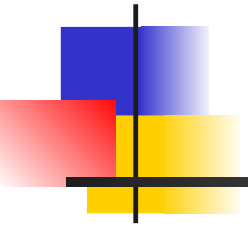
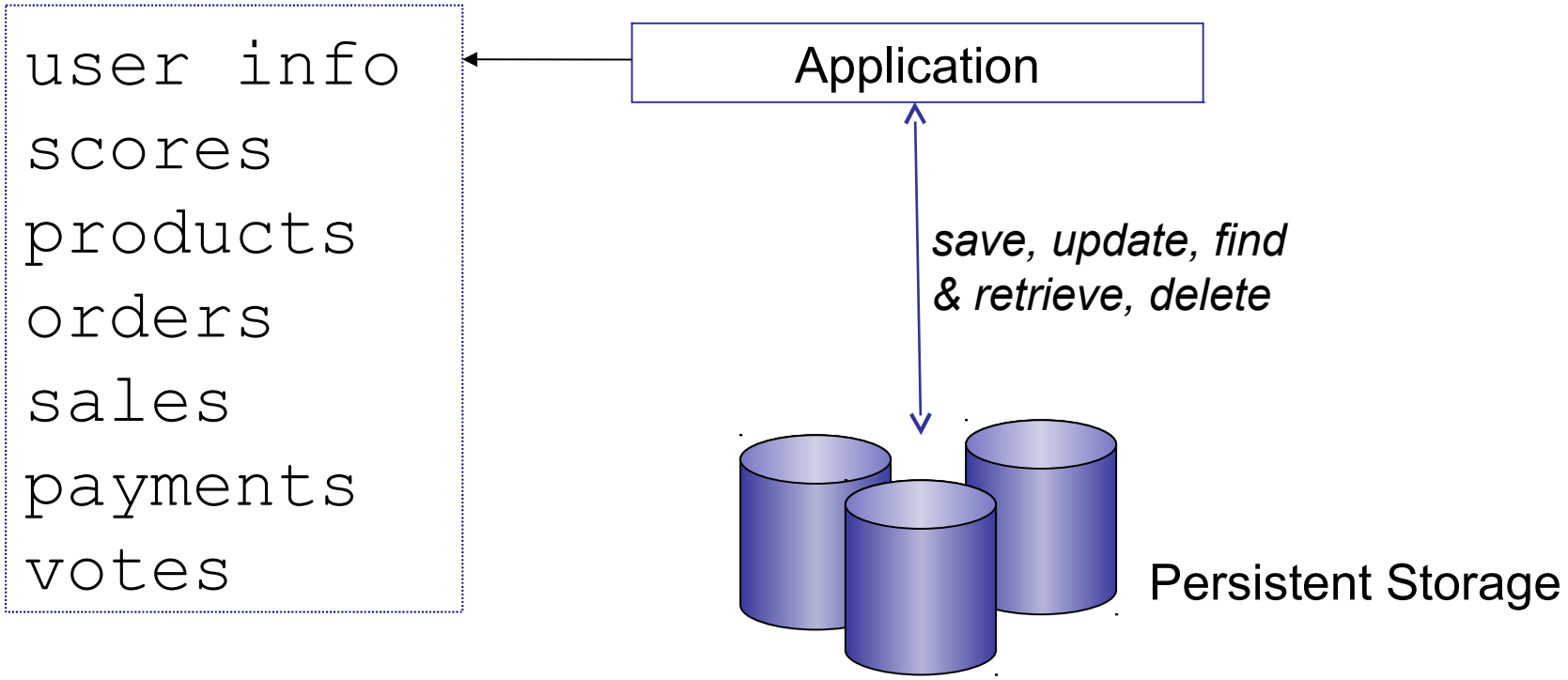


