부 가져오기 위해서는 <code>모델클래스명.objects.all()</code> 를 사용한다. <code>QuerySet</code> 객체가 리턴된다.
<code>get()</code>	하나의 행(Row)을 가져오기 위해서는 <code>get()</code> 메서드를 사용하며 가져오려는 행의 pk 값 또는 id 값을 아규먼트로 지정한다.
<code>filter()</code>	특정 조건에 맞는 행(Row)들을 가져오기 위해서는 <code>filter()</code> 메서드를 사용한다. <code>rows = 모델클래스명.objects.filter(name='유니코')</code>
<code>exclude()</code>	특정 조건을 제외한 나머지 Row들을 가져오기 위해서는 <code>exclude()</code> 메서드를 사용한다. <code>rows = 모델클래스명.objects.exclude(name='유니코')</code>
<code>count()</code>	데이터의 갯수(row 수)를 알려면 <code>count()</code> 메서드를 사용한다.
<code>order_by()</code>	데이터를 키에 따라 정렬하기 위해 <code>order_by()</code> 메서드를 사용한다. <code>order_by()</code> 안에는 정렬 키를 나열할 수 있는데, 앞에 <code>-</code> 가 붙으면 내림차순이다. 다음 예는 id를 기준으로 올림차순, <code>createDate</code> 로 내림차순으로 정렬하게 된다. <code>rows = 모델클래스명.objects.order_by('id', '-writedata')</code>
<code>distinct()</code>	중복된 값은 하나로만 표시하기 위해 <code>distinct()</code> 메서드를 사용한다. SQL의 <code>SELECT DISTINCT</code> 와 같은 효과를 낸다. 아래는 <code>name</code> 필드가 중복되는 경우 한 번만 표시하게 된다. <code>rows = 모델클래스명.objects.distinct('name')</code>
<code>first()</code>	데이터들 중 처음에 있는 row만을 리턴한다. 아래는 <code>name</code> 필드로 정렬했을 때 첫 번째 row를 리턴한다. <code>row = 모델클래스명.objects.order_by('name').first()</code>
<code>last()</code>	데이터들 중 마지막에 있는 row만을 리턴한다.

위의 쿼리 메서드들은 실제 데이터 결과를 직접 리턴하는 것이 아니라 `쿼리 표현식(Django에서 QuerySet이라 한다)`을 리턴하는데, 여러 메서드들을 체인처럼 연결하여 사용할 수 있다. 여러 체인으로 연결되면 최종적으로 리턴된 쿼리가 해석되어 DB에는 실제 하나의 쿼리를 보내게 된다. 다음은 여러 메서드들을 사용하여 체인으로 연결한 예제이다.

```
row = 모델클래스명.objects.filter(name='유니코').order_by('-writedate').first()
```

[UPDATE]

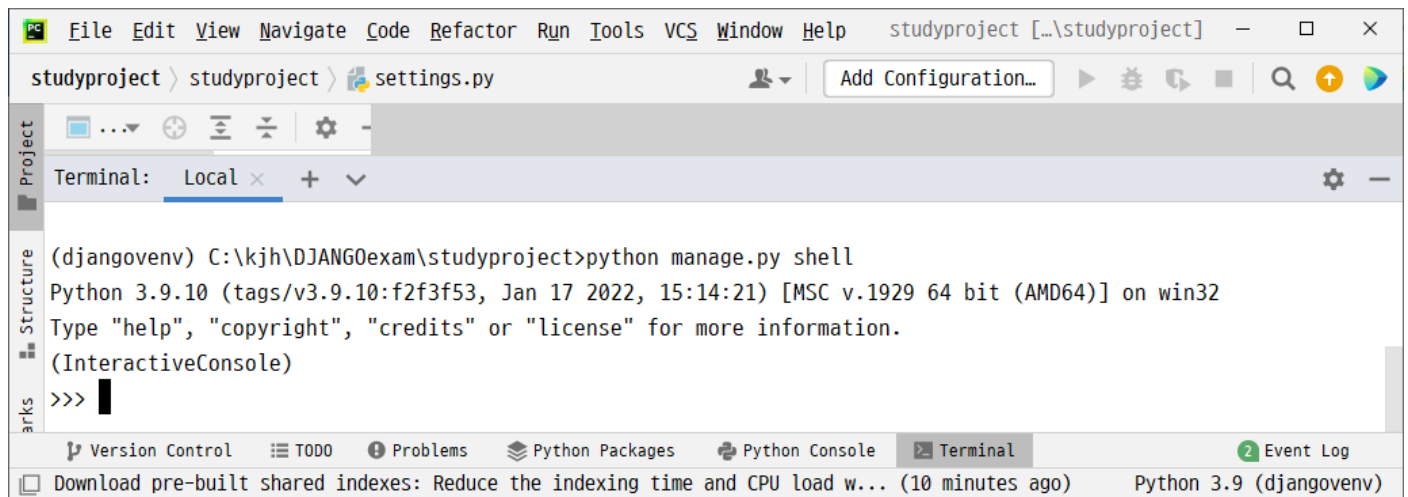
데이터를 수정하기 위해서는 먼저 수정할 행(Row)객체를 얻은 후 변경할 필드들을 수정한다. 마지막에 `save()` 메서드를 호출되면, SQL의 UPDATE 명령이 실행되어 테이블의 데이터가 갱신된다. 아래는 id가 1인 Feedback 객체에 이름을 변경하는 코드이다.

[DELETE]

데이터를 삭제하기 위해서는 먼저 삭제할 행(Row)객체를 얻은 후 `delete()` 메서드를 호출한다.

- Django 모델 API를 테스트 해보기

터미널에서 `python manage.py shell` 을 실행시키고 인터랙티브 파이썬 실행 모드를 기동시킨다.



```
studyproject > studyproject > settings.py
Terminal: Local x + v
(djangoven) C:\kjh\DJANGOexam\studyproject>python manage.py shell
Python 3.9.10 (tags/v3.9.10:f2f3f53, Jan 17 2022, 15:14:21) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
(InteractiveConsole)
>>>
```

```
>>> from thirdapp.models import DBTest
>>> DBTest.objects.all()
<QuerySet []>
```

```
>>> DBTest(name='유니코', age=10).save()
>>> DBTest(name='둘리', age=11).save()
>>> DBTest(name='또치', age=12).save()
>>> DBTest('도우너', 10).save()
Traceback (most recent call last):
  File "C:\kjh\python_venv\djangoven\lib\site-packages\django\db\models\fields\_init_.py", line 2018, in
    return int(value)
ValueError: invalid literal for int() with base 10: '도우너'
```

```
>>> DBTest(name='도우너', age=10).save()
>>> DBTest.objects.all()
<QuerySet [<DBTest: 유니코:10>, <DBTest: 둘리:11>, <DBTest: 또치:12>, <DBTest: 도우너:10>]>
```



```

>>> DBTest.objects.get(id=1)
<DBTest: 유니코:10>
>>> DBTest.objects.get(id=2)
<DBTest: 둘리:11>
>>> DBTest.objects.get(id=3)
<DBTest: 또치:12>
>>> DBTest.objects.get(id=4)
<DBTest: 도우너:10>

>>> DBTest.objects.get(id=5)
Traceback (most recent call last):
  File "<console>", line 1, in <module>
  File "C:\kjh\python_venv\djangoenv\lib\site-packages\django\db\models\manager.py", line 85, in manager_method
    return getattr(self.get_queryset(), name)(*args, **kwargs)
  File "C:\kjh\python_venv\djangoenv\lib\site-packages\django\db\models\query.py", line 650, in get
    raise self.model.DoesNotExist(
thirdapp.models.DBTest.DoesNotExist: DBTest matching query does not exist.

>>> DBTest.objects.count()
4
>>> DBTest.objects.first()
<DBTest: 유니코:10>
>>> DBTest.objects.last()
<DBTest: 도우너:10>
>>> DBTest.objects.order_by('age')
<QuerySet [<DBTest: 유니코:10>, <DBTest: 도우너:10>, <DBTest: 둘리:11>, <DBTest: 또치:12>]>
>>> DBTest.objects.order_by('-age')
<QuerySet [<DBTest: 또치:12>, <DBTest: 둘리:11>, <DBTest: 유니코:10>, <DBTest: 도우너:10>]>

>>> DBTest.objects.exclude(age=10)
<QuerySet [<DBTest: 둘리:11>, <DBTest: 또치:12>]>
>>> DBTest.objects.exclude(age=11)
<QuerySet [<DBTest: 유니코:10>, <DBTest: 또치:12>, <DBTest: 도우너:10>]>

>>> row = DBTest.objects.get(id=1)
>>> row.name
'유니코'
>>> row.age
10
>>> row.name = 'unico'
>>> row.save()
>>> DBTest.objects.all()
<QuerySet [<DBTest: unico:10>, <DBTest: 둘리:11>, <DBTest: 또치:12>, <DBTest: 도우너:10>]>
>>> row.name
'unico'
>>> row.delete()
(1, {'thirdapp.DBTest': 1})
>>> DBTest.objects.all()
<QuerySet [<DBTest: 둘리:11>, <DBTest: 또치:12>, <DBTest: 도우너:10>]>

```



```

>>> DBTest.objects.filter(age__gt = 10)
<QuerySet [<DBTest: 돌리:11>, <DBTest: 또치:12>]>
>>> DBTest.objects.filter(age__gte = 10)
<QuerySet [<DBTest: 돌리:11>, <DBTest: 또치:12>, <DBTest: 도우너:10>]>
>>> DBTest.objects.filter(name__exact = '돌리')
<QuerySet [<DBTest: 돌리:11>]>
>>> DBTest.objects.filter(name__contains = '너')
<QuerySet [<DBTest: 도우너:10>]>
>>> DBTest.objects.filter(name__startswith = '또')
<QuerySet [<DBTest: 또치:12>]>


>>> type(DBTest.objects.filter(age__gte = 10))
<class 'django.db.models.query.QuerySet'>
>>> type(DBTest.objects.all())
<class 'django.db.models.query.QuerySet'>
>>> type(DBTest.objects.get(id=2))
<class 'thirdapp.models.DBTest'>


>>> DBTest.objects.all().values()
<QuerySet [{ 'id': 2, 'name': '돌리', 'age': 11}, { 'id': 3, 'name': '또치', 'age': 12}, { 'id': 4, 'name': '도우너', 'age': 10}]>


>>> DBTest.objects.values_list('name')
<QuerySet [( '돌리',), ( '또치',), ( '도우너',)]>
>>> DBTest.objects.values_list('name', flat=True)
<QuerySet [ '돌리', '또치', '도우너']>
>>>
>>> DBTest.objects.values_list('name').order_by('name')
<QuerySet [( '도우너',), ( '돌리',), ( '또치',)]>
>>> DBTest.objects.values_list('name', flat=True).order_by('name')
<QuerySet [ '도우너', '돌리', '또치']>


>>> DBTest.objects.values_list('name').order_by('name')
<QuerySet [( '도우너',), ( '돌리',), ( '또치',)]>
>>> DBTest.objects.values_list('name', flat=True).order_by('name')
<QuerySet [ '도우너', '돌리', '또치']>


>>> DBTest.objects.all()
<QuerySet [<DBTest: 돌리:11>, <DBTest: 또치:12>, <DBTest: 도우너:10>]>
>>> DBTest.objects.all()[:2]
<QuerySet [<DBTest: 돌리:11>, <DBTest: 또치:12>]>


>>> type(DBTest.objects)
<class 'django.db.models.manager.Manager'>

```

※ 필드 룩업 (Field lookups)

(<https://docs.djangoproject.com/en/4.0/topics/db/queries/>에서 소개된 주요 내용 정리)

여기서 Entry, Blog 등은 이 사이트에서 소개하고 있는 내용의 모델 클래스들이다. 자세한 것은 위의 URL 사이트를 참고한다.

필드 룩업 (Field lookups) 은 SQL의 WHERE 절(추출조건)에 해당하는 부분으로 QuerySet 객체의 메서드인 `filter()`, `exclude()`, `get()`에 키워드 인자 형태로 전달된다. 필드 룩업의 기본적인 형태는 다음과 같다.

필드이름__룩업타입=조건값

예를 들면, 2006년 이전에 발행된 Entry 객체들의 쿼리셋을 가져오려면 다음과 같이 할 수 있다.

`Entry.objects.filter(pub_date__lte='2006-01-01')` # lte: less than or equal to

```
SELECT *
FROM blog_entry
WHERE pub_date <= '2006-01-01';
```

룩업에 들어가는 필드이름은 검색하려는 모델의 필드이름이며, 예외적으로 ForeignKey 필드의 경우에는 필드이름에 `_id` 가 들어간다. 외래키 필드를 검색하는 경우에는 조건값으로 외래키 필드가 참조하고 있는 레코드의 기본키 필드 값을 전달해주어야 한다.

예를 들어, 다음은 Blog 테이블의 기본키 값이 4인 레코드를 참조하고 있는 모든 Entry 테이블의 레코드들의 쿼리셋을 반환한다.

`Entry.objects.filter(blog_id=4)`

잘못된 룩업 타입을 전달한 경우, `TypeError` 가 발생한다.

[자주 사용하는 룩업 타입]

exact: 완전히 일치하는 값 검색.

`Entry.objects.get(headline__exact="Cat bites dog")`

위 명령은 Entry 테이블에서 headline 변수가 정확히 "Cat bites dong" 인 레코드를 가져온다.

필드 룩업을 사용할 때, 룩업 타입, 즉, `__` 와 그 이후 부분을 지정하지 않으면, 자동으로 `exact` 를 사용한 매치를 실행한다.

`Blog.objects.get(id__exact=4)`

`Blog.objects.get(id=4)`

`Blog.objects.get(name__exact=None)`

icontains: 대소문자 구분을 하지 않는 exact.

`Blog.objects.get(name__icontains="beatles blog")`

Beatles blog, beatles Blog, BEAtles BlOg 등등이 모두 매치됨.

contains: 조건값을 포함한 값과 매치됨. 대소문자 구분함.

`Entry.objects.get(headline__contains='Lennon')`

위 명령은 아래 SQL 문과 같다.

```
SELECT *
FROM Entry
WHERE headline
LIKE '%Lennon%';
```

icontains: contains 의 대소문자 무시 버전.

startswith, endswith: 각각 조건값으로 시작, 끝나면 매치된다. 대소문자 구분함.

istartswith, iendswith: startswith, endswith 의 대소문자 무시 버전.

[다양한 필드 룩업의 예]

```
e = Entry.objects.filter(pub_date__year=2006)
```

```
e = Entry.objects.exclude(pub_date__year=2006)
```

```
e = Entry.objects.filter(headline__startswith='What')
```

```
.exclude(pub_date__gte=datetime.date.today())
```

```
.filter(pub_date__gte=datetime.date(2005, 1, 5))
```

```
e1 = Entry.objects.filter(headline__startswith='What')
```

```
e2 = e1.exclude(pub_date__gte=datetime.date.today())
```

```
e3 = e1.filter(pub_date__gte=datetime.date(2005, 1, 5))
```

```
one_entry = Entry.objects.get(pk=1)
```

get() 을 이용할 경우 매치되는 값이 없으면 **모델클래스.DoesNotExist** 가 발생하게 된다. **get()** 에 매치된 결과가 여러 개일 경우에도 에러가 발생한다. 이 경우 발생하는 에러는 **모델클래스.MultipleObjectsReturned** 이다.

[쿼리 제한하기]

Python의 슬라이싱 문법을 통해서 쿼리셋에 들어갈 데이터를 제한할 수 있다. SQL의 LIMIT 과 OFFSET 문과 같은 역할을 한다.

(음수 슬라이싱은 지원하지 않는다.)

```
Entry.objects.all()[:5]
```

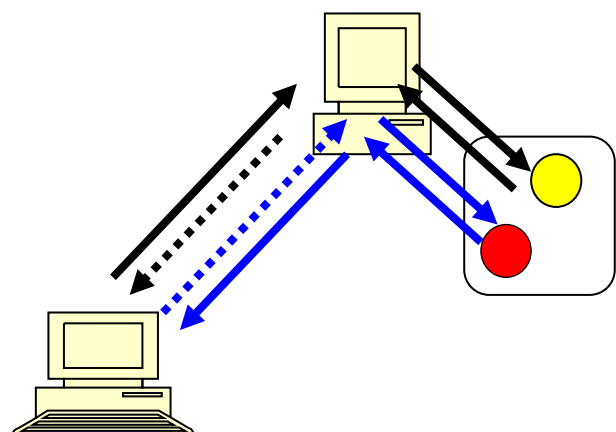
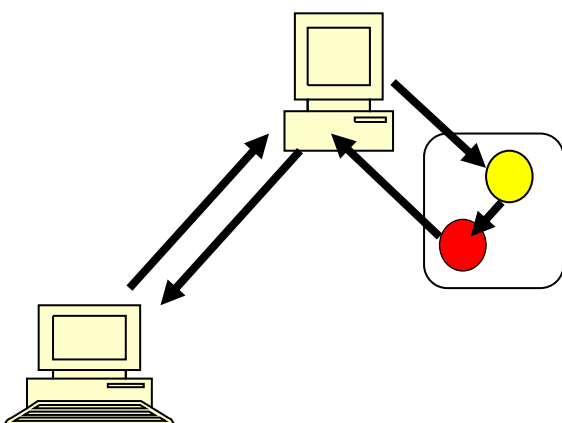
```
Entry.objects.all()[5:10]
```

```
Entry.objects.all()[10:20]
```

하나의 값을 가져오려면 인덱싱을 해준다. 가져오려는 인덱스의 값이 없을 경우 **IndexError** 를 일으킨다.

```
Entry.objects.order_by('headline')[0]
```

[redirect 와 render]



ORM(Object-Relation Mapping)

ORM

- oop 프로그래밍에서 RDBMS 를 연동할 때, 데이터베이스와 OOP 프로그래밍 언어간의 호환되지 않는 데이터를 변환하는 프로그래밍 기법

장점

- SQL 문을 몰라도 DB 연동이 가능하다.
- SQL 의 절차적인 접근이 아닌 객체 지향적인 접근으로 인해 생산성이 증가한다.

단점

- ORM 만으로 완전한 서비스를 구현하는데에는 어렵다.

CRUD

Objects

- models.py에 작성한 클래스를 불러와서 사용할 때 DB와의 인터페이스 역할을 하는 manager.
- Objects manager 라고 부름
- python class(python)Objects(인터페이스 역할).....DB(SQL)

```
Article.objects.filter(title='first', content='djago!')

# 내림 차순
Article.objects.order_by('-pk')

# LIKE
Article.objects.filter(title__contain='fi')
Article.objects.filter(title__startswith='fi')
Article.objects.filter(content__endswith='!')
```

QuerySet

- Object 매니저를 사용하여 복수의 데이터를 가져오는 함수를 사용할 때 반환되는 객체 타입
- 단일 객체는 Query(class 의 인스턴스로 반환)
- query(질문)을 DB에게 보내서 글을 조회하거나 생성, 수정, 삭제.
- query 를 보내는 언어를 활용해서 DB에게 데이터에 대한 조작을 실행.

CREATE

```
#1
article = Article()
article.title = 'first' # 인스턴스에 값을 넣어준다
article.content = 'django!!'
article.save()

#2
article = Article(title='second', content='django!!')
article.save()

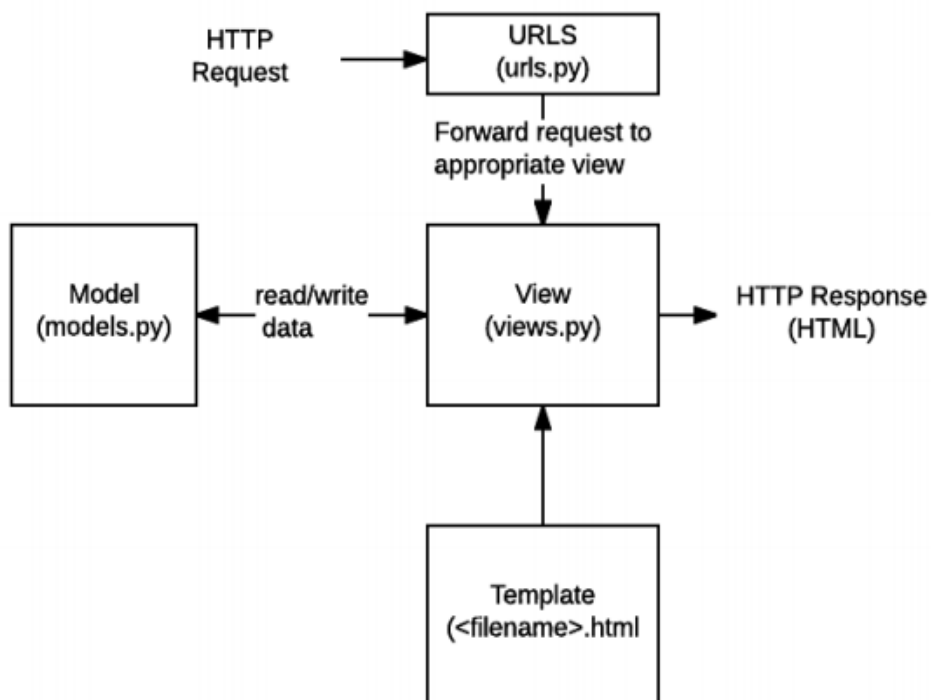
#3
Article.objects.create(title='third', content='django!!')
#save 안해도 등록 됨
```

Update

```
article = Article.objects.get(pk=1)
article.title = 'edit title'
article.save()
```

Delete

```
article = Article.objects.get(pk=1)
article.delete()
```



[리스트 페이지징]

Django에서는 리스트 페이지징을 편하게 구현할 수 있게 Paginator 라는 클래스를 제공한다. 리스트 출력시에 페이지징을 원하는 views에서는 데이터들을 추출하고 나서 Paginator 클래스의 객체를 생성한 다음 생성시 설정된 데이터 갯수로 구성된 Page 객체를 추출하여 템플릿에게 전달한다.

```
from django.core.paginator import Paginator
```

```
vlist = Visitor.objects.all()
```

```
paginator = Paginator(vlist, 3)
```

```
vlistpage = paginator.get_page(pagenum)
```

페이지징 대상

한 페이지에 담길 글의 개수

Page 객체가 리턴됨

속성 또는 메서드	기능	제공 객체
count	총 모델 객체 수	Paginator
num_page	총 페이지 수	Paginator
get_page(n)	n 번째 페이지 가져오기	Paginator
page_range	1~페이지수로 구성된 range객체 리턴	Paginator
has_next()	다음 페이지 유무를 bool 형으로 리턴	Page
has_previous()	이전 페이지 유무를 bool 형으로 리턴	Page
previous_page_number()	이전 페이지 번호 리턴	Page
next_page_number()	이후 페이지 번호 리턴	Page

- 페이지징 관련 템플릿에서는 다음과 같이 Page 객체에서 제공하는 메서드를 사용해서 페이지 단위로 요청할 수 있는 링크를 출력한다.

<h5>

```
{% if vlist.has_previous %}
```

```
<a href=?page={{vlist.number|add:-1}}>이전페이지</a>
```

```
{% endif %}
```

```
Page {{ vlist.number }} / {{ vlist.paginator.num_pages }}
```

```
{% if vlist.has_next %}
```

```
<a href=?page={{vlist.number|add:+1}}>다음페이지</a>
```

```
{% endif %}
```

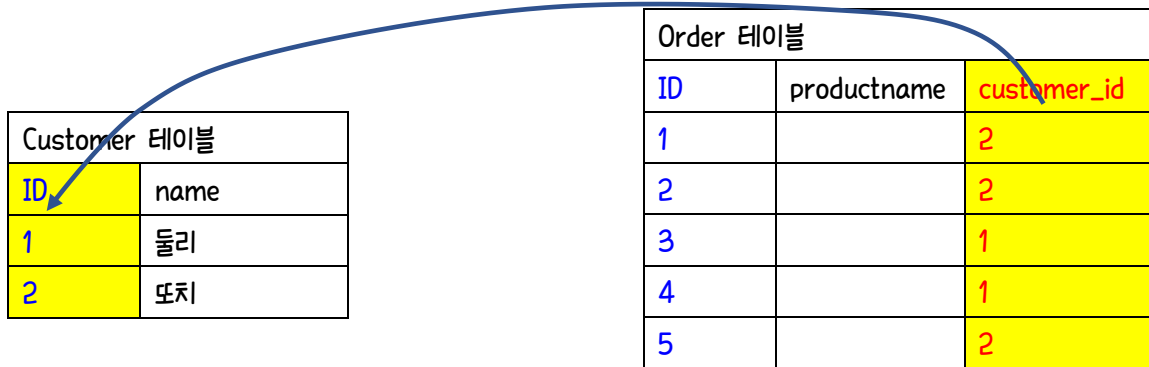
</h5>

Django 의 Relationships

관계형 데이터베이스시스템의 핵심은 테이블 간의 관계 설정이라고 할 수 있다. Django 는 데이터베이스 관계의 가장 흔한 유형인 다대일 (Many-to-One), 다대다 (Many-to-Many) 그리고 일대일 (One-to-One) 관계를 구현할 수 있는 기능을 제공한다.

[다대일 관계 (Many-to-one relationships)]

한 테이블에 있는 두 개 이상의 레코드가 다른 테이블에 있는 하나의 레코드를 참조할 때, 두 모델간의 관계를 다대일 관계라고 한다.



Order 테이블의 customer_id 필드는 Customers 테이블의 기본키 (Primary Key) 인 ID 필드를 참조하고 있다. 이 때, Order 테이블의 Customer_ID 필드를 외래키 (Foreign Key) 필드라고 한다.

ForeignKey

Django 에서 다대일 관계를 설정할 때는 아래와 같이 ForeignKey 를 사용한다. 다른 필드 타입과 마찬가지로 모델 클래스의 속성으로 입력하며, 연결대상이 될 모델 객체를 위치인자로 전달해주어야 하고, on_delete 옵션을 필수로 입력해주어야 한다.

class 모델이름(models.Model):

 필드이름 = models.ForeignKey(연결대상모델, on_delete=삭제옵션)

Customer Table

class Customer(models.Model):

 name = models.CharField(max_length=50)

Order Table

class Order(models.Model):

 customer = models.ForeignKey(Customer, on_delete=models.CASCADE) # 테이블 생성시 해당 컬럼엔 _id가 붙는다.

 productname = models.CharField(max_length=50)

- ForeignKey 필드의 이름

ForeignKey 필드의 필드이름은 자유롭게 설정할 수도 있지만, 연결대상이 되는 모델 클래스명을 모두 소문자로 바꿔서 정하는 것을 권장한다.

customer = models.ForeignKey(Customer)

manufacturer = models.ForeignKey(Manufacturer)

- on_delete 옵션

on_delete 은 참조되는 레코드(부모 레코드)를 삭제할 때, 그 레코드를 참조하는 레코드(자식 레코드)들에 대한 행동을 정의한다.

models.CASCADE: 레코드가 삭제시 이 레코드를 외래키로 참조하고 있는 모든 레코드들을 함께 삭제한다.

models.PROTECT: 외래키가 참조하고 있는 레코드를 삭제하지 못하게 만든다. 삭제를 시도하면 ProtectedError 를 발생시킨다.

models.SET_NULL: 외래키가 참조하고 있는 레코드가 삭제되면, 외래키 필드의 값이 null 이 된다.

외래키 필드에 null=True 옵션이 있을 때만 가능함.

models.SET_DEFAULT: 외래키가 참조하고 있는 레코드가 삭제되면, 외래키 필드의 값이 기본값으로 바뀐다.

default 옵션이 설정되어 있을 때만 가능함.

models.DO_NOTHING: 아무 작업도 하지 않음

models.SET(값 또는 함수): SET() 함수에 값이나 호출가능한 객체를 전달할 수 있으며, 외래키가 참조하고 있는 레코드가 삭제되면 전달된 값 또는 객체를 호출한 결과로 외래키 필드를 채운다.

다대일 관계의 참조와 역참조

위의 Orders, Customers 테이블[테이블 보기] 을 예로 들면, Customer 모델은 Order 모델의 타겟모델이다.
소스모델의 인스턴스에서 타겟모델의 인스턴스를 가져오려면 아래와 같이 관계가 정의된 속성의 이름을 붙여준다.

```
o = Order.objects.get(id=1)
```

o.customer # 1번 주문을 한 고객을 가져온다. → 2번

여기서 반대로 타겟 인스턴스에서 소스 인스턴스를 역참조하려면 아래와 같이 한다.

```
c = Customer.objects.get(id=2)
```

c.order_set.all() # 소스모델의 이름은 Order 이므로, 역참조 매니저의 이름은 order_set 이 된다.

이것의 결과로 1번 고객에 연결된 모든 Order 모델의 인스턴스들이 쿼리셋으로 리턴된다.

참조-역참조 요약

	ManyToOneField	OneToOneField
참조(소스->타겟)	source.attrname	source.attrname
역참조(타겟->소스)	target.lower_sourcename_set	target.lower_source

```
class Visitor(models.Model):  
    name = models.CharField(max_length=6)  
    memo = models.TextField()  
    writedate = models.DateTimeField(auto_now_add=True)
```

타겟 모델 객체

```
class Reply(models.Model):  
    content = models.CharField(max_length=80)  
    visitor = models.ForeignKey(Visitor, on_delete=models.CASCADE)
```

소스 모델 객체

DataBase 개요

[데이터의 분류]

- 정형 데이터, 반정형 데이터, 비정형 데이터

- 정형 데이터(structured data)

구조화된 데이터, 즉 미리 정해진 구조에 따라 저장된 데이터

예 : 엑셀의 스프레드시트, CSV, 관계 데이터베이스의 테이블

	A	B	C	D
1	일자	배송 업체	배송 건수	전일대비 상승률
2	2019-03-02	빠르다 택배	100	0%
3	2019-03-02	한빛 택배	200	10%
4	2019-03-02	안전 택배	50	3%
5	2019-03-02	당일 택배	30	-10%

- 반정형 데이터(semi-structured data)

구조에 따라 저장된 데이터이지만 데이터 내용 안에 구조에 대한 설명이 함께 존재

구조를 파악하는 파싱(parsing) 과정이 필요

보통 파일 형태로 저장

예 : 웹에서 데이터를 교환하기 위해 작성하는 HTML, XML, JSON 문서나 웹 로그, 센서 데이터(JSON) 등

```
{  
  "이름" : "오형준",  
  "나이" : 23,  
  "성별" : "남"  
}
```

```
<친구정보>  
  <이름> 오형준 </이름>  
  <나이> 23 </나이>  
  <성별> 남 </성별>  
</친구정보>
```

- 비정형 데이터(unstructured data)

정해진 구조가 없이 저장된 데이터

예 : 소셜 데이터의 텍스트, 영상, 이미지, 워드나 PDF 문서와 같은 멀티미디어 데이터



[데이터와 정보]

데이터 : 컴퓨터 사용 시 불규칙하게 만들어지는 다양하고 많은 값들

정보 : 체계적이고 조직적으로 관리하고 사용되는 데이터, 여러 가지 값들로 표현되는 데이터를 의미 있고 가치 있는 형태로 가공해 놓은 것



[데이터베이스와 데이터베이스 관리 시스템]

데이터베이스 : 특정 조직 내에서 다수의 사용자들이 공유해 사용할 수 있도록 통합하고 저장한 운영 데이터의 집합체

데이터베이스 관리 시스템(DBMS) : 데이터베이스를 생성하여 안정적이고 효율적으로 운영하는 데 필요한 기능들을 제공하는 소프트웨어

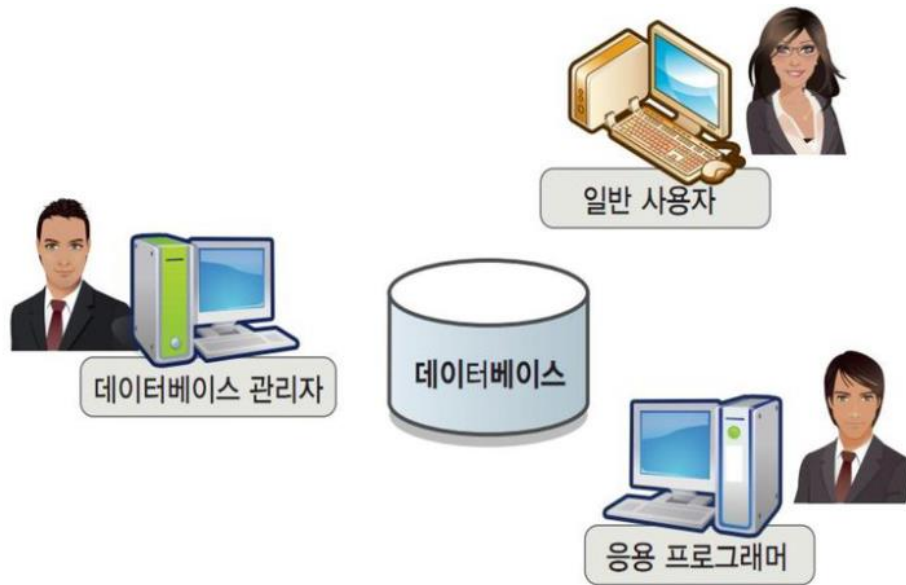


[데이터베이스 구조 데이터베이스 관리 시스템(DBMS)의 종류]

개발사	DBMS	특징
마이크로소프트	액세스	윈도우즈 플랫폼으로 중소 규모 데이터베이스를 위한 데스크톱용 DBMS
	SQL 서버	저렴한 제품 가격으로 Windows NT 플랫폼에서 최적의 성능을 발휘
IBM	인포믹스	성능이 뛰어나며 병렬 처리를 위한 멀티스레드(Multithread) 지원
	DB2	다수 사용자가 다수 관계형 데이터베이스를 동시에 접근할 수 있는 대형 데이터베이스를 위한 시스템
오라클	Oracle	PC급에서 메인 프레임급까지 모두 설치할 수 있으며 분산 처리 지원 기능이 우수
MySQL	MySQL	다양한 플랫폼과 API를 지원하는 비상업용 DBMS

[데이터베이스 사용자]

일반 사용자, 응용 프로그래머, 데이터베이스 관리자



[데이터베이스 특징]

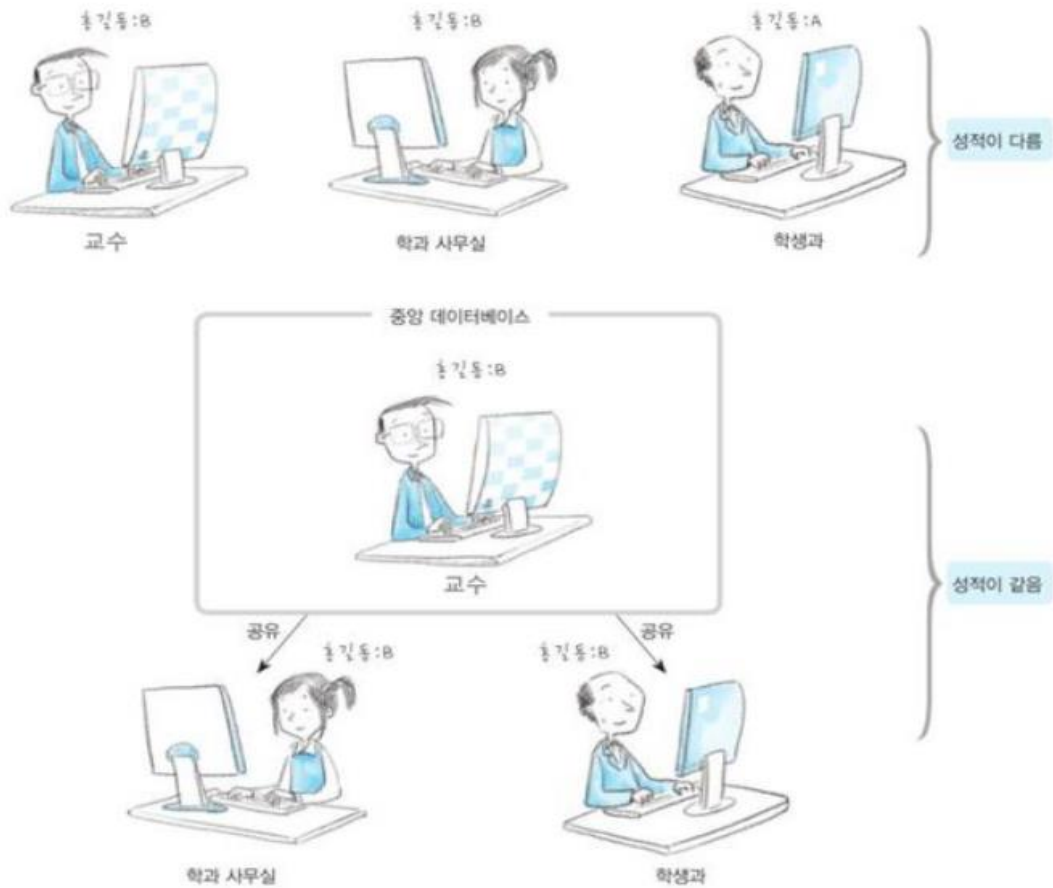
데이터의 중복성을 최소화 할 수 있다

데이터베이스를 이용하면 업무의 흐름에 따라 데이터를 통합 및 분리하여 관리할 수 있게 되므로 이러한 중복성을 줄일 수 있음



데이터의 일관성을 유지할 수 있다

데이터의 불일치성(Inconsistency)을 미리 방지하여 데이터를 정확하게 만들고 사용자에게 신뢰할 만한 정보를 제공할 수 있음.