Persistence and Object-Relational Mapping

James Brucker

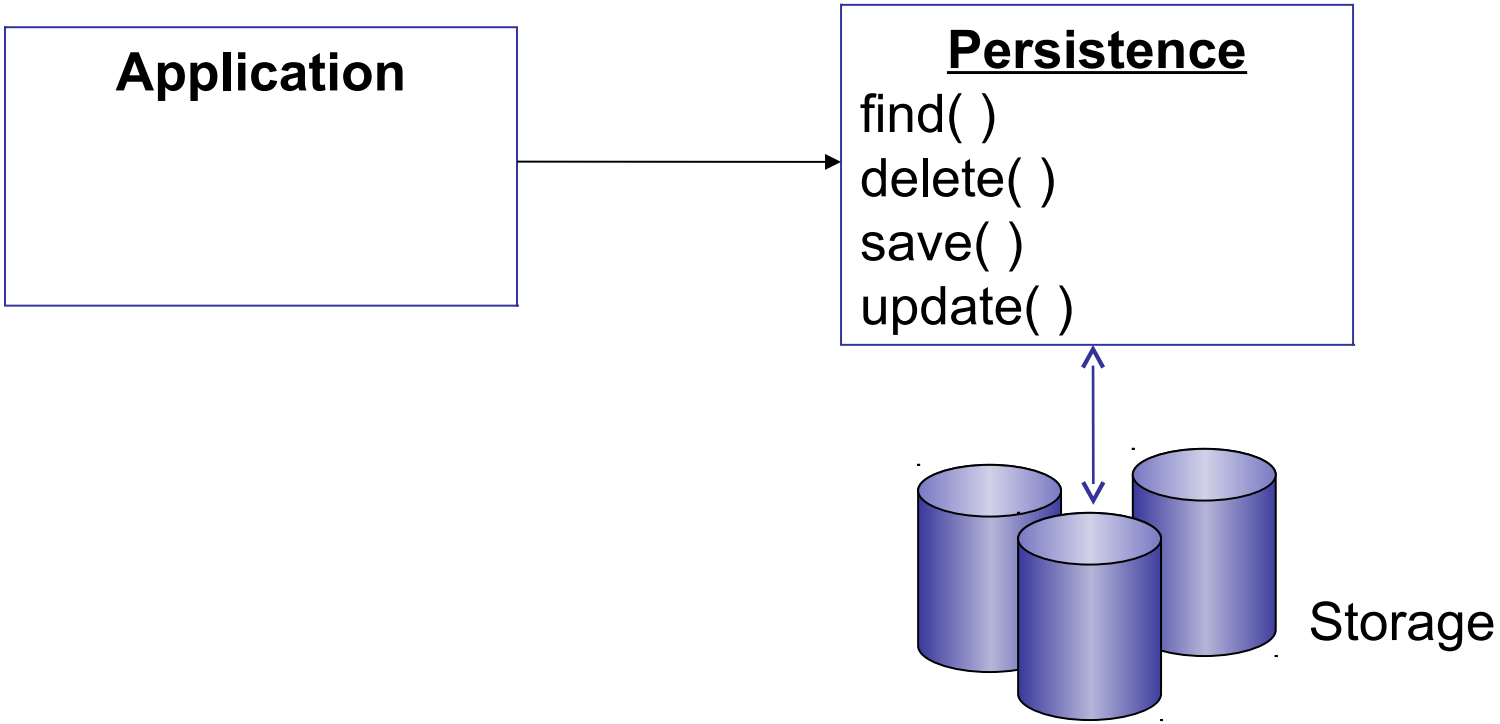
Goal

- Applications need to **save** data to ***persistent storage***.
- Persistent storage can be database, directory service, plain files, spreadsheet, cloud service, ...



Abstraction - just do it

- We want to ***abstract*** (hide) details of how data is being saved and retrieved.
- The application only knows *what* it wants done (save, retrieve, update), not *how*.



Terminology

Persistence - *prolonged existence of something.*

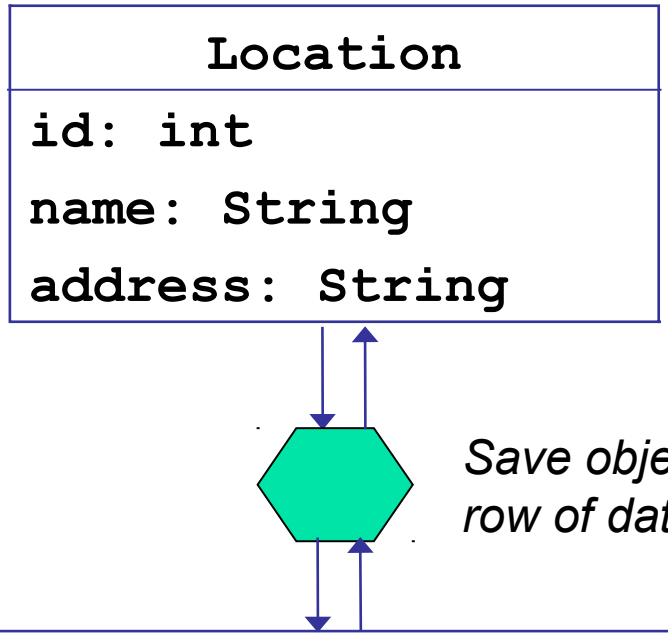
In software, persistence refers to preserving the existence of data after program stops.

Entity - *something with a distinct, independent existence.*

Software entity: an object that can exist (persist) from one program execution to the next.

Saving & Recreating Objects

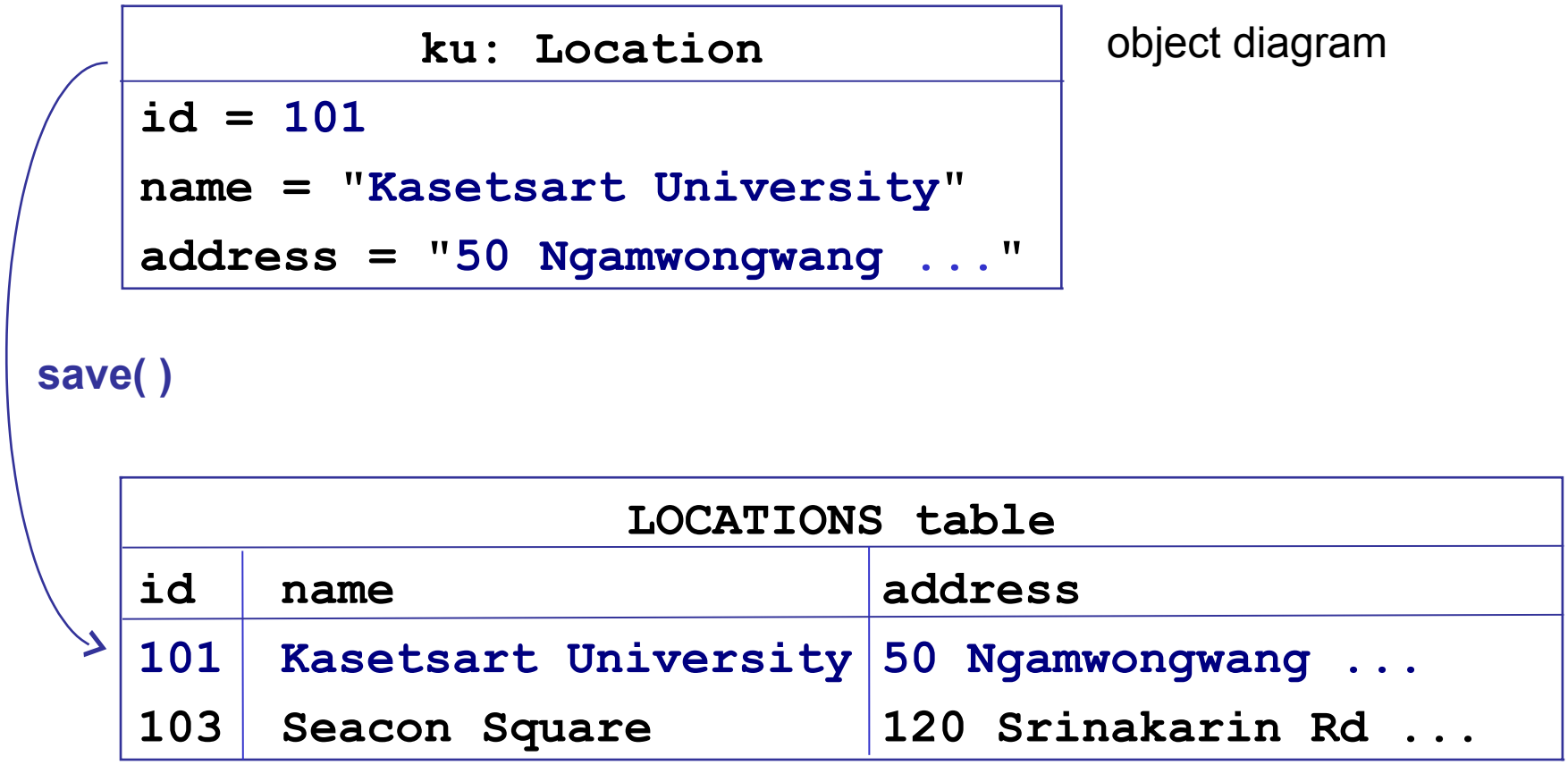
An object's attributes are similar to the fields in a table.