데이터의 무결성을 유지할 수 있다

무결성(Integrity) : 데이터베이스에 정확한 데이터가 유지되고 있음을 보장하는 것으로 데이터베이스에서 가장 중요한 개념
 제약조건에 맞지 않는 데이터는 아예 입력되지 않도록 방지하는 기능을 제공해 데이터베이스의 무결성을 유지함



데이터의 독립성을 유지할 수 있다

독립성 : 데이터의 표현 방법이나 저장 위치가 변하더라도 응용 프로그램에는 아무런 영향을 미치지 않는 것
 데이터 베이스의 경우 데이터를 테이블 구조로 관리하기 때문에 독립성을 더욱 강화할 수 있음

데이터의 공유성을 최대화할 수 있다

데이터베이스는 공동작업에 맞게 구조적으로 설계되며 통합된 체계에 의해 저장된 값들이 유지 · 관리되어야 하며, 조직의 구성원들은 응용 프로그램을 통해 데이터베이스에 저장된 공유 데이터들로 부터 각자의 업무에 필요한 정보를 생성할 수 있음

데이터의 보안성을 최대화 할 수 있다

데이터베이스 시스템의 성능을 평가하는 중요한 요소로, 최근 대부분의 데이터베이스 시스템은 이러한 보안 기능을 제공함

데이터를 표준화하여 관리할 수 있다

데이터를 사용 목적 등의 유형별로 분류해 데이터의 형식이나 길이, 이름 등을 설정할 수 있음

[관계형 데이터베이스]

관계형 데이터베이스는 키와 값들의 간단한 관계를 테이블화 시킨 매우 간단한 원칙의 전산정보 데이터베이스이다.

1969년 E.F.Codd라는 학자가 수학 모델에 근거해 고안하였다.

데이터베이스는 **테이블(Table)**이라 불리는 최소 단위로 구성되며 테이블은 하나 이상의 필드(열)로 이루어진다.

- 관계형 데이터베이스 구성 요소



테이블 : 릴레이션 혹은 **엔티티**, **열(Column)**과 **행(Row)**으로 구성됨

필드 : 속성 혹은 컬럼, 열에 해당, 데이터 값을 기억하는 기억 단위

레코드 : 튜플, 테이블의 행에 해당

후보 키 : 한 테이블 내에서 데이터 레코드를 고유하게 식별할 수 있는 필드

기본 키 : 후보 키 중에서 대표로 선정된 키

외래 키 또는 참조 키 : 관계가 설정된 다른 테이블의 기본 키를 참조하는 키



[DBMS 을 활용한 데이터베이스 구축의 장점]

1. 데이터 중복을 통제할 수 있다.
2. 데이터 독립성이 확보된다.
3. 데이터를 동시 공유할 수 있다.
4. 데이터 보안이 향상된다.
5. 데이터 무결성을 유지할 수 있다.
6. 표준화할 수 있다.
7. 장애 발생 시 회복이 가능하다.
8. 응용 프로그램 개발 비용이 줄어든다.

[DBMS 을 활용한 데이터베이스 구축의 단점]

1. 비용이 많이 든다.
2. 백업과 회복 방법이 복잡하다.
3. 중앙 집중 관리로 인한 취약점이 존재한다.

[데이터 모델링의 개요]

데이터 모델링이란?

현실 세계에 존재하는 데이터를 추상화를 통해 컴퓨터 세계의 데이터베이스로 옮기는 변환 과정이다.

시스템을 구축하기 위해, 어떤 데이터가 존재하는지 또는 업무가 필요로 하는 정보가 무엇인지를 분석하는 방법이다.

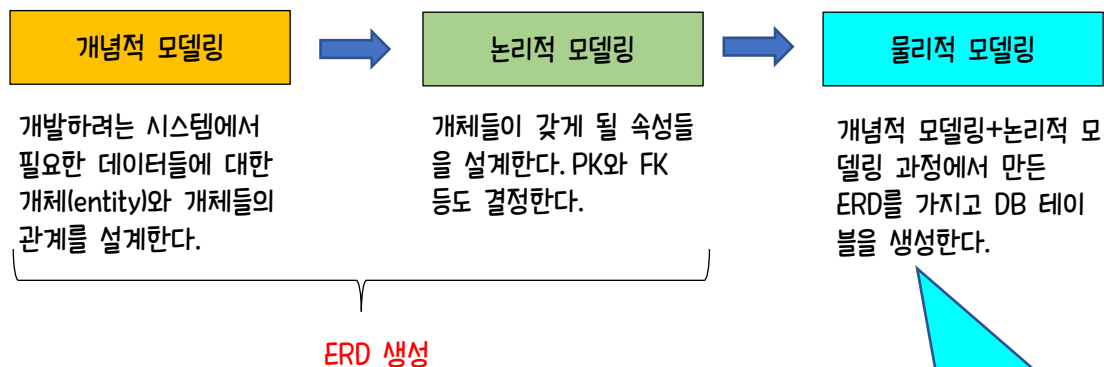
데이터베이스를 구축하기 위한 분석과 설계를 진행하는 것으로 데이터들을 저장하기 위한 테이블 설계 과정이다.

개념적 모델링: 개체와 개체들 간의 관계에서 E-R 다이어그램을 만드는 과정

논리적 모델링: E-R 다이어그램을 사용하여 관계 스키마 모델을 만드는 과정

물리적 모델링: 관계 스키마 모델의 물리적 구조를 정의하고 구현하는 과정

E-R 다이어그램을 생성하는 과정에서 결정



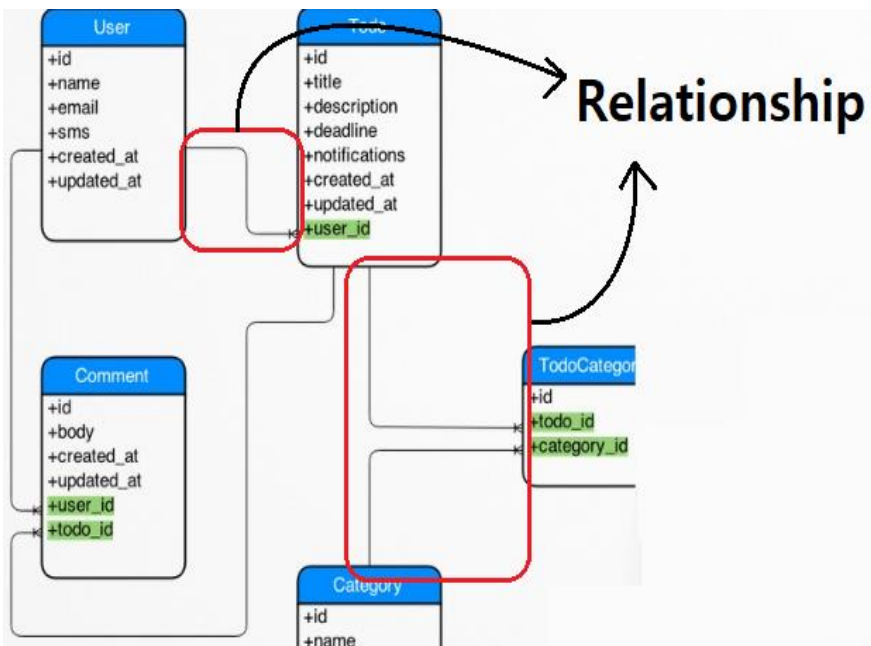
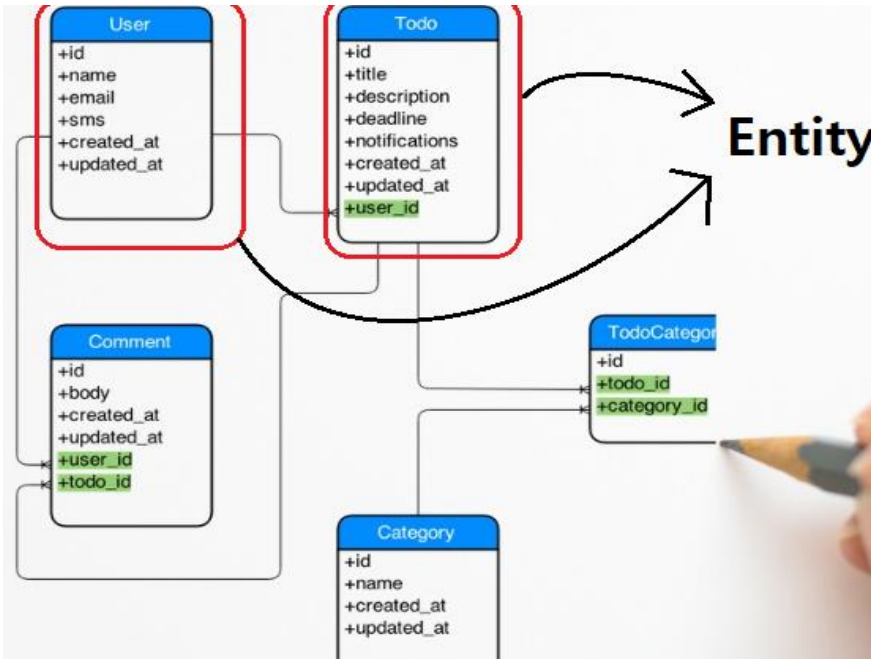
Django에서는 DB 테이블 생성 대신 Model 클래스를 생성

[Entity Relationship Diagram]

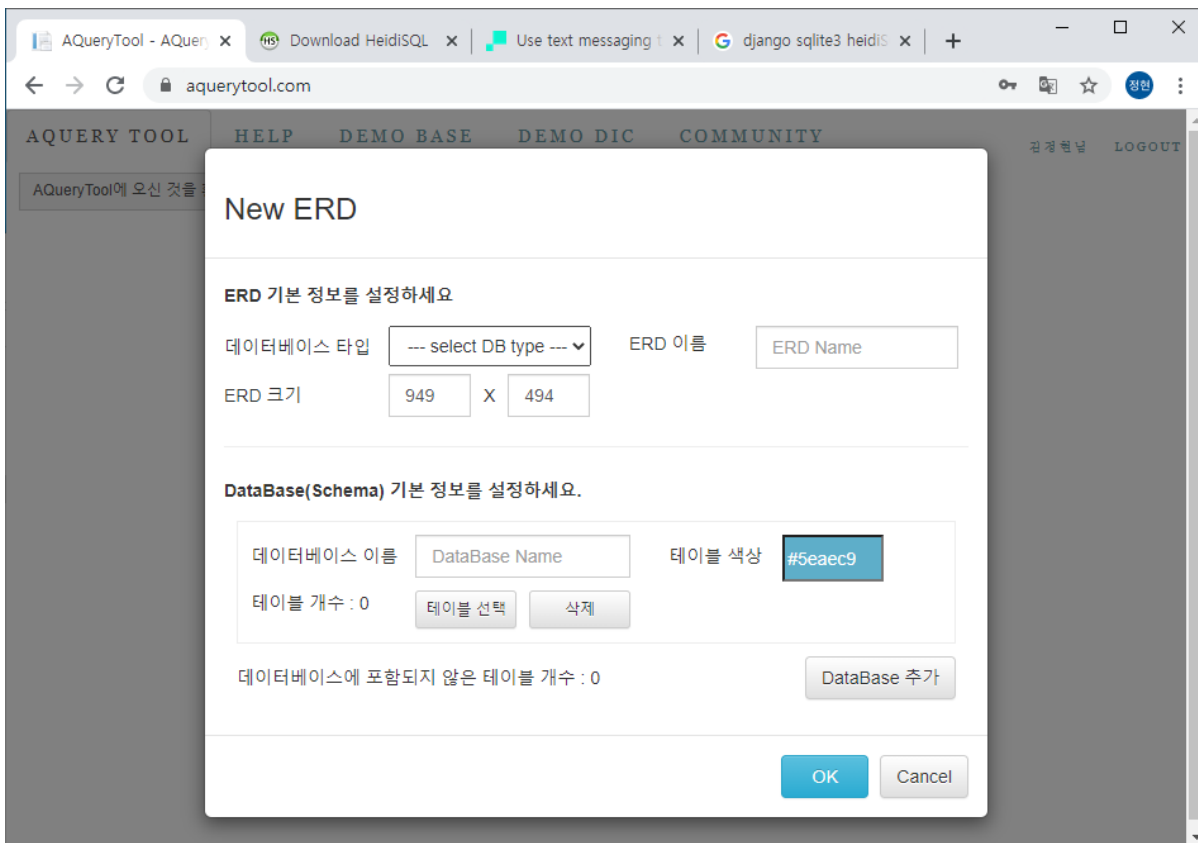
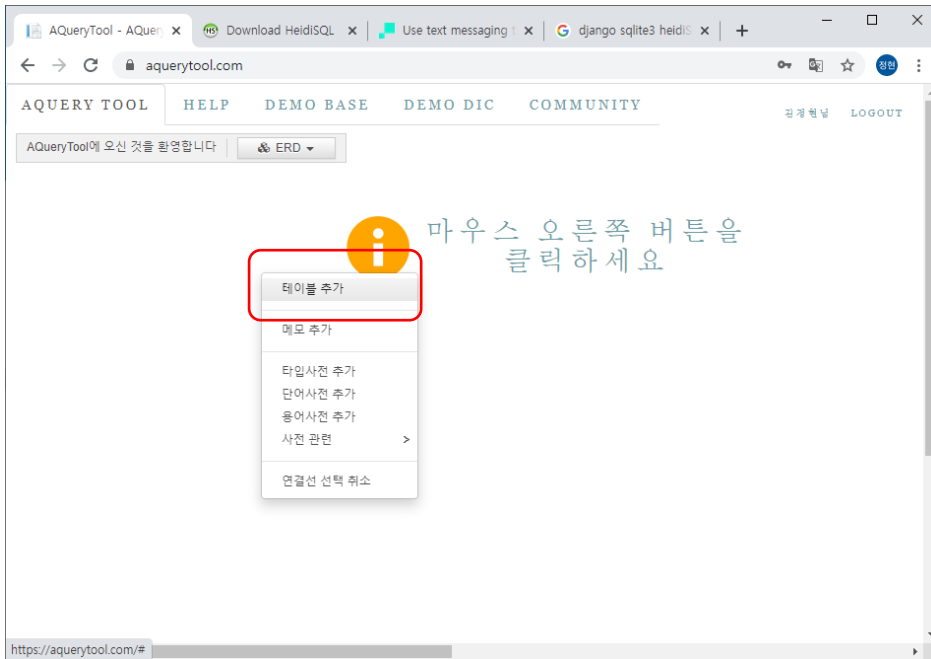
Entity Relationship Diagram은 E-R 다이어그램이라고 불리우며 ERD 라고 줄여 부르기도 한다.

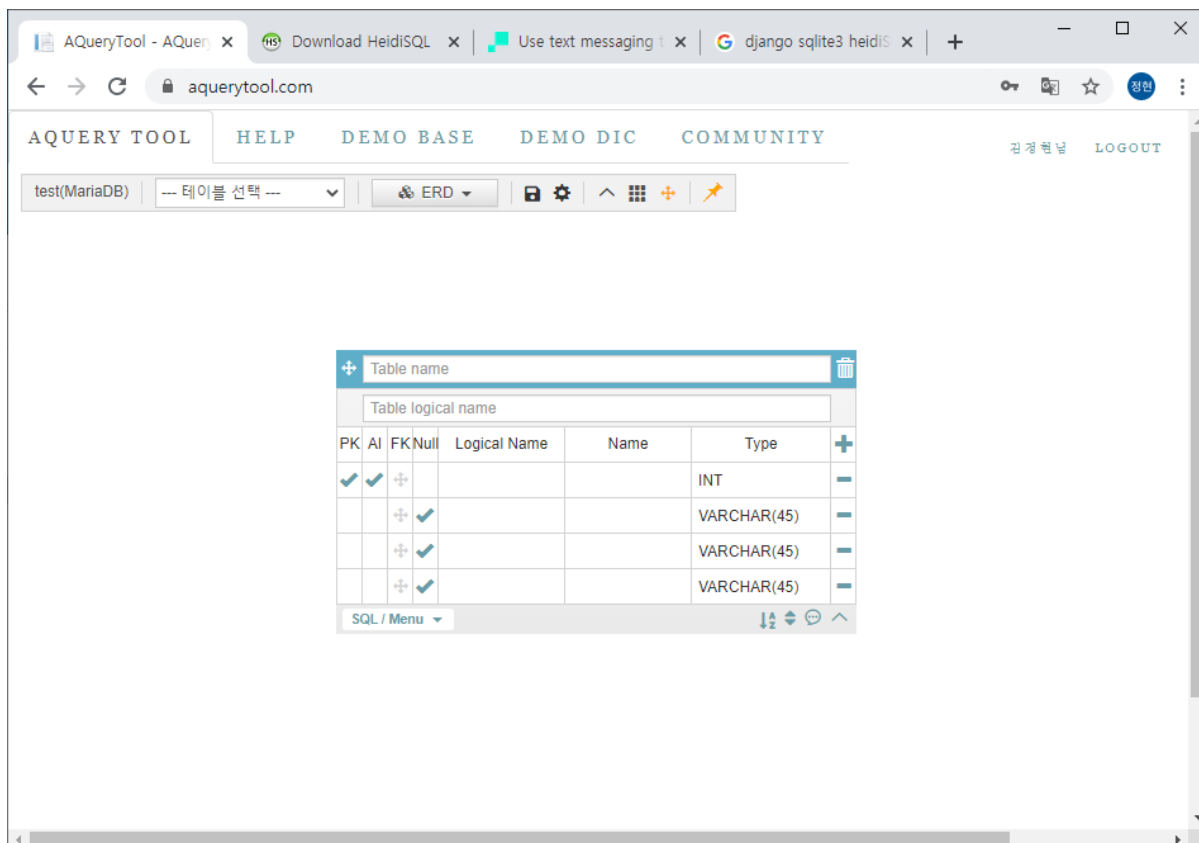
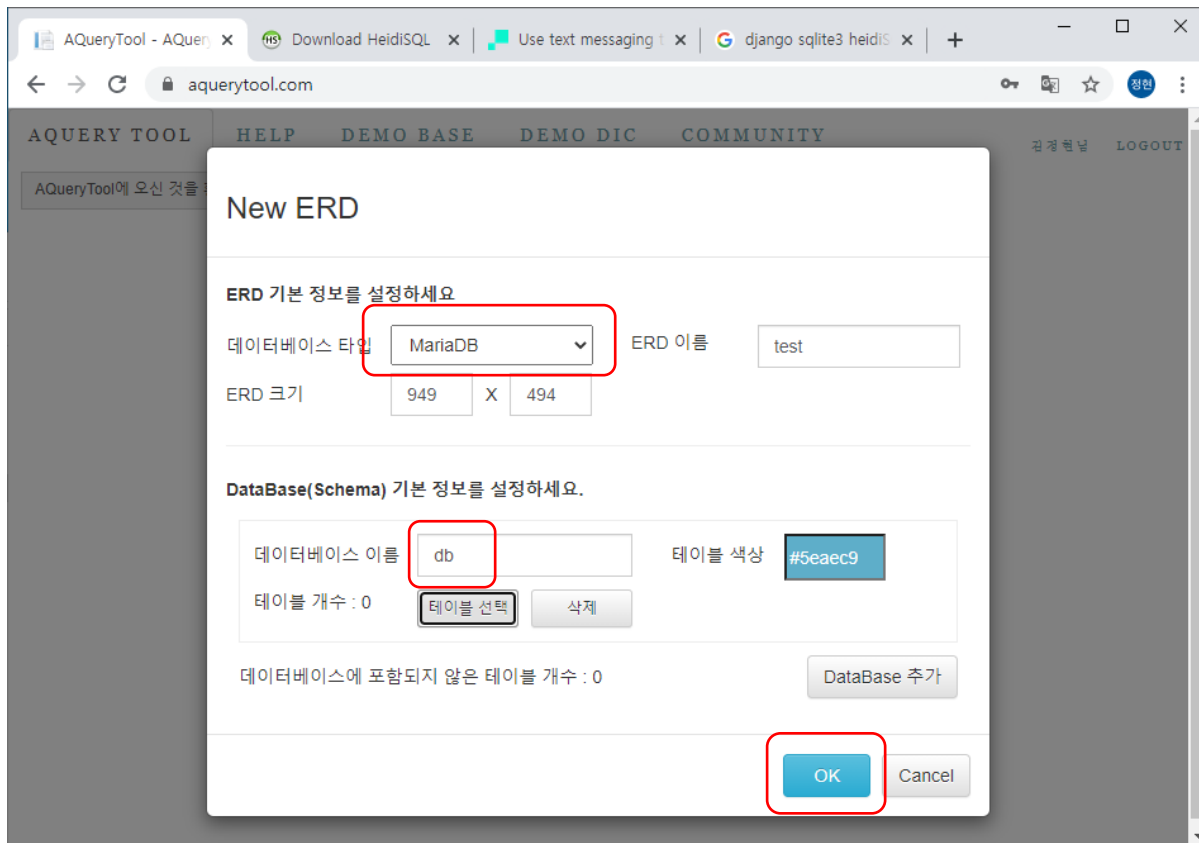
영어 약자 그대로 '존재하고 있는 것(Entity)들의 관계(Relationship)을 나타낸 도표(Diagram)' 이며 여기서 말하는 존재하고 있는 것이란 데이터를 뜻하니 데이터들의 관계를 나타낸 도표라고 할 수 있다.

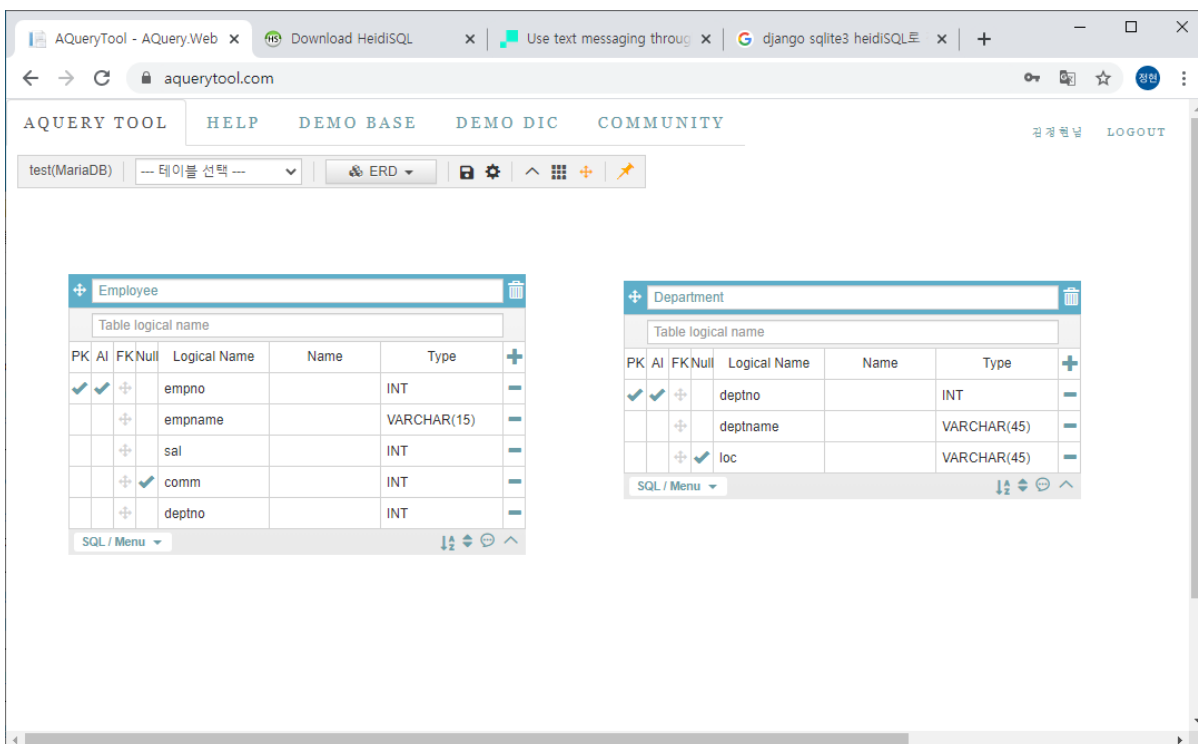
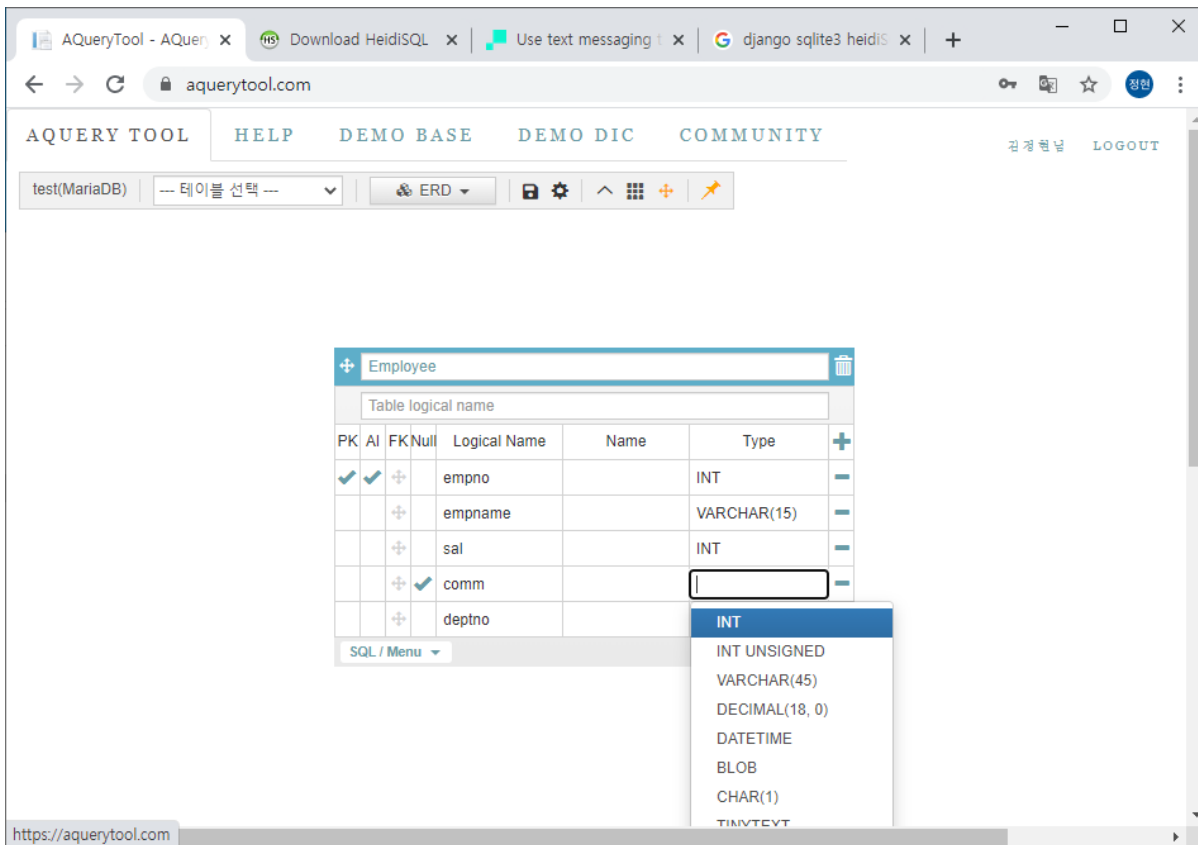
현실세계의 요구사항(Requirements)으로 부터 Database 설계과정에서 활용된다. 즉, 개념을 모델링하는 것으로 개체(entity)와 속성(attribute), 관계성(relationship)을 표현한다.

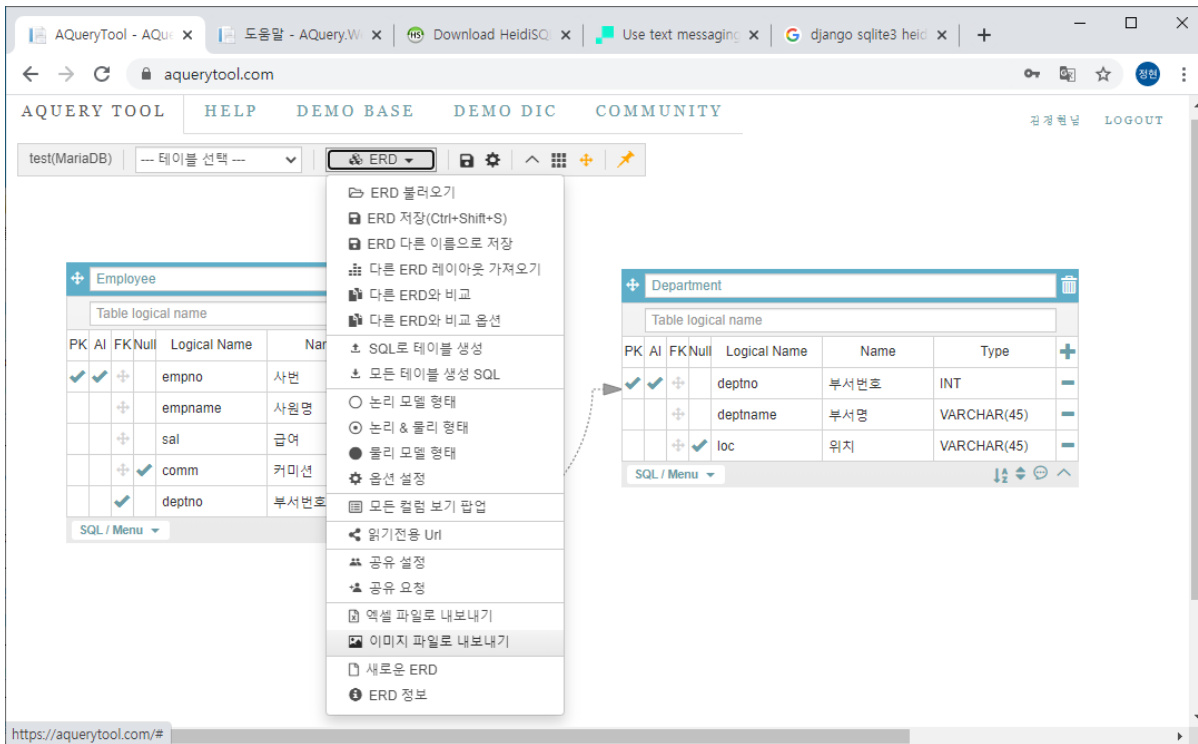
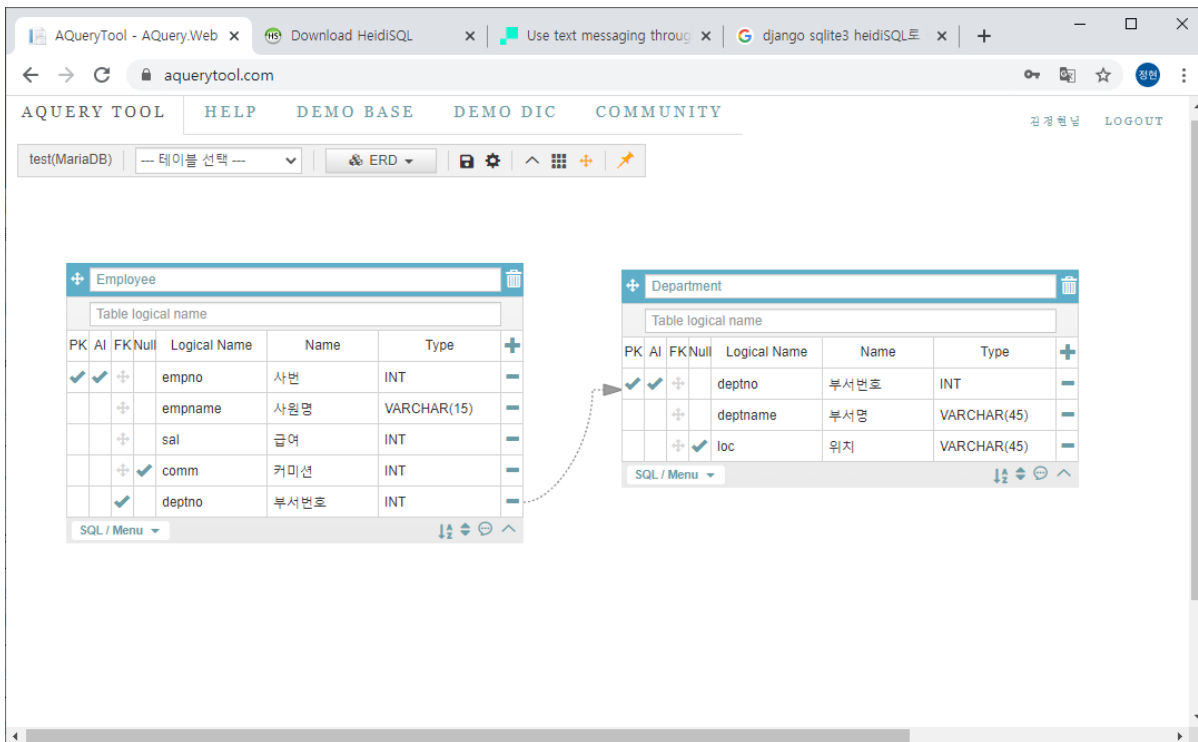


[AQueryTool로 ERD 그리기] 작성 무료 웹 - <https://aquerytool.com/>









테이블 설계 시 데이터의 중복을 최소화하기 위해 테이블을 나눈다. 다음 예를 체크해 보자.