Save object as row in a table, retrieve row of data and (re)create an object

LOCATIONS table		
id (PK)	name	address
101	Kasetsart	50 Ngamwongwang Rd, ...
102	Pizza Hut	44 Pahonyotin Rd, Jatujak,..

Mapping an Object



Object-Relational Mapping

Purpose

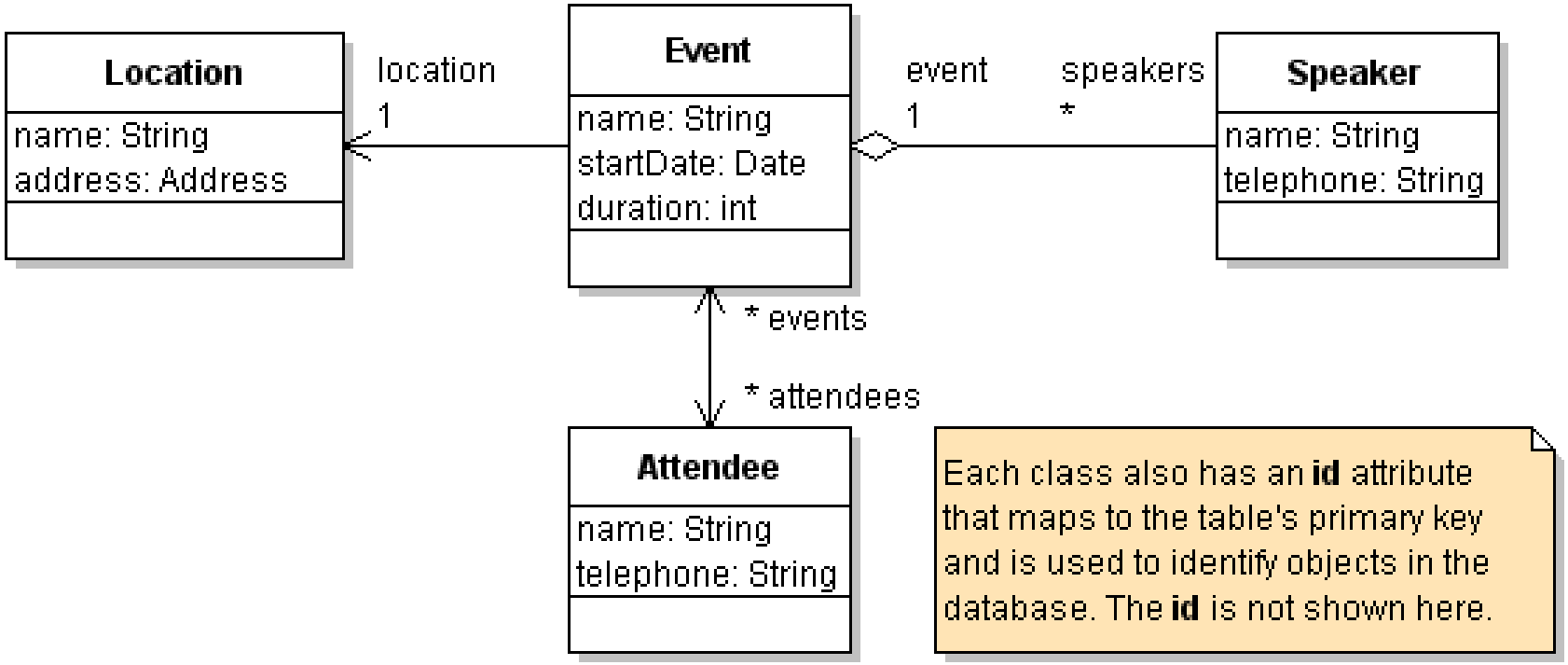
- save an object to a database table(s)
- recreate object(s) using data from a database
- save and recreate *associations* between objects

Design Goals

- **separate** the O-R mapping service from our application
- **abstract details** of how its done -- app just calls save()
- **preserve identity** - don't create 2 copies of same object
- **localize** the impact of change in the database.