고객 이름	출생년도	주소	연락처	구매한 물건	단가(천 원)	수량
이승기	1987	서울	011-111-1111			
김범수	1979	경남	011-222-2222	운동화	30	2
김범수	1979	경남	011-222-2222	노트북	1000	1
김경호	1971	전남	019-333-3333			
조용필	1950	경기	011-444-4444	모니터	200	1
바비킴	1973	서울	010-000-0000	모니터	200	5
윤종신	1969	경남	안 남김			
김범수	1979	경남	011-222-2222	청바지	50	3
임재범	1963	서울	016-666-6666			
바비킴	1973	서울	010-000-0000	메모리	80	10
성시경	1979	경남	안 남김	책	15	5
은지원	1978	경북	011-888-8888	책	15	2
임재범	1963	서울	016-666-6666			
은지원	1978	경북	011-888-8888	청바지	50	1
바비킴	1973	서울	010-000-0000	운동화	30	2
은지원	1978	경북	011-888-8888			
은지원	1978	경북	011-888-8888	책	15	1
바비킴	1973	서울	010-000-0000	운동화	30	2
조관우	1965	경기	018-999-9999			

방문 내역 & 구매내역 데이터이다.
메모장이나 엑셀로 작성되었다고
가정한다.

고객 이름	출생년도	주소	연락처	구매한 물건	단가(천 원)	수량
이승기	1987	서울	011-111-1111			
김경호	1971	전남	019-333-3333			
윤종신	1969	경남	안 남김			
임재범	1963	서울	016-666-6666			
임재범	1963	서울	016-666-6666			
은지원	1978	경북	011-888-8888			
조관우	1965	경기	018-999-9999			
김범수	1979	경남	011-222-2222	운동화	30	2
김범수	1979	경남	011-222-2222	노트북	1000	1
조용필	1950	경기	011-444-4444	모니터	200	1
바비킴	1973	서울	010-000-0000	모니터	200	5
김범수	1979	경남	011-222-2222	청바지	50	3
바비킴	1973	서울	010-000-0000	메모리	80	10
성시경	1979	경남	안 남김	책	15	5
은지원	1978	경북	011-888-8888	책	15	2
은지원	1978	경북	011-888-8888	청바지	50	1
바비킴	1973	서울	010-000-0000	운동화	30	2
은지원	1978	경북	011-888-8888	책	15	1
바비킴	1973	서울	010-000-0000	운동화	30	2

기록된 내용에서 물건 구매 내역이
없는 고객 위로 정렬한다.
L자형 테이블이 되어 낭비되는
공간 생기게 된다.

고객 테이블

고객 이름	출생년도	주소	연락처
이승기	1987	서울	011-111-1111
김경호	1971	전남	019-333-3333
윤종신	1969	경남	안 남김
임재범	1963	서울	016-666-6666
임재범	1963	서울	016-666-6666
은지원	1978	경북	011-888-8888
조관우	1965	경기	018-999-9999
김범수	1979	경남	011-222-2222
김범수	1979	경남	011-222-2222
조용필	1950	경기	011-444-4444
바비킴	1973	서울	010-000-0000
김범수	1979	경남	011-222-2222
바비킴	1973	서울	010-000-0000
성시경	1979	경남	안 남김
은지원	1978	경북	011-888-8888
은지원	1978	경북	011-888-8888
바비킴	1973	서울	010-000-0000
은지원	1978	경북	011-888-8888
바비킴	1973	서울	010-000-0000

구매 테이블

구매한 물건	단가(천 원)	수량
운동화	30	2
노트북	1000	1
모니터	200	1
모니터	200	5
청바지	50	3
메모리	80	10
책	15	5
책	15	2
청바지	50	1
운동화	30	2
책	15	1
운동화	30	2

L자형 테이블에서 빈칸이 있는
곳과 없는 곳으로 분류한다.

고객테이블, 구매테이블로
분류하여 공간을 절약하고
고객 테이블 중복 제거한다.

기본 키 (PK, Primary Key)필요

- 고객을 구분할 수 있는 구분자로 고객
이름 열 사용
- 각 행을 구분하는 유일한 값
- 기본 키의 조건은 중복되지 않고
비어 있지 않아야 함
- 구매 테이블에 '누가 구매했는지'
표기 위해 고객 이름 필요



[테이블 간의 업무적인 연관성(Relation) 정의]

주 (Master)가 되는 쪽이 **부모 테이블**, 참조하는 쪽이 **자식 테이블**

[데이터 무결성을 위한 데이터 제약 조건]

개체 무결성

기본키(PK)로 제약조건을 지정한 속성은 공백(NULL) 값이나 중복된 값을 가질 수 없다.

(예)

‘사원’ 테이블에서 ‘사원번호’가 기본키로 정의되면 튜플을 추가할 때 ‘주민번호’나 ‘성명’ 필드에는 값을 입력하지 않아도 되지만 ‘사원번호’ 속성에는 반드시 값이 입력되어야 한다. 또한 ‘사원번호’ 속성에는 이미 한 번 입력한 속성값을 중복하여 입력할 수 없다.

참조 무결성

참조키(FK)를 통해 참조할 수 없는 참조키 값을 가질 수 없도록 함으로써 두 테이블 간의 데이터 무결성을 유지하는 것이다.

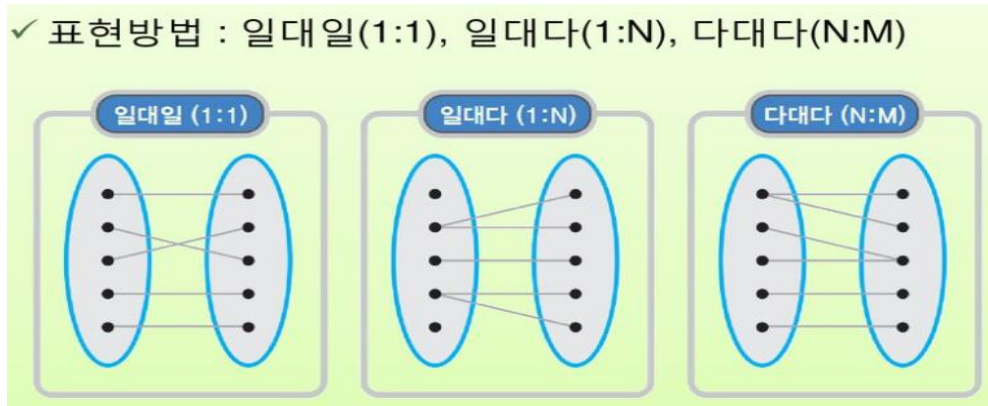
참조키 값은 NULL이거나 참조 테이블의 기본키 값과 동일해야 한다.

(예) ‘사원’ 테이블의 ‘부서 번호’ 속성에는 ‘부서’ 테이블의 ‘부서 번호’ 속성에 없는 값을 입력할 수 없다.

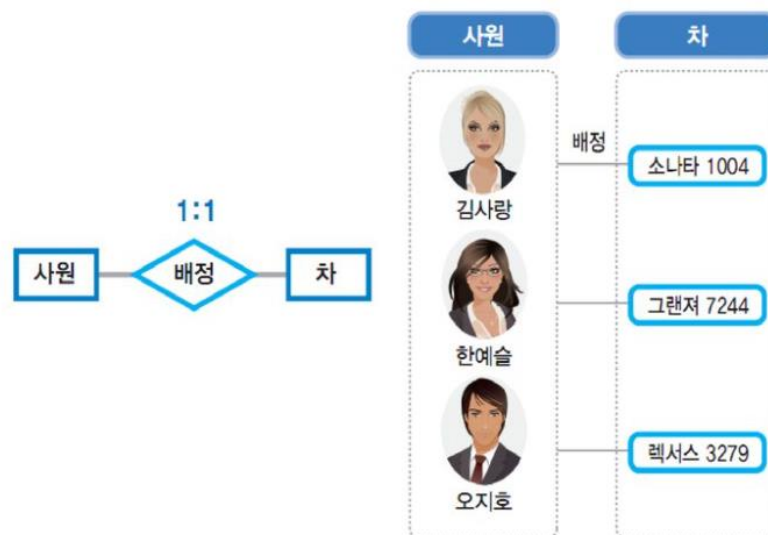


[테이블과 테이블의 관계 유형]

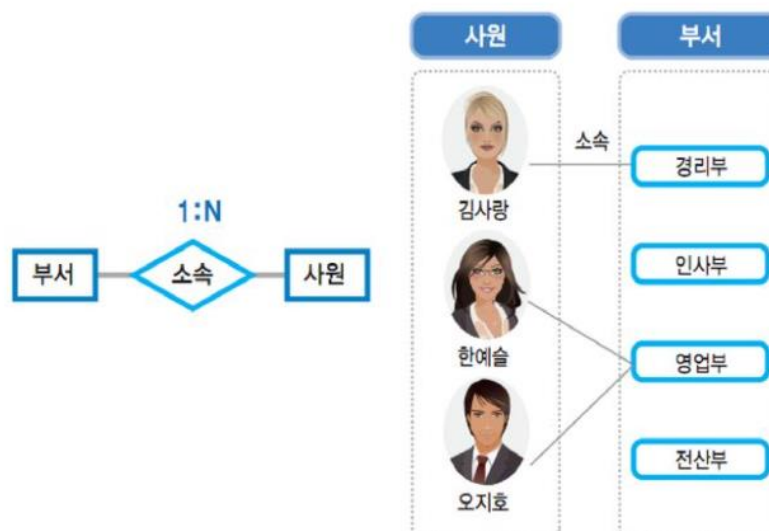
표현방법 : 일대일(1:1), 일대다(1:N), 다대다(N:M)



일대일(One To One) 관계(1:1)



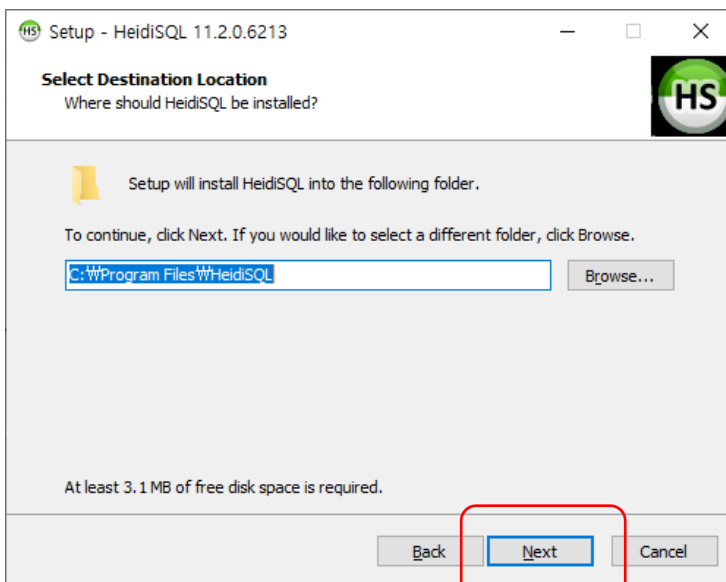
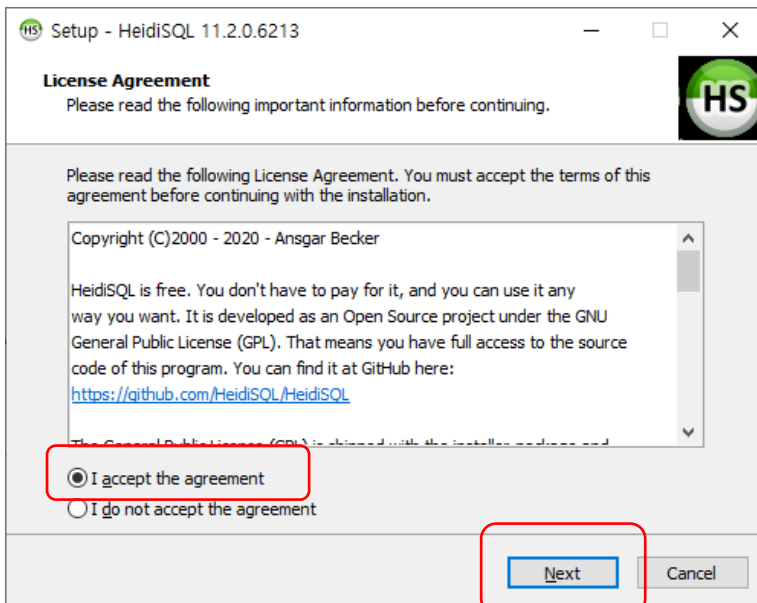
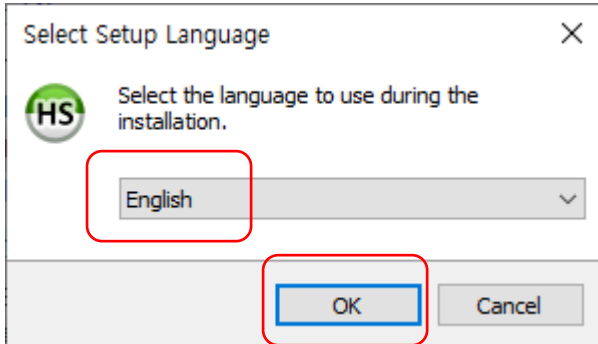
일대다(One To Many) 관계(1:N)

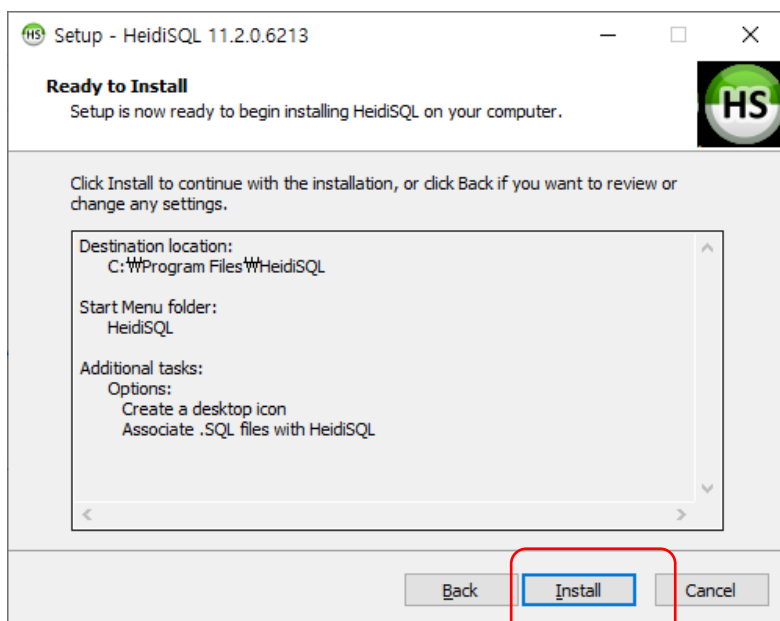
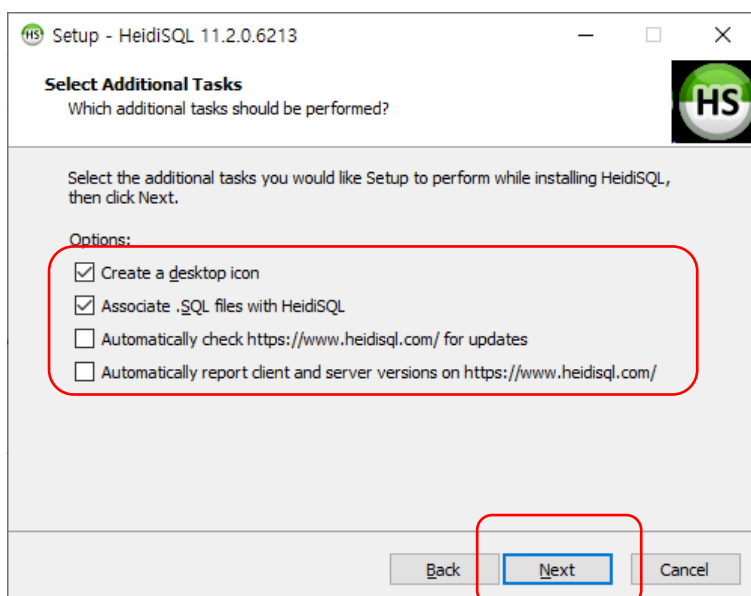
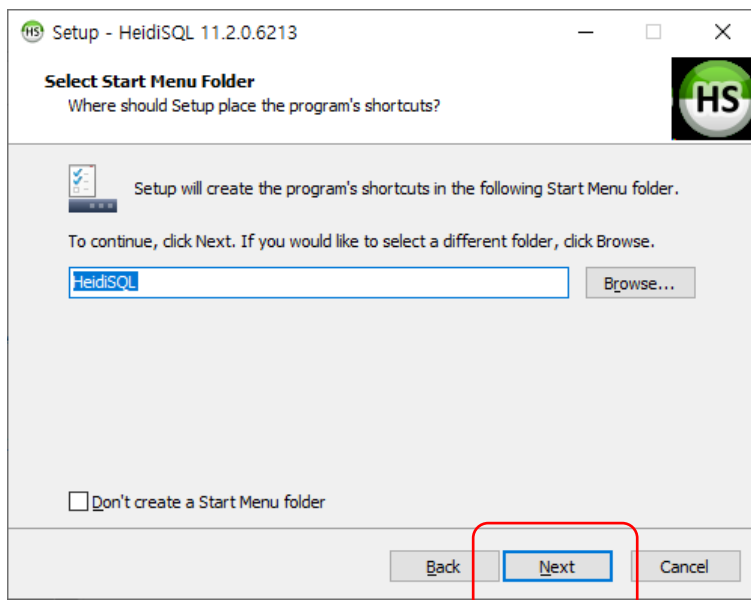