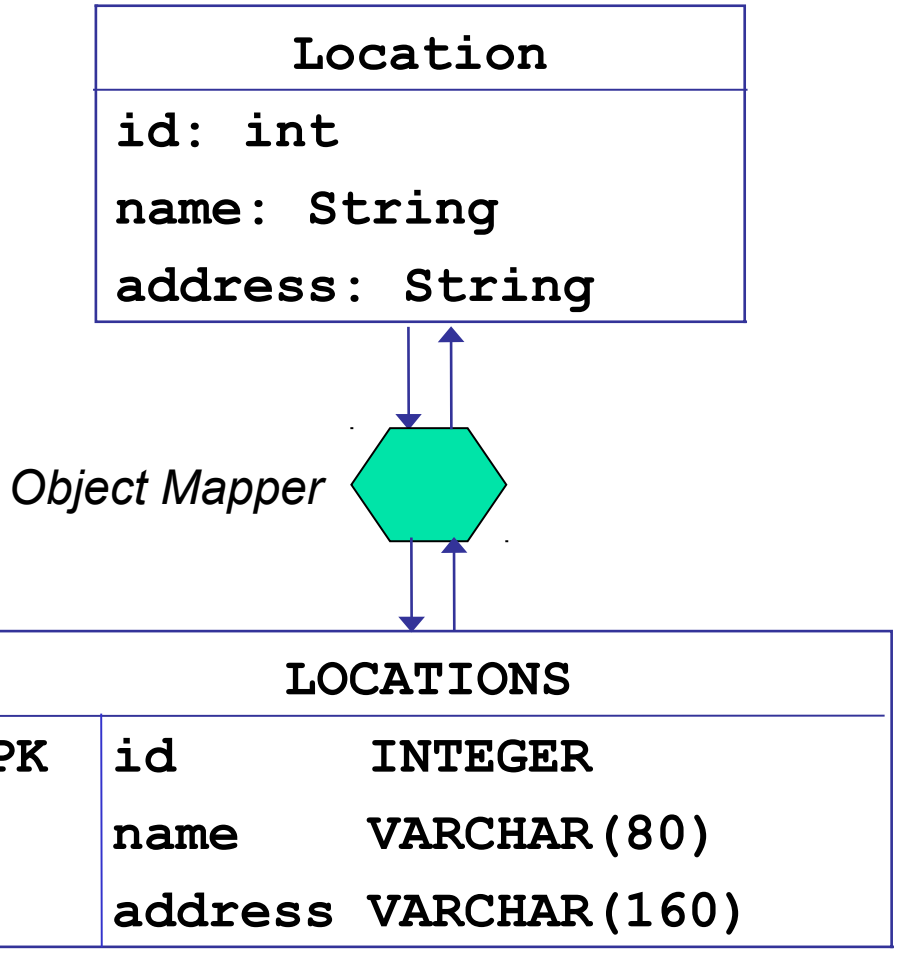
An Example

An Event Manager application with these classes:



Object-Relational Mapping details

Each *entity* class needs an id field that is PK in table.



Class

*should have an **identifier** attribute*

Object Mapper

save objects to rows in tables, retains uniqueness

Database Table

identifier is usually the primary key of table

Code for ORM

```
ku = Location( "Kasetsart University",
               "50 Ngamwongwang Road, Bangkok" );
# save it!
object_mapper.save( ku )
# object_mapper assigns an id to object
print( ku.id )
101
```

Issues:

- mapper should choose a unique ID for each saved object
- what if same data (Kasetsart University) is already in the table?

Finding and Retrieving an Object

```
# find by id (only one match possible)
ku1 = object_mapper.find(id=101)
# find by name (may have many matches)
list = object_mapper.find(name="Kasetsart University")
```

Does object_mapper always return the same object?

```
ku1 = object_mapper.find(id=101)
ku2 = object_mapper.find(id=101)
```

ku1 == ku2 => true or false?

Essential Operations: CRUD

Most common persistence operations are:

Create save a new object to the database

Retrieve an object (or objects) from the database

Update data for an object already in database

Delete object data from the database

Which one is most *Complex*?

Of the 4 CRUD operations, which do you think is the most complex to provide?

Create save a new object to the database

Retrieve an object from the database

Update data for an object already in database

Delete object data from the database

Providing CRUD

Simple:

Create save(object)

Update update(object) or save(object)

Delete delete(object)

Complex:

Retrieve one object by id = get(id)

Retrieve all objects

Retrieve using a query expression:
address contains "Bangkok" or population > 1000000

Retrieve first 10 objects, sorted by date

Try ORM in Django

```
cmd> python manage.py shell
>>> from polls.models import Question
>>> q = Question(question_text="Understand ORM?")
>>> q.pub_date = datetime.now()
>>> q.id
(nothing is printed)
>>> q.save( )
>>> q.id
6
>>> Question.objects.all( )
<QuerySet: [..., <Question: Understand ORM?>,...
```

Try it in Django

```
# Change something, then update data in database
>>> q.question_text = "Next question?"
>>> q.save()
# Did it update the question in database?
>>> Question.objects.get( id=6 )
<Question: Next Question?>
# Can we delete it from database?
>>> q.delete( )
>>> Question.objects.get( id=6 )
DoesNotExist: Question matching query does not
              exist.
```

Design of a Persistence Service

2 Design Patterns for a persistence service

Data Access Objects - define a **separate class** that is responsible for persistence services.
Your app calls the DAO class to save/retrieve objects.

Active Object Pattern - entity classes perform CRUD operations themselves.

- Behavior is defined in a ***superclass***.
- Each entity is a **subclass** and *inherits* the CRUD operations, so no new code is needed.

Which Design does Django Use?

Data Access Objects - define a **separate class** that is responsible for saving & recreating objects.
Your app calls the DAO class to save/retrieve objects.

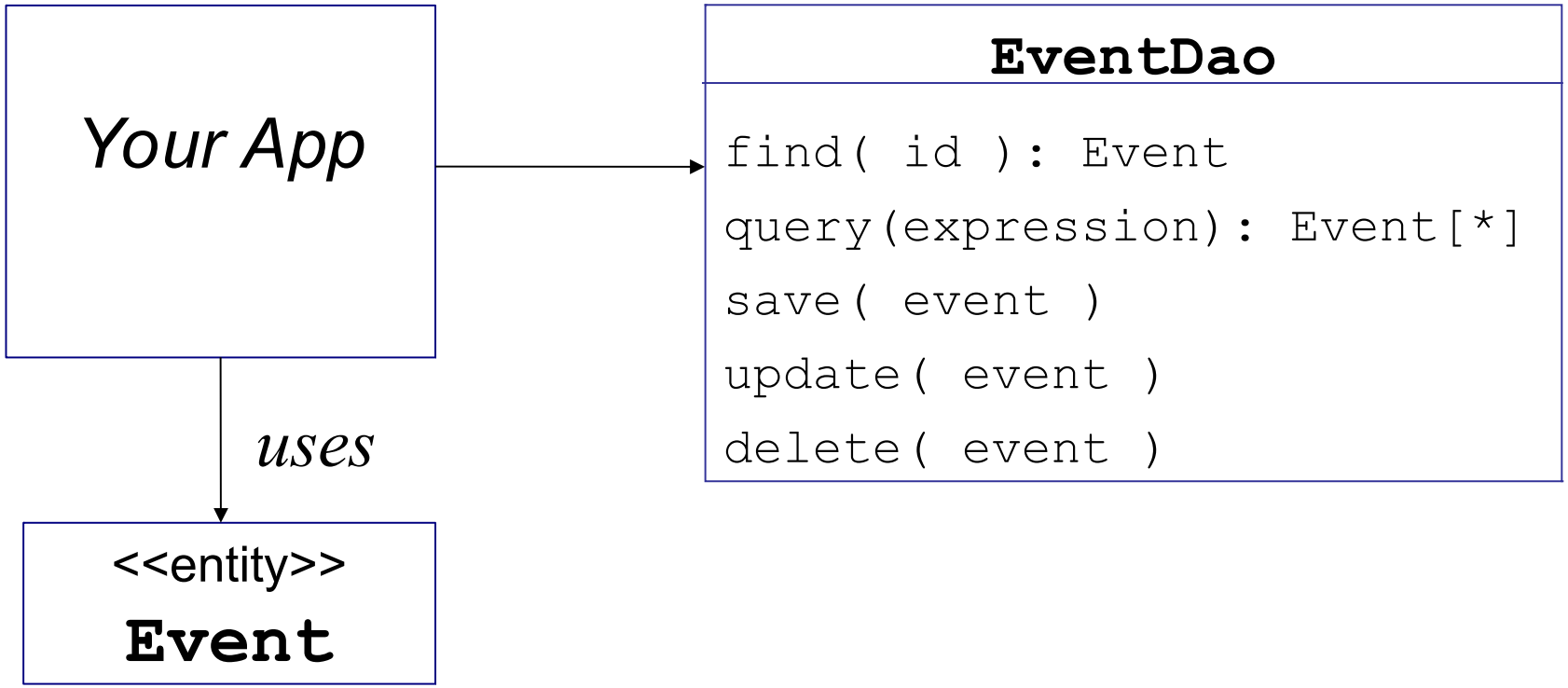
Active Object Pattern - entity classes perform CRUD operations themselves.

- Behavior is defined in a ***superclass***.
- Each entity is a **subclass** and *inherits* the CRUD operations, so no new code is needed.

Data Access Object Pattern

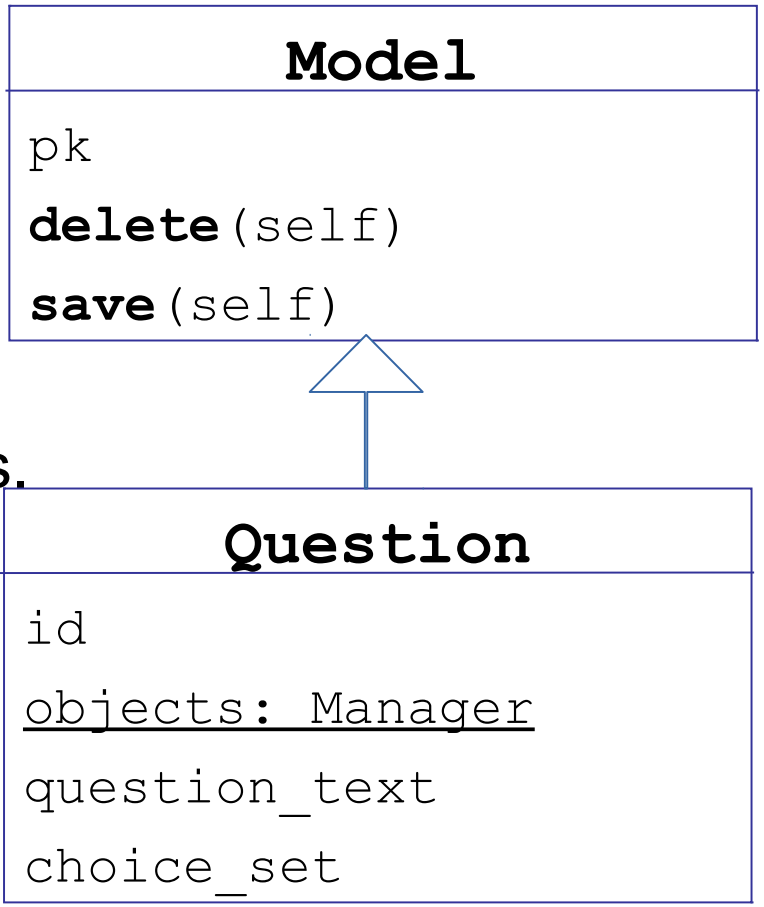
A separate class provides persistence services.

- Append "Dao" to the class name, e.g. **`EventDao`**.



Active Object Pattern

- A super-class provides persistence operations.
- Entity classes are subclasses & inherit the behavior.
- Entity saves itself.



- Django *automatically* adds `id` and `objects` attributes.