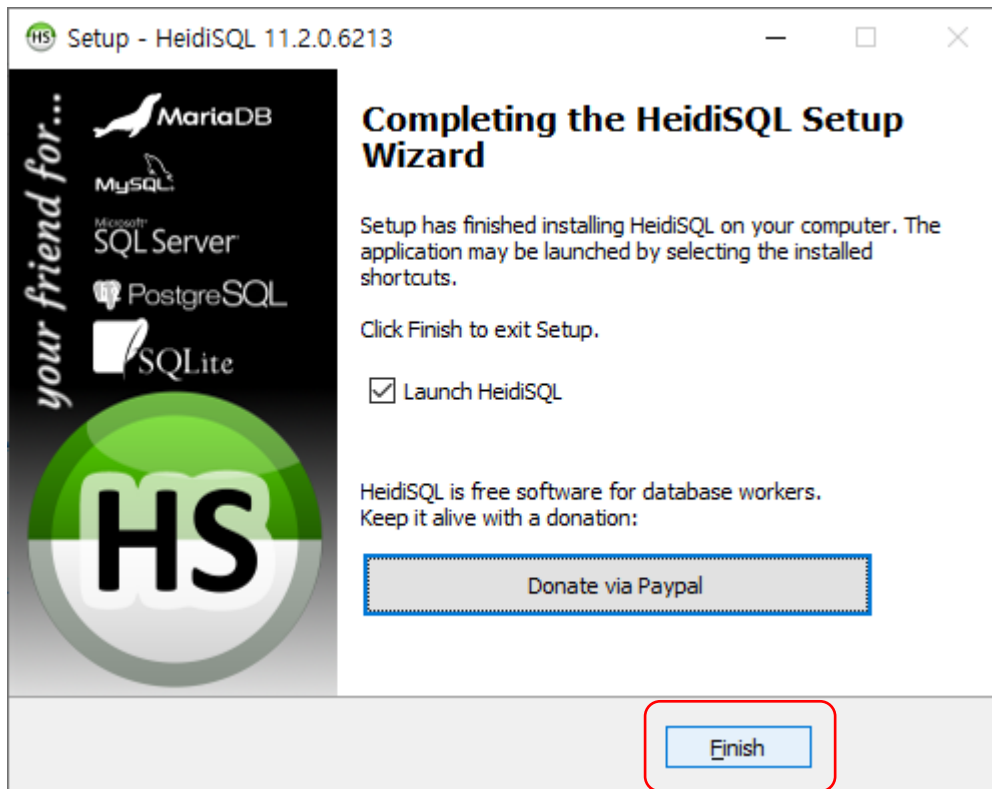
[HeidiSQL]

HeidiSQL은 이전에는 'MySQL Front'로 알려졌던 제품으로서 MySQL, MariaDB 그리고 SQLite 등 DBMS를 직접 접속하여 사용하려는 경우에 선택할 수 있는 DBMS 클라이언트 무료 프로그램이다.

제공된 HeidiSQL_11.2.0.6213_Setup.exe 를 더블 클릭하여 설치를 시작한다.



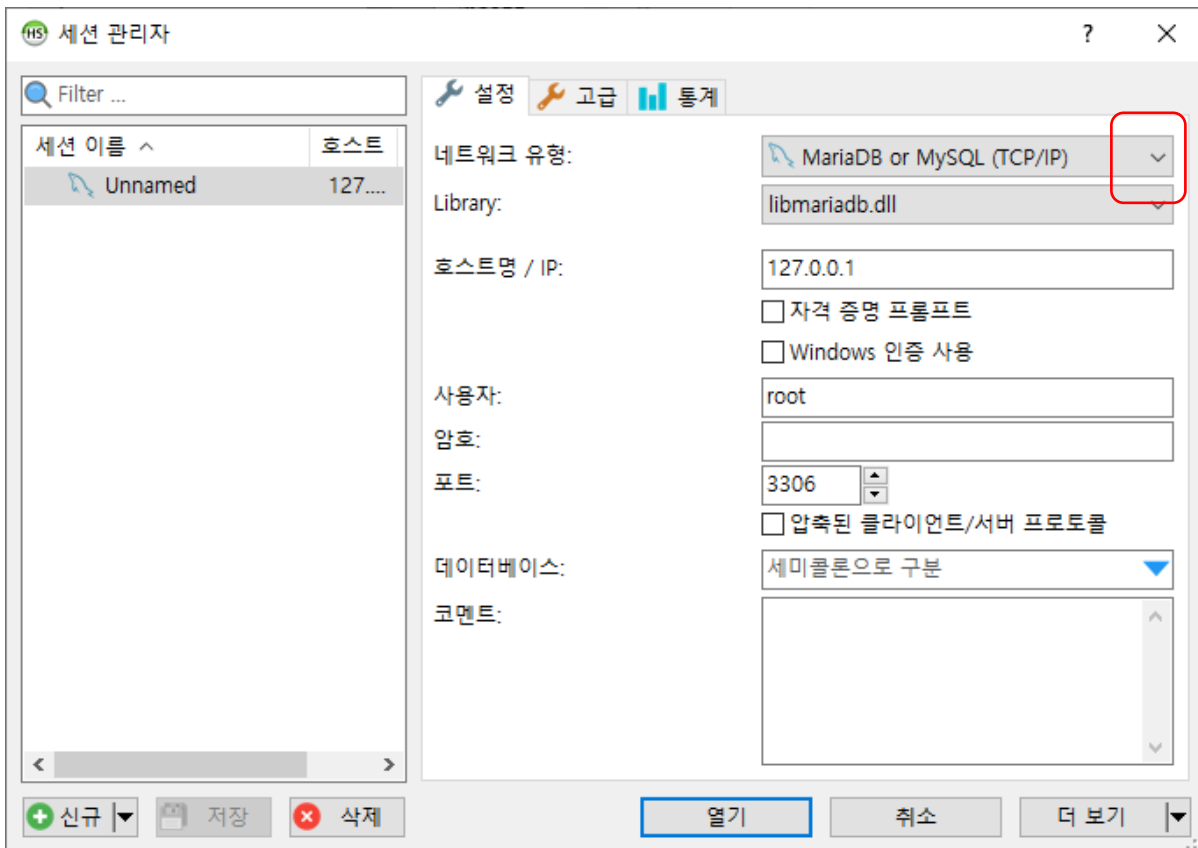




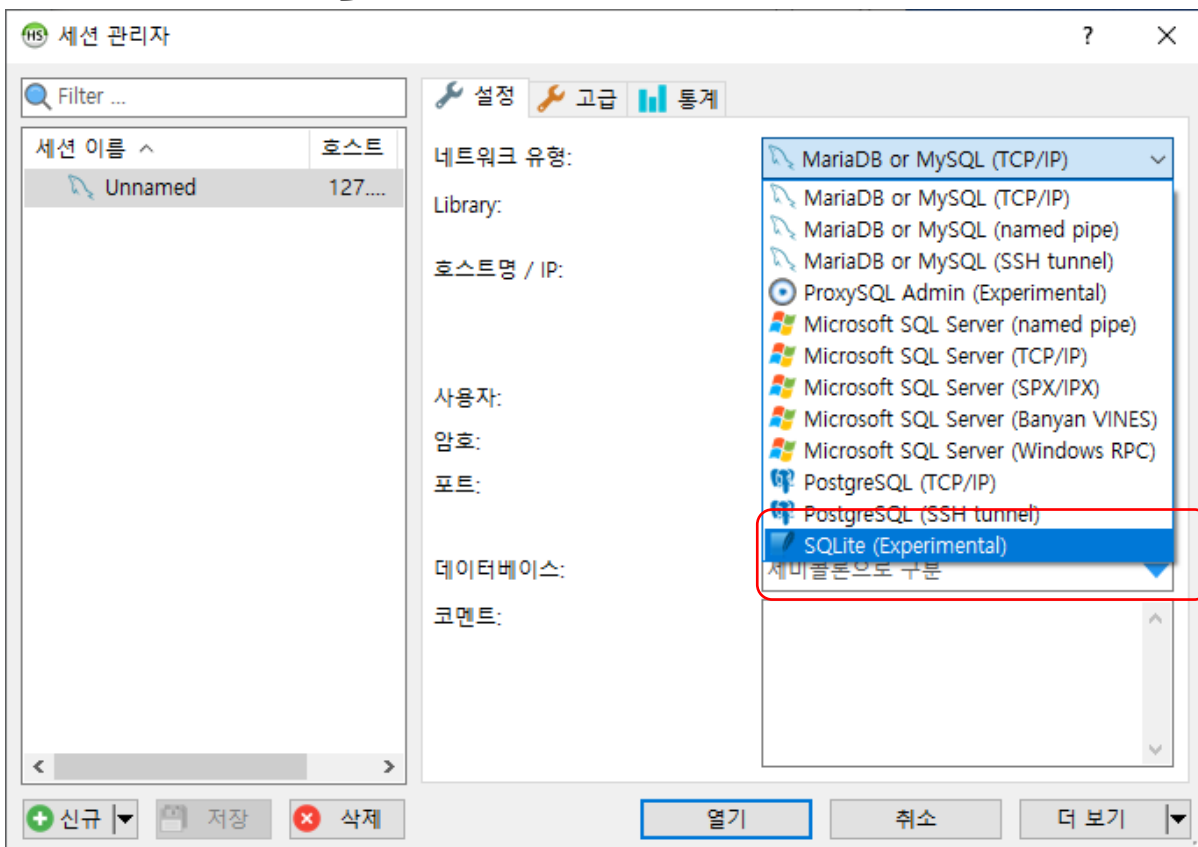
자동으로 HeidiSQL이 기동되며 다음 화면이 출력된다.



신규 버튼을 클릭하면 다음 화면의 서버 윈도우가 출력된다.



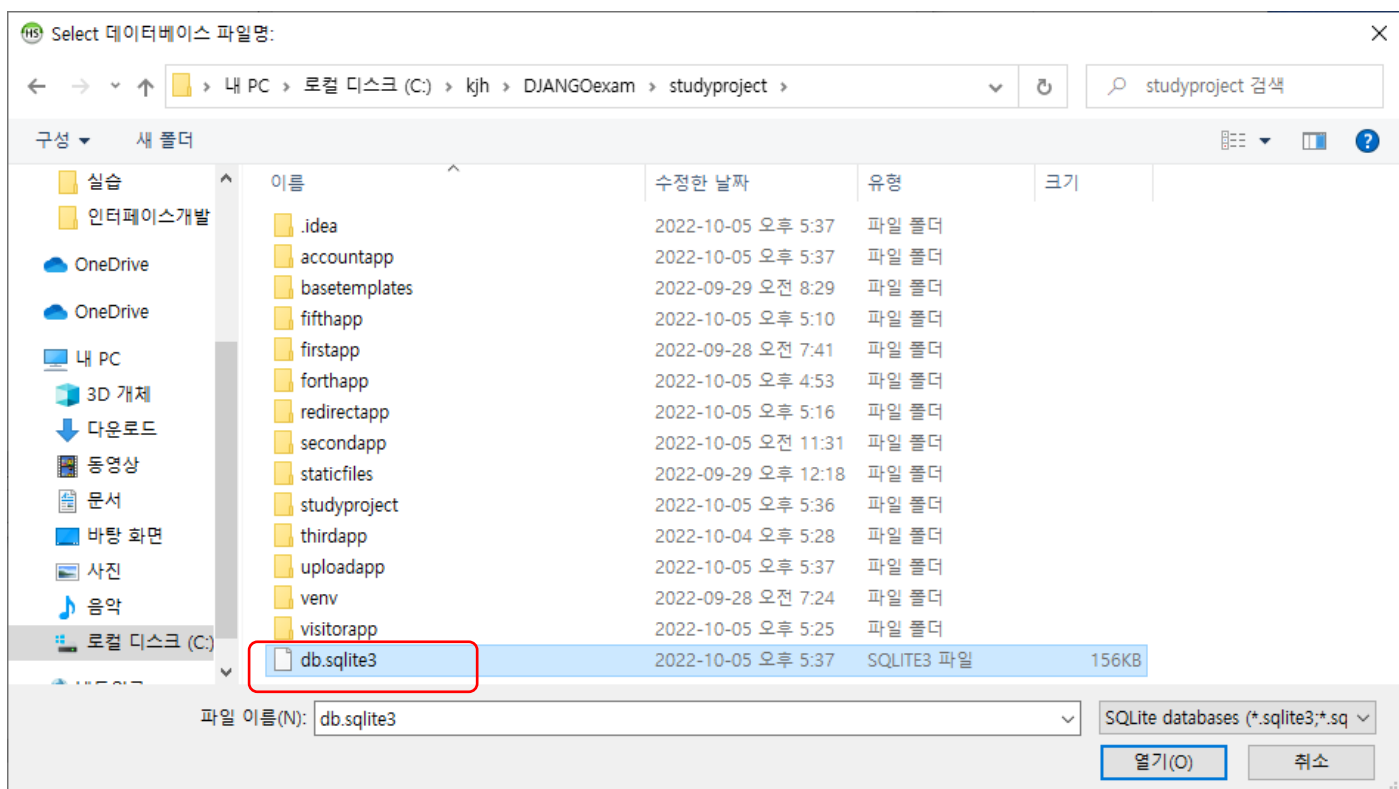
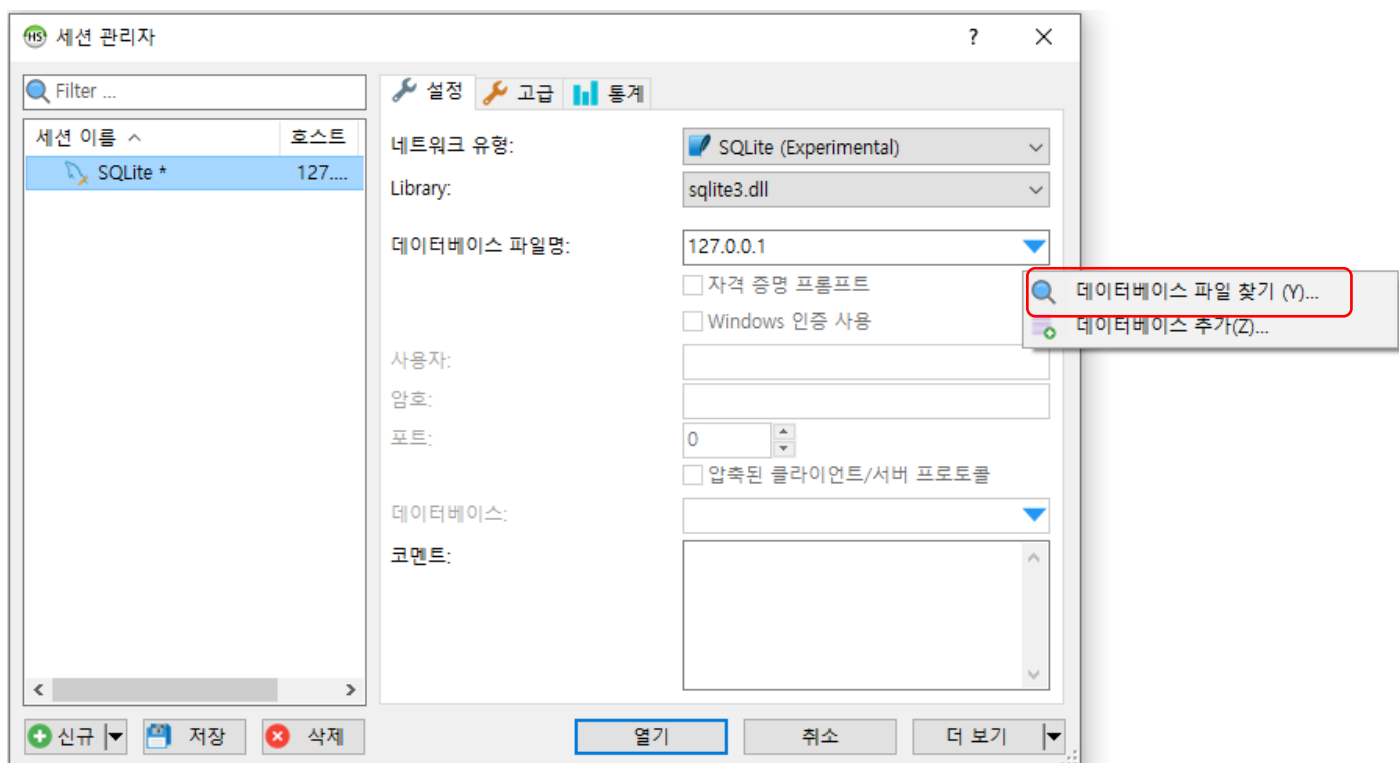
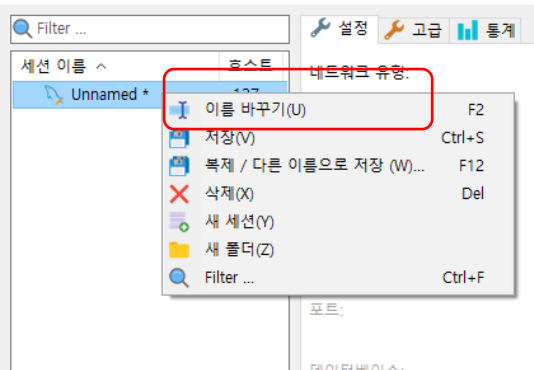
네트워크 유형에서 SQLite 항목을 선택한다.