What does the underline mean?

SQL Data Types

Each field in a database table has a fixed data type.

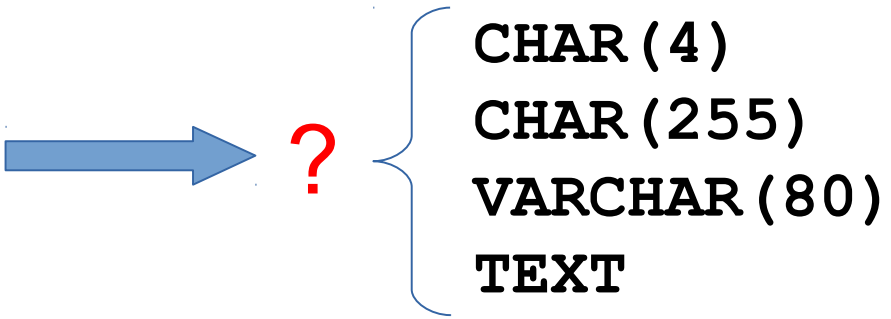
But SQL data types are not the same as Python or Java data types.

CHAR(20), CHARACTER(20)	fixed length string
VARCHAR(200)	variable length string
BOOLEAN	0 = false, x = true
SMALLINT	2-byte integer
INT	4-byte integer
FLOAT	8-byte floating pt (double)
DECIMAL(n,p)	stored in decimal (base 10) format

Mapping Data Types

Ambiguity in converting data type from Python (or Java) to SQL data type.

Example: how to save a Python **`str`** variable?



Mapping dates and times is even more ambiguous!

Django: programmer must specify

Persistent fields in model classes must use model data types. Field sizes can be specified or use default size.

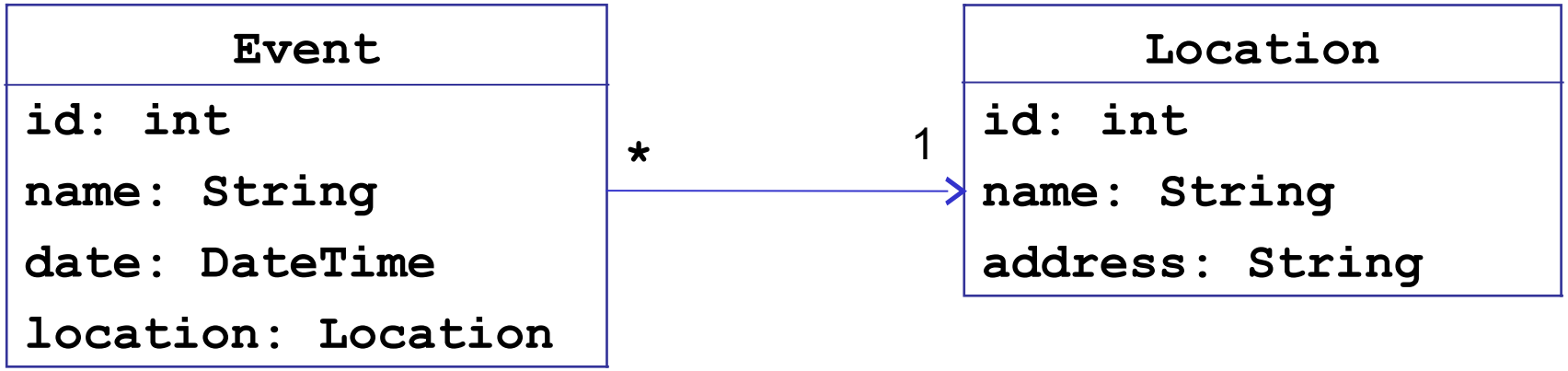
```
class Person(django.db.models.Model):
    name =      models.CharField(max_length=80)
    birthday = models.DateField(auto_now=True)
    email =     models.EmailField(max_length=254)
    thai_id =  models.IntegerField(max_length=13,
                                   unique=True)


class BankAccount(django.db.models.Model):
    balance = models.DecimalField(decimal_places=2)
    owner =    models.ForeignKey('Person')
```

How to Save Associations?

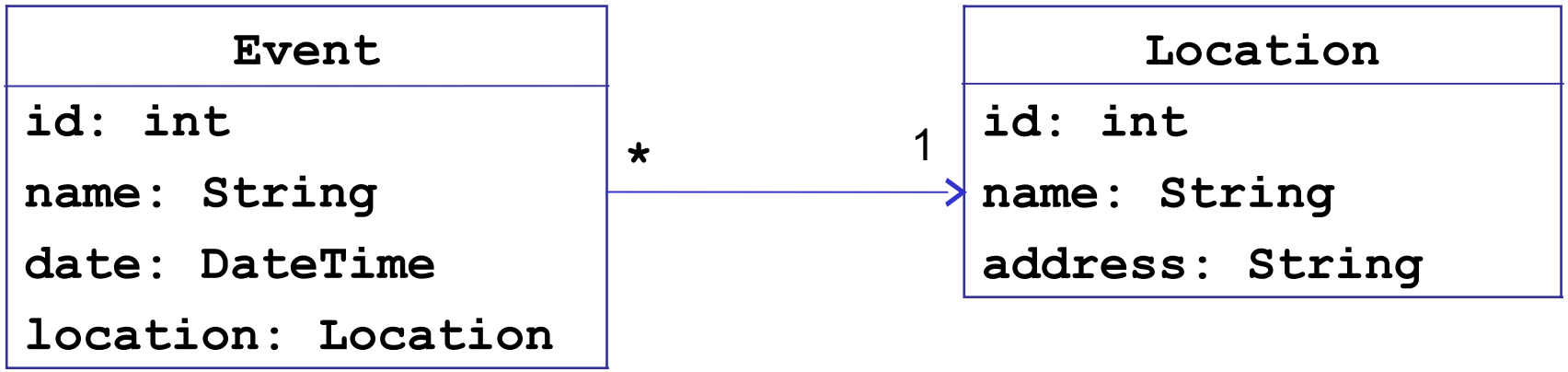
Objects have associations (references) to other objects.

How can we save associations?

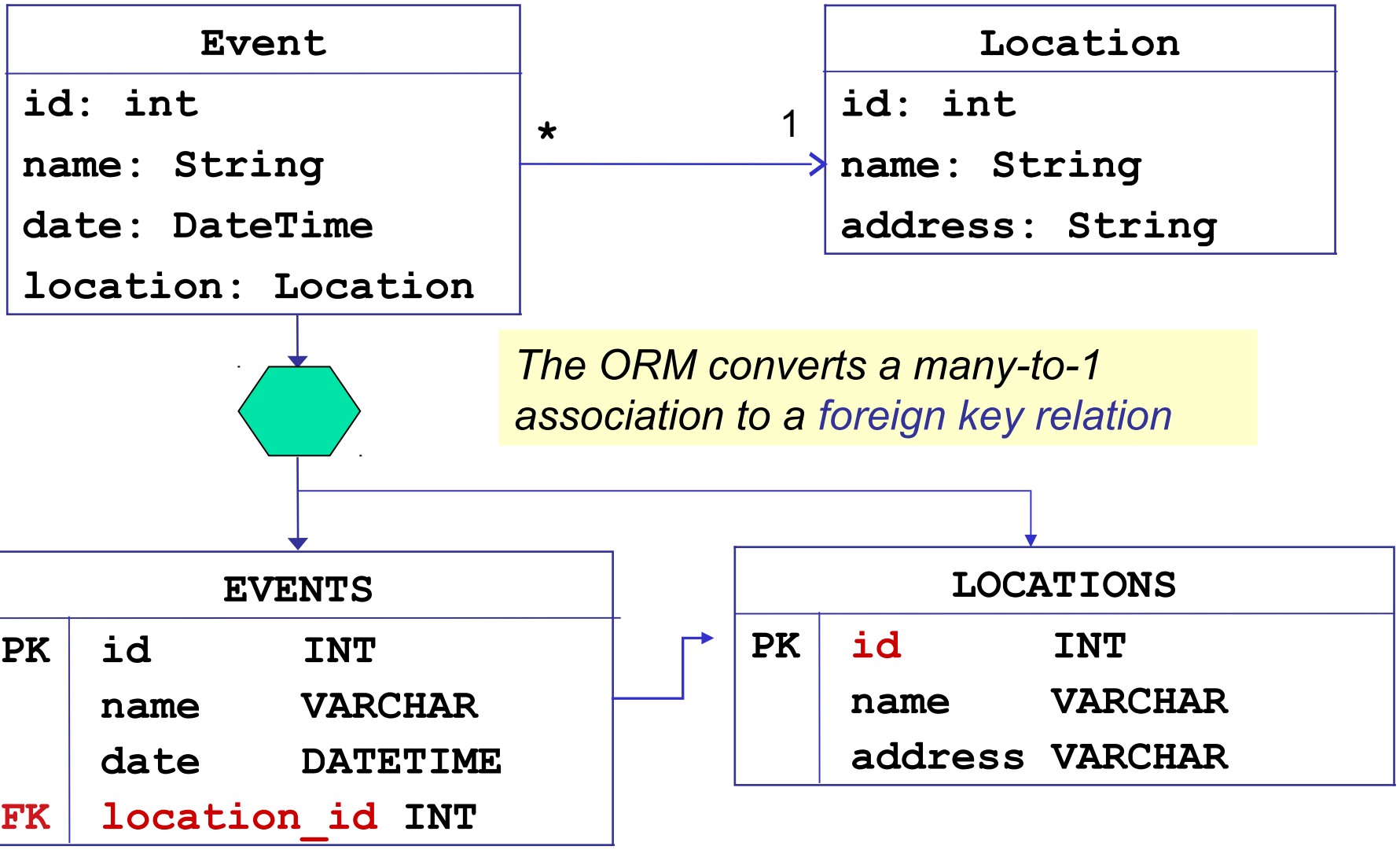
An Event has a Location:



O-R Mapping of n-to-1 Associations



O-R Mapping of n-to-1 Associations



n-to-1 association in Django

You specify only the related class (`Location`),

not the name of field in the database.

```
class Event(models.Model):
    name = models.CharField('name',max_length=80)
    date = models.DateTimeField('date')
    location = models.ForeignKey(Location)
```

Save What?

```
event = Event( "BarCamp 2019" )
ku = Location( "Kasetsart University", "..." )
# Yeah! Bar Camp is coming to KU!
event.set_location( ku )
event.set_date( datetime.date(2019, 11, 25) )
# save the event
object_mapper.save( event )
```

Did object mapper save the location, too?

Or do we have to save location ourselves?

Fetching an Event

```
# Retrieve the event
event2 = object_mapper.find( name="BarCamp 2019" )
# object mapper finds the event...
print( event2.name )
"BarCamp 2019"
# did it recreate the location, too?
print( event2.location.name )
???
```