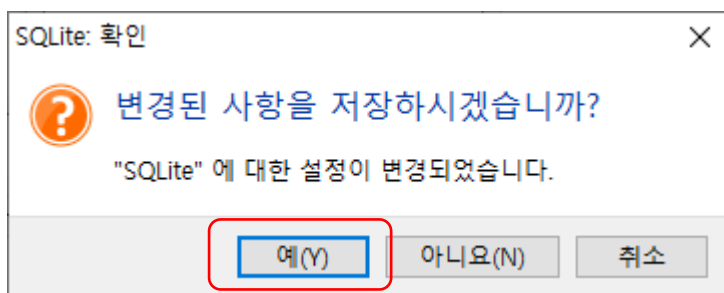
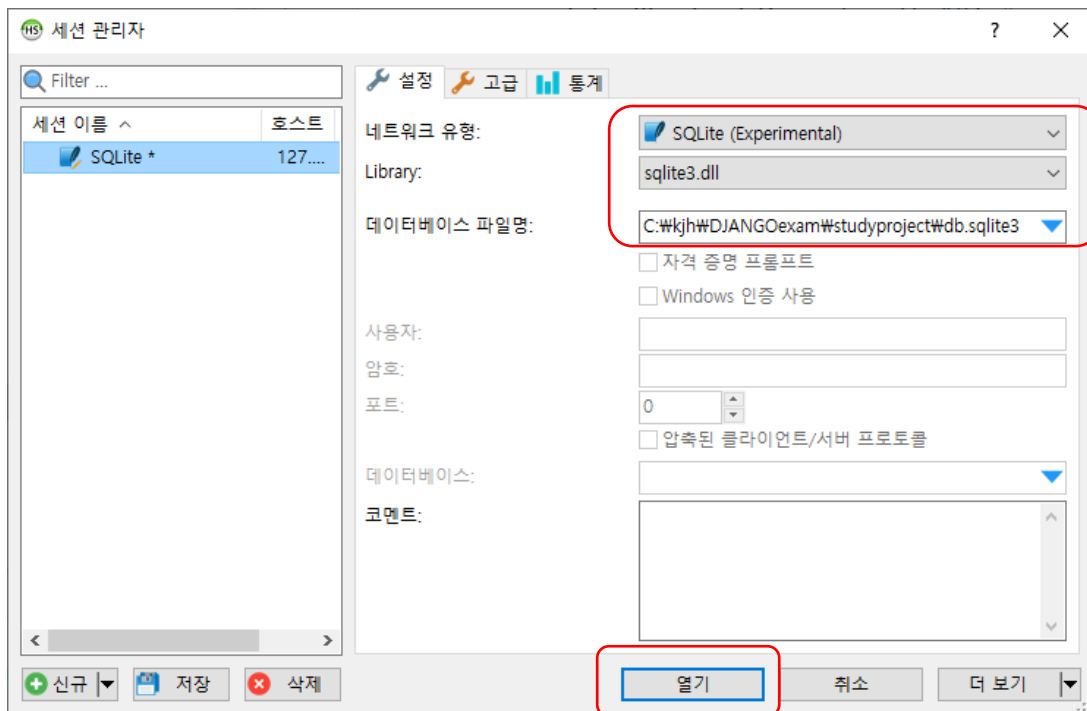


네트워크 유형에서 SQLite 항목을 선택하면 Library 에 sqlite3.dll 이 자동 설정되는 것을 볼 수 있다.

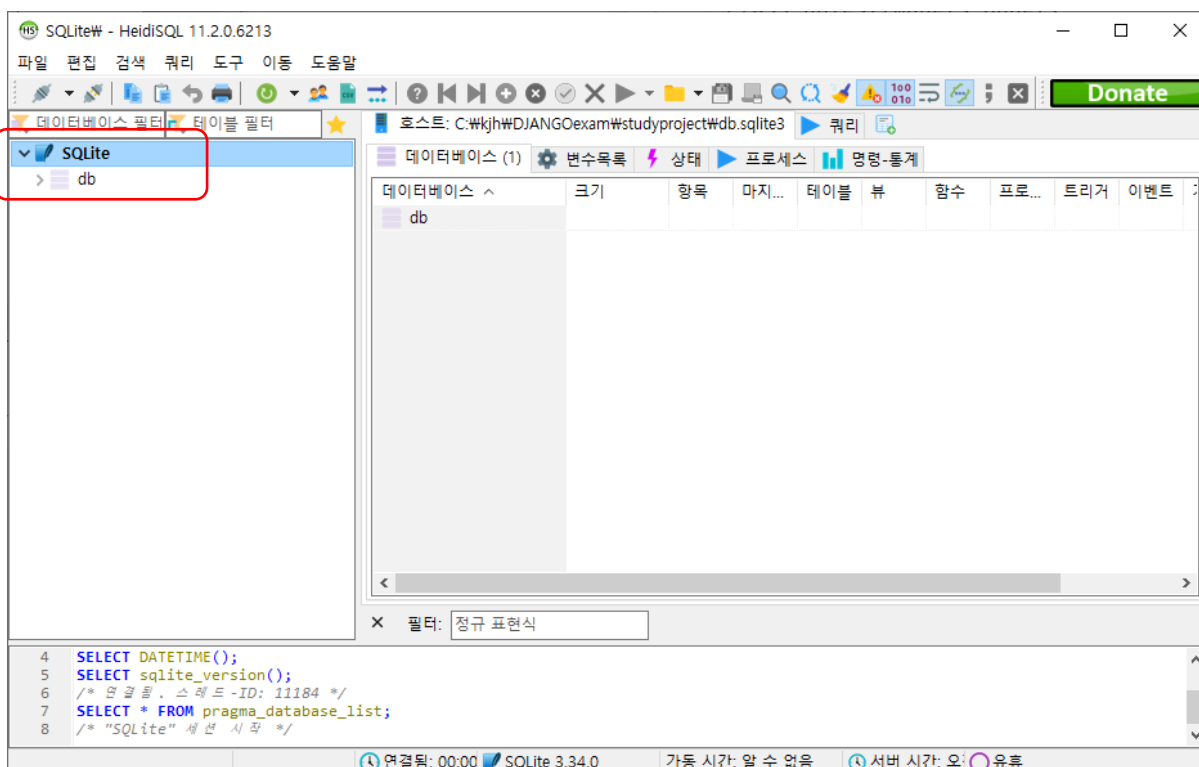
데이터베이스 파일명으로 studyproject 의 db.sqlite3 를 찾아서 설정한다.

HS 세션 관리자

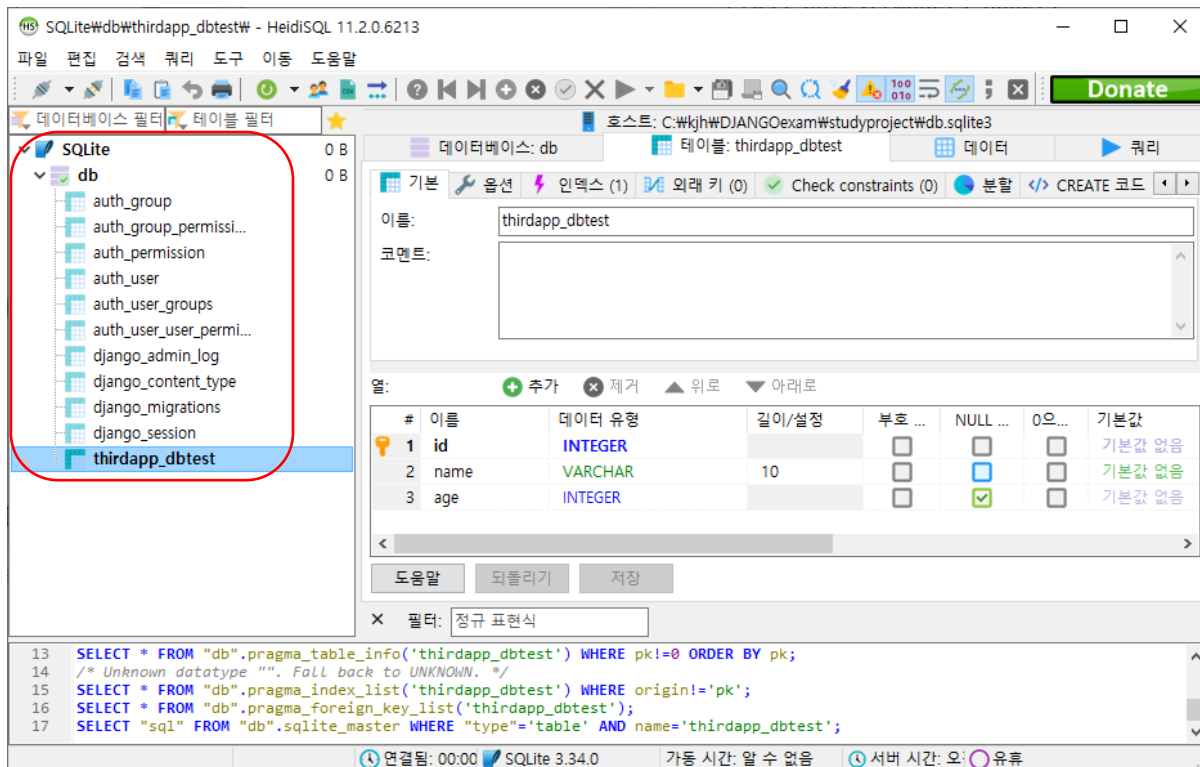




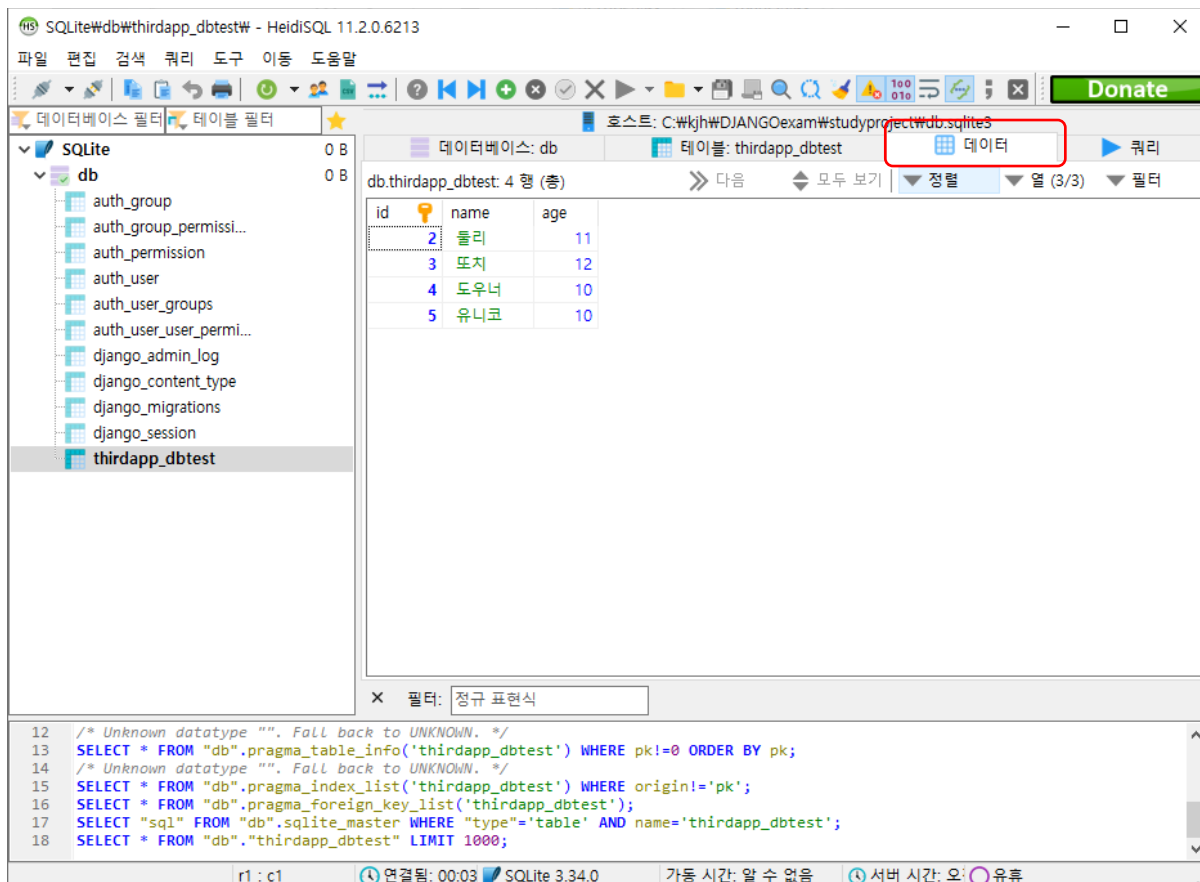
오픈된 studyproject 의 db.sqlite3 에는 db 라는 이름의 데이터베이스가 생성되어 있는 것을 볼 수 있다.



db 라는 이름의 데이터베이스에는 python manage.py migrate 명령의 실행에 의해서 생성된 테이블들이 여러 개 존재하며 그 중에서 제일 아래에 있는 thirdapp_dbtest 가 DBTest 라는 모델 클래스에 의해서 생성된 테이블이다. 오른쪽에서 이 테이블의 컬럼 사양을 체크해 볼 수 있다.



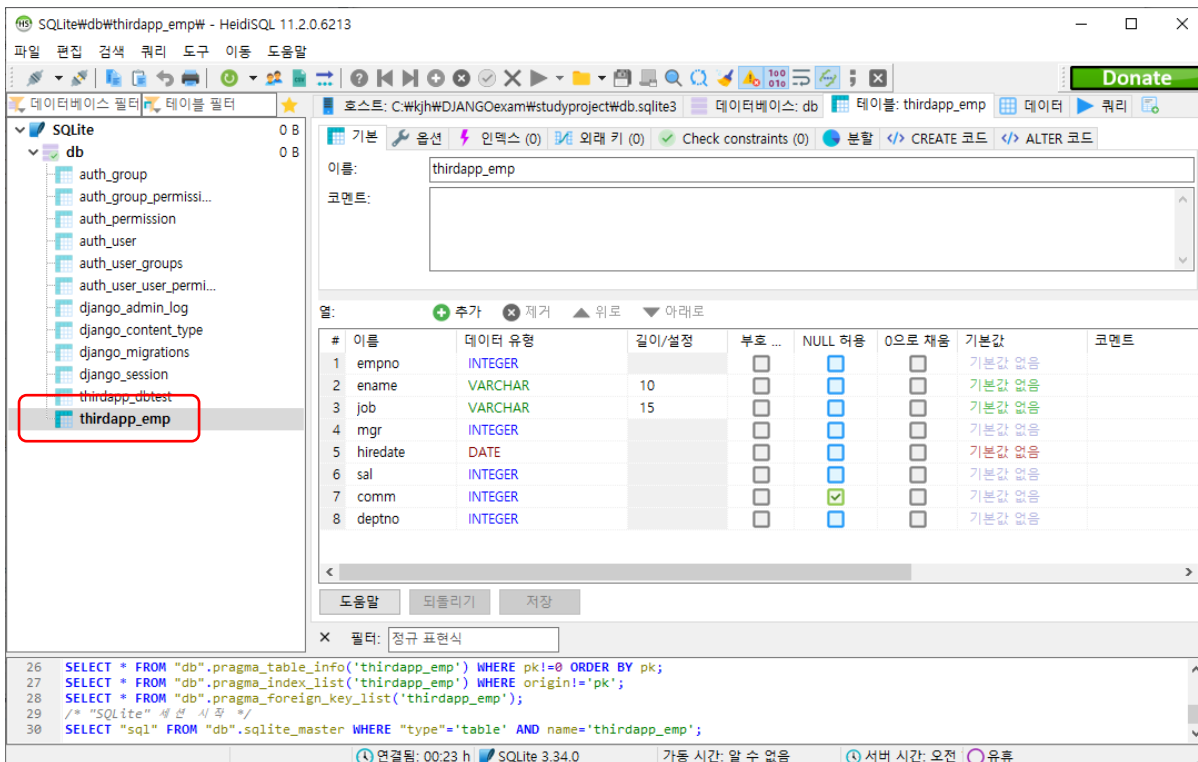
데이터라는 버튼을 클릭하면 thirdapp_dbtest 테이블에 저장된 데이터들을 직접 확인할 수 있다.