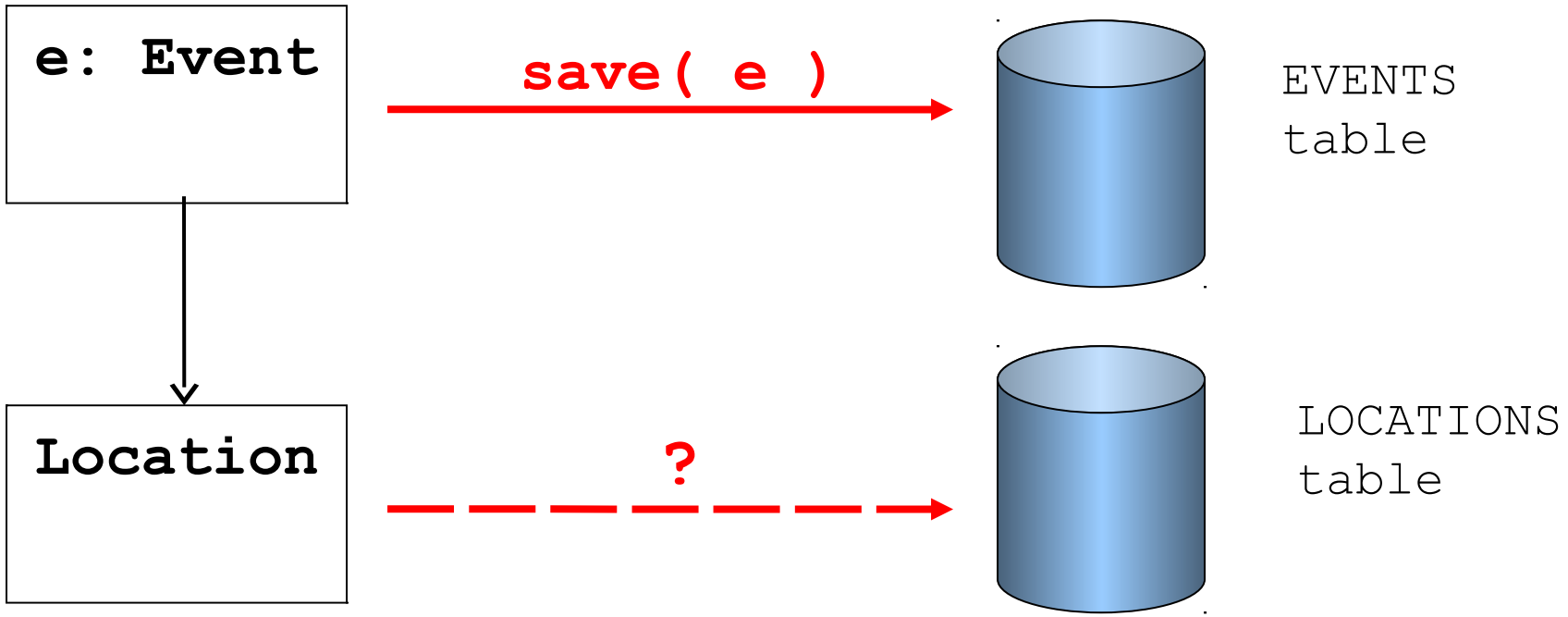
When we retrieve an event,

does the ORM retrieve the location object, too?

Cascading

When you save, update, delete an object in database...

are associated objects also saved/updated/deleted?



Cascading

Cascading means that an operation on one object should propagate (or cascade) to related objects.

Cascade = true: when you save an Event, save its Location, too (if necessary).

Cascade = false: when you save an Event, don't save its Location. Programming should save Location *first* so that Location has an id.

Frameworks Provide Cascading

In JPA, using annotations:

```
@Entity
class Event {

  @OneToMany(mappedBy="event", cascade=PERSIST)
  private List<Person> attendees;
```

NONE
PERSIST
REFRESH
REMOVE
ALL

Does Django do cascading save?

Try it with the polls app:

```
>>> c1 = Choice(choice_text="First Choice")
>>> q = Question(question_text="What's your choice?")
>>> q.choice_set.add( c1 )
TraceBack...
ValueError: <Choice: First Choice> isn't saved.
```

Django wants you to save associated objects yourself.

Django Cascading Delete

Specify that question.delete() should *cascade*

```
class Choice(models.Model):
    """A possible answer to a poll Question"""
    choice_text = models.CharField(max_length=80)
    question = models.ForeignKey(Question,
                    on_delete=models.CASCADE)
```

When you delete a question, all it's choices are deleted, too.

Other Kinds of Associations

There are other cases that ORM must handle:

- 1-to-many and many-to-many associations
- object containing an *ordered* collection, such as List.

Django invisibly handles all these.

For other ORM frameworks like SQLAlchemy (Python) or JPA (Java) it helps to understand how framework handles associations.
Especially cascading save/delete and lazy or eager fetching.

Django Query Methods

`Model.objects` provides many query methods and a simple query syntax.

Django has several built-in methods to compute sum, average, min, max, etc. for a QuerySet.

To use Django effectively, you need to know how to use the query methods.

Making Queries in Django

https://docs.djangoproject.com/en/3.1/topics/db/queries/

Example of a Dumb Query

Find all poll questions containing the word "programming"

```
questions = Question.objects.all()
# Create a list of questions about "programming"
qlist = [ q for q in questions
                if "programming" in q.question_text ]
```

Why is this inefficient?

Python Quiz:

what is `[q for q in questions if ...]` called?

Smarter Query

Let the database filter results for you:

```
questions = Question.objects.filter(
                question_text__icontains='programming')
# questions is a QuerySet.  Convert to a list
qlist = list(questions)
```

Why is this more efficient?

- You don't retrieve lots of data that you don't want.
- You don't create objects that you don't need.

```
# Find questions with pub_date >= 1 Jan 2020
Question.objects.filter(
      pub_date__gte=datetime.date(2020,1,1) )
```

Learn More

Making Queries in Django.

https://docs.djangoproject.com/en/3.1/topics/db/queries/

* You don't need the URL, of course -- *you already have the Django documentation on your own computer, right?*

Lazy Instantiation

Another import ORM property.

Meaning is "*don't create objects until you need them*".

Django QuerySet uses this.

The Django docs describe *lazy instantiation*.

Persistence Frameworks

SQLAlchemy - *"the database toolkit for Python"*

- The most popular ORM framework for Python
- Excellent documentation

EclipseLink - *reference implementation* of the Java Persistence API (JPA) standard for Java

ORMLite - *easy to use Java ORM framework.*

- Has it's own API + provides JPA API.
- Excellent documentation