[illegible]

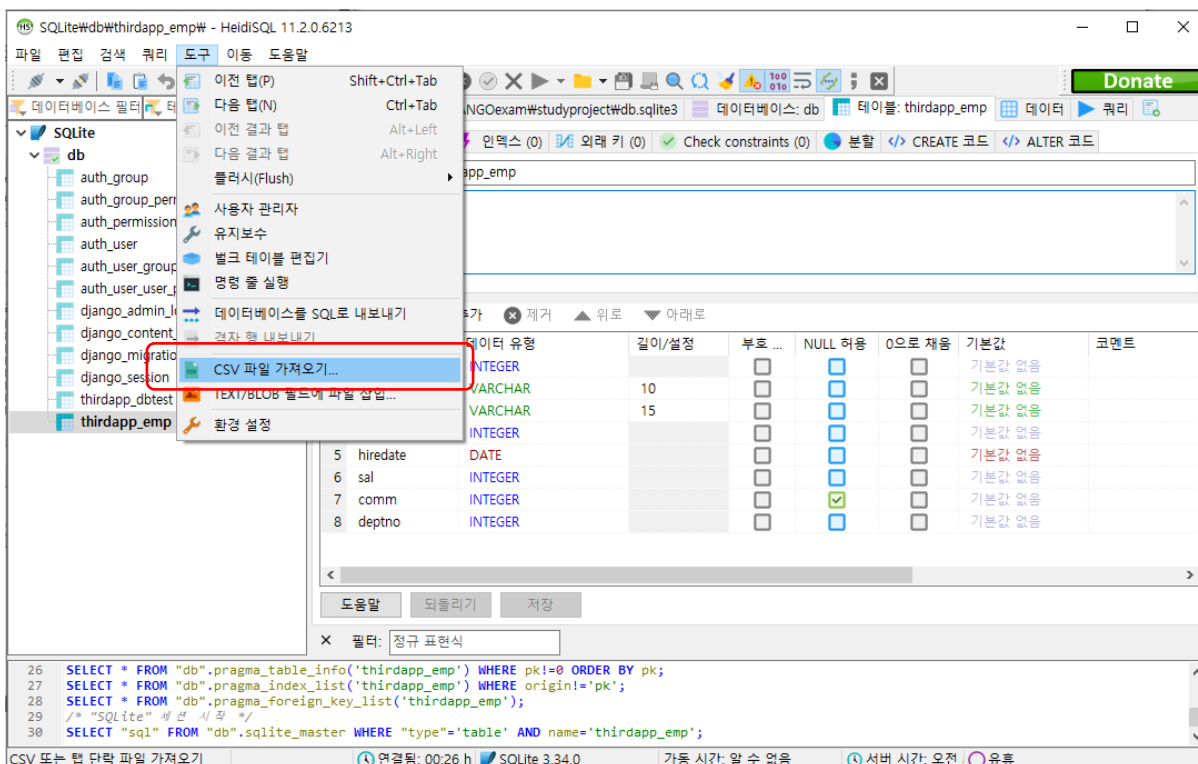
The screenshot shows the HeidiSQL interface with the 'thirdapp_emp' table structure displayed. The table has 8 columns: empno (INTEGER), ename (VARCHAR), job (VARCHAR), mgr (INTEGER), hiredate (DATE), sal (INTEGER), comm (INTEGER), and deptno (INTEGER). The 'comm' column has a checkmark in the 'NULL 허용' (Allow NULL) column. The 'thirdapp_emp' table is highlighted with a red box. The '저장' (Save) button is also highlighted with a red box.

다음과 같이 thirdapp_emp 라는 테이블이 생성된 것을 볼 수 있다.



생성된 테이블에 csv 파일의 내용을 저장해 보자.

제공된 emp.csv 를 적당한 폴더에 저장한 다음에 다음 과정을 수행한다.(c:\Temp)



문서 파일 가져오기

입력 파일

파일명:

인코딩: 자동 감지 (실패할 수 있음)

옵션

무시할 첫: 1 줄

☒ 낮은 우선 순위 (서버 부하 방지)

☐ 입력 파일에 로컬 형식의 숫자를 포함 (예시. 1.234,56)

☐ 가져오기 전, 대상 테이블 초기화

제어 문자

필드 종결자: ;

필드를 감싸는 구분자: " ☒ 선택

필드를 벗어나는 구분자: "

줄 종결자: \r\n

중복 행 처리

☒ INSERT (오류 발생 가능)

☐ INSERT IGNORE (중복)

☐ REPLACE (중복)

메서드

☐ 서버에서 파일 내용 분석 (데이터 불러오기)

☒ 클라이언트에서 파일 내용 분석

목적지

데이터베이스: db

테이블: thirdapp_emp

열:

☒ empno

☒ ename

☒ job

☒ mgr

☒ hiredate

☒ sal

☒ comm

☒ deptno

☒ 모두

가져오기!

취소

열기

찾는 위치(I): Temp

이름	수정된 날짜	유형
day11	2021-01-19 오전 9:21	파일 폴더
exerciseproject	2021-01-29 오전 8:16	파일 폴더
log	2021-01-14 오전 10:21	파일 폴더
studyproject	2021-01-29 오전 8:16	파일 폴더
emp.csv	2017-01-03 오후 3:54	Microsoft

파일 이름(N): emp.csv

파일 형식(T): CSV 파일 (*.csv)

인코딩: UTF-8

열기(O)

취소

문서 파일 가져오기

입력 파일

파일명: C:\Temp\emp.csv

인코딩: UTF-8

옵션

무시할 첫 1 줄

☒ 낮은 우선 순위 (서버 부하 방지)

☐ 입력 파일에 로컬 형식의 숫자를 포함 (예시. 1.234,56)

☐ 가져오기 전, 대상 테이블 초기화

중복 행 처리

☒ INSERT (오류 발생 가능)

☐ INSERT IGNORE (중복)

☐ REPLACE (중복)

메서드

☐ 서버에서 파일 내용 분석 (데이터 불러오기)

☒ 클라이언트에서 파일 내용 분석

제어 문자

필드 종결자 ;

필드를 감싸는 구분자 "

필드를 벗어나는 구분자 "

줄 종결자 \r\n

목적지

데이터베이스: db

테이블: thirdapp_emp

열:

☒ empno

☒ ename

☒ job

☒ mgr

☒ hiredate

☒ sal

☒ comm

☒ deptno

가져오기! 취소

emp.csv의 경우 데이터
구분자가 , 이므로 ; 을 ,
로 변경한다.

emp.csv 파일의 내용이 thirdapp_emp 테이블에 저장된 것을 볼 수 있다.

SQLite#db#thirdapp_emp# - HeidiSQL 11.2.0.6213

파일 편집 검색 쿼리 도구 이동 도움말

데이터베이스 필터 테이블 필터

호스트: C:\kjh\DJANGOexam\studyproject\db.sqlite3 데이터베이스: db 테이블: thirdapp_emp 데이터 쿼리

db.thirdapp_emp: 14 행 (총)

empno	ename	job	mgr	hiredate	sal	comm	deptno
7,369	SMITH	CLERK	7,902	1980-12-17	800		20
7,499	ALLEN	SALESMAN	7,698	1981-02-20	1,600	300	30
7,521	WARD	SALESMAN	7,698	1981-02-03	1,250	500	30
7,566	JONES	MANAGER	7,839	1981-03-02	2,975		20
7,654	MARTIN	SALESMAN	7,698	1981-10-22	1,250	1,400	30
7,698	BLAKE	MANAGER	7,839	1981-05-01	2,850		30
7,782	CLARK	MANAGER	7,839	1981-09-06	2,450		10
7,788	SCOTT	ANALYST	7,566	1982-12-08	3,000		20
7,839	KING	PRESIDENT		1981-11-17	5,000		10
7,844	TURNER	SALESMAN	7,698	1984-10-08	1,500		30
7,876	ADAMS	CLERK	7,788	1983-01-12	1,100		20
7,900	JAMES	CLERK	7,698	1981-12-03	950		30
7,902	FORD	ANALYST	7,566	1981-12-13	3,000		20
7,934	MILLER	CLERK	7,782	1982-01-25	1,300		10

필터: 정규 표현식

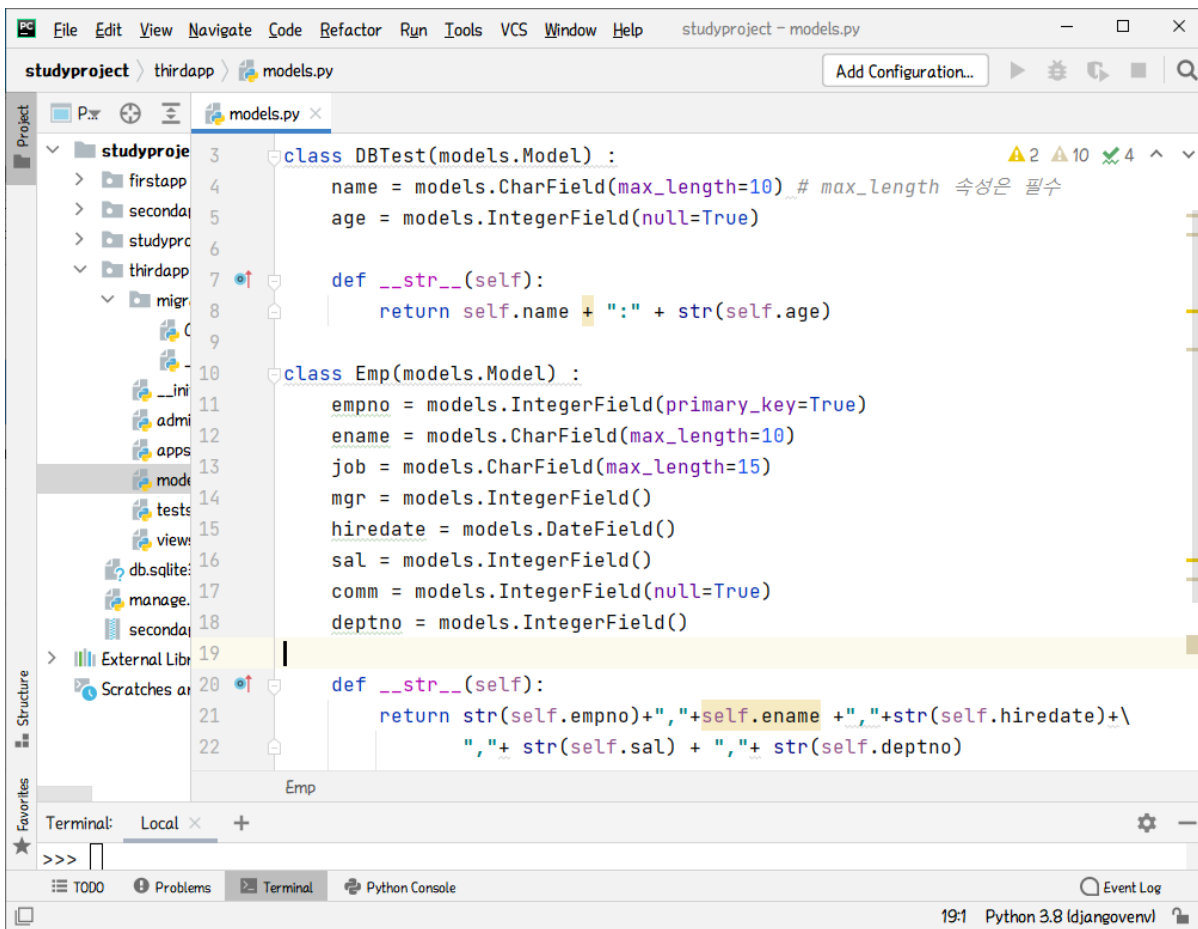
```

28 SELECT * FROM "db"."pragma_foreign_key_list('thirdapp_emp');
29 /* "SQLite" 쿼리 시작 */
30 SELECT "sql" FROM "db"."sqlite_master" WHERE "type"='table' AND name='thirdapp_emp';
31 /* 14 rows imported in 0.078 seconds. */
32 SELECT * FROM "db"."thirdapp_emp" LIMIT 1000;

```

r1 : c1 연결됨: 00:29 h SQLite 3.34.0 가동 시간: 알 수 없음 서버 시간: 오전 유틸

thirdapp_emp 테이블의 내용을 파이썬 프로그램에서 다루기 위해 thirdapp 의 models.py 파일에 다음과 같이 Emp 라는 클래스를 생성한다.



```
3 class DBTest(models.Model) :
4     name = models.CharField(max_length=10) # max_length 속성은 필수
5     age = models.IntegerField(null=True)
6
7     def __str__(self):
8         return self.name + ":" + str(self.age)
9
10 class Emp(models.Model) :
11     empno = models.IntegerField(primary_key=True)
12     ename = models.CharField(max_length=10)
13     job = models.CharField(max_length=15)
14     mgr = models.IntegerField()
15     hiredate = models.DateField()
16     sal = models.IntegerField()
17     comm = models.IntegerField(null=True)
18     deptno = models.IntegerField()
19
20     def __str__(self):
21         return str(self.empno)+","+self.ename+","+str(self.hiredate)+\
22             ","+ str(self.sal) + ","+ str(self.deptno)
```

djangoenv 가상환경 기반의 터미널에서 `python manage.py shell` 명령을 실행시키고 인터랙티브 실행모드를 기동시킨 후에 다음 명령들을 실행시켜 본다.

```
from thirdapp.models import Emp
Emp.objects.all()
```



```
studyproject > thirdapp > models.py
16 sal = models.IntegerField()
17 comm = models.IntegerField(null=True)
18 deptno = models.IntegerField()
19
20 def __str__(self):
21     return str(self.empno)+","+self.ename +","+str(self.hiredate)+\
22         ", "+ str(self.sal) + ", "+ str(self.deptno)

Emp > __str__

Terminal: Local x +
(djangoven) C:\kjh\DJANGOexam\studyproject>python manage.py shell
Python 3.8.7 (tags/v3.8.7:6503f05, Dec 21 2020, 17:59:51) [MSC v.1928 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
(InteractiveConsole)
>>> from thirdapp.models import Emp
>>> Emp.objects.all()
<QuerySet [(<Emp: 7369,SMITH,1980-12-17,800,20>, <Emp: 7499,ALLEN,1981-02-20,1600,30>, <Emp: 7521,WARD,1981-02-03,1250,30>, <Emp: 7566,JONES,1981-03-02,2975,20>, <Emp: 7654,MARTIN,1981-10-22,1250,30>, <Emp: 7698,BLAKE,1981-05-01,2850,30>, <Emp: 7782,CLARK,1981-09-06,2450,10>, <Emp: 7788,SCOTT,1982-12-08,3000,20>, <Emp: 7839,KING,1981-11-17,5000,10>, <Emp: 7844,TURNER,1984-10-08,1500,30>, <Emp: 7876,ADAMS,1983-01-12,1100,20>, <Emp: 7900,JAMES,1981-12-03,950,30>, <Emp: 7902,FORD,1981-12-13,3000,20>, <Emp: 7934,MILLER,1982-01-25,1300,10>]>
>>>
```

```
>>> for d in Emp.objects.all() :
...     print(d)
```

```
...
7369,SMITH,1980-12-17,800,20
7499,ALLEN,1981-02-20,1600,30
7521,WARD,1981-02-03,1250,30
7566,JONES,1981-03-02,2975,20
7654,MARTIN,1981-10-22,1250,30
7698,BLAKE,1981-05-01,2850,30
7782,CLARK,1981-09-06,2450,10
7788,SCOTT,1982-12-08,3000,20
7839,KING,1981-11-17,5000,10
7844,TURNER,1984-10-08,1500,30
7876,ADAMS,1983-01-12,1100,20
7900,JAMES,1981-12-03,950,30
7902,FORD,1981-12-13,3000,20
7934,MILLER,1982-01-25,1300,10
```

```
>>> for d in Emp.objects.order_by('hiredate') :  
...     print(d)
```

```
...  
7369, SMITH, 1980-12-17, 800, 20  
7521, WARD, 1981-02-03, 1250, 30  
7499, ALLEN, 1981-02-20, 1600, 30  
7566, JONES, 1981-03-02, 2975, 20  
7698, BLAKE, 1981-05-01, 2850, 30  
7782, CLARK, 1981-09-06, 2450, 10  
7654, MARTIN, 1981-10-22, 1250, 30  
7839, KING, 1981-11-17, 5000, 10  
7900, JAMES, 1981-12-03, 950, 30  
7902, FORD, 1981-12-13, 3000, 20  
7934, MILLER, 1982-01-25, 1300, 10  
7788, SCOTT, 1982-12-08, 3000, 20  
7876, ADAMS, 1983-01-12, 1100, 20  
7844, TURNER, 1984-10-08, 1500, 30
```

```
>>> for d in Emp.objects.order_by('-sal') :  
...     print(d)
```

```
...  
7839, KING, 1981-11-17, 5000, 10  
7788, SCOTT, 1982-12-08, 3000, 20  
7902, FORD, 1981-12-13, 3000, 20  
7566, JONES, 1981-03-02, 2975, 20  
7698, BLAKE, 1981-05-01, 2850, 30  
7782, CLARK, 1981-09-06, 2450, 10  
7499, ALLEN, 1981-02-20, 1600, 30  
7844, TURNER, 1984-10-08, 1500, 30  
7934, MILLER, 1982-01-25, 1300, 10  
7521, WARD, 1981-02-03, 1250, 30  
7654, MARTIN, 1981-10-22, 1250, 30  
7876, ADAMS, 1983-01-12, 1100, 20  
7900, JAMES, 1981-12-03, 950, 30  
7369, SMITH, 1980-12-17, 800, 20
```

[제공된 forthapp 과 visitorapp 을 등록하기]

(1) settings.py 소스에 다음 내용을 추가한다.

```
INSTALLED_APPS = [  
    'secondapp',  
    'thirdapp',  
    'forthapp',  
    'visitorapp',  
    'accountapp',  
]
```

(2) 두 개의 앱은 makemigrations 를 수행하여 migrate 하는데 사용되는 소스를 생성한다.

```
python manage.py makemigrations forthapp visitorapp
```

(3) 다음 명령을 실행하여 DB 테이블을 생성한다.

```
python manage.py migrate
```

(4) 메인 urls.py 파일에 다음의 패스 정보를 추가한다.

```
path('forthapp/', include('forthapp.urls')),  
path('visitorapp/', include('visitorapp.urls')),
```