Data modeling

First things first...

Please turn on your webcams

What is data modeling?

Using abstraction in order to represent and better understand the nature of data flow within an information system

- Relational Databases and SQL in a nutshell
- Unified Modeling Language (ERD, DFD)
- Data modeling in Django ORM

source: https://pixabay.com

Relational Databases and SQL in a nutshell

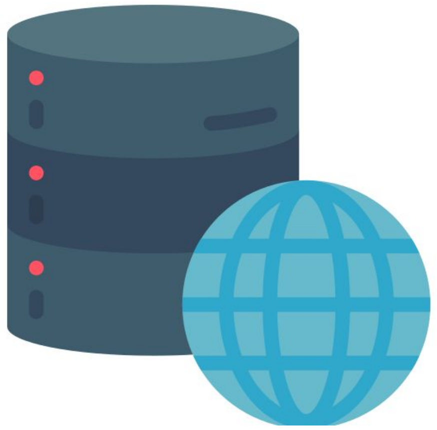
Databases

Relational

- Postgresql
- MySQL/Mariadb
- MS SQL
- Oracle
- SQLite

NoSQL

- MongoDb
- ElasticSearch
- CouchDB
- DynamoDB



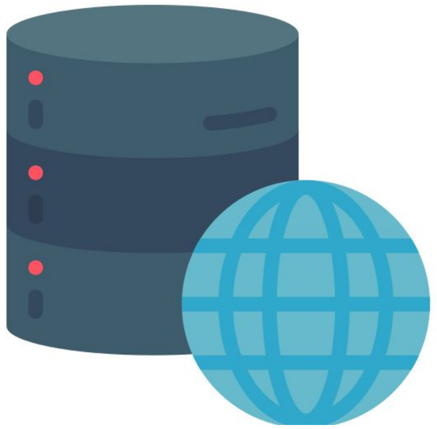
Databases

Create data

Read data

Update data

Delete data



What is a relational database?

Database tables

Information in a relational database resides in multiple tables

Book table

id	name	author-id	language	publication-date	pages	description
000001	origin	0000001	English	2017-10-03	461	When billionaire researcher Edmond Kirsch is killed...
000002	The Da VInci Code	0000001	English	2003-04-15	489	Louvre curator and Priory of Sion grand ma...

Author table

id	First-name	Last-name	Date-of-birth
0000001	Dan	Brown	1964-06-22

ACID Compliance

Atomicity

every transaction (set of statements executed together) either completely succeeds or fails. There is no partial success.

Consistency

All successful transactions keep the database in a valid state (eg.. referential integrity/triggers).

Isolation

Once a transaction starts, no concurrent transactions will be visible to it until it completes.

Durability

Once a transaction completes, the data will remain even if there is a power failure.

The basics of SQL language

Constraints

Primary key
Each row in every table should have a one-to-one unique identifier that represents the row.
Cannot be NULL and should not change over time.
example: Row ID, Social security number.

Foreign key
A reference to another table's **primary key**.
This allows you to join tables together to retrieve all the information you need without duplicating data.
example: a Book's author ID.

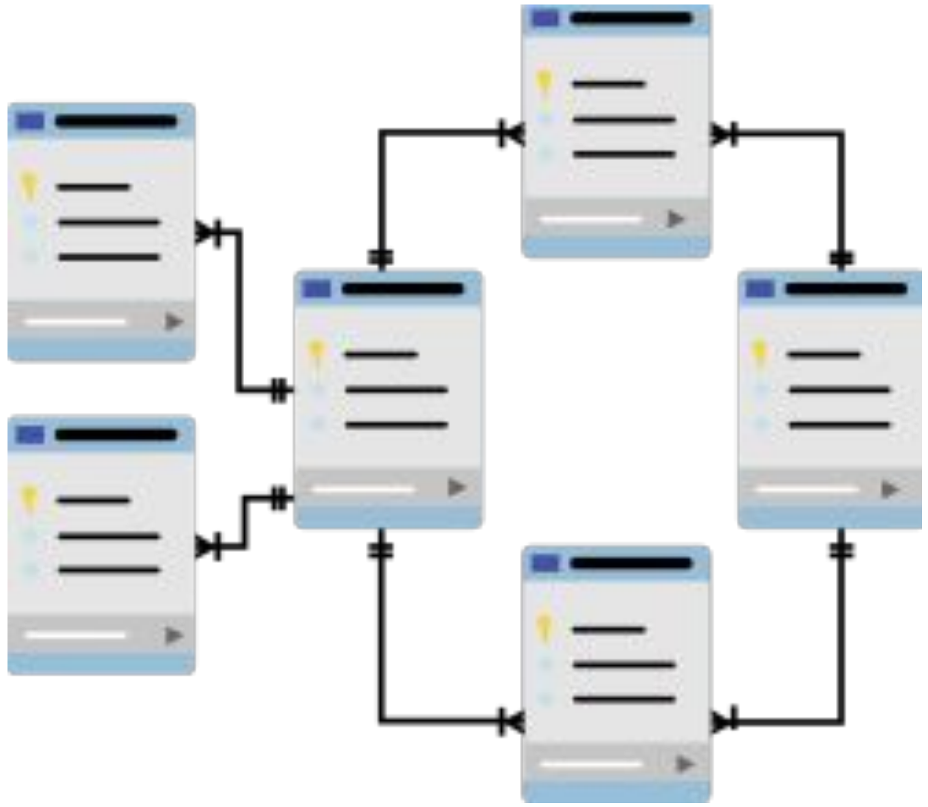
Unique
You can declare a field or set of fields to be unique (one-to-one), even if they are not part of the primary key.
example: username, domain.

Deletion
You can define what should happen if you try to delete a row in a table whose primary key is referenced in another table.
example: when a user is removed, delete all of his posts.

The basics of SQL language

Types

- **int** - integer value (e.g. 5)
- **numeric** - exact decimal number (e.g. 5.53)
- **bool** - TRUE/FALSE (stored as 0/1)
- **date** - YYYY-MM-DD
- **time** - 00:00:00
- **datetime** - YYYY-MM-DD 00:00:00
- **varchar**(x) - text x <= 65,535
- **enum** - list of valid values



The basics of SQL language

Common commands

- **CREATE TABLE** y (<column_name> <type> (<attribute>), ...); → book(name varchar(50), ...);
- **INSERT INTO** y (column1_info, column2_info, ...);
- **SELECT** column_name **FROM** table_name
  - **WHERE** condition
  - **ORDER BY** column_name;
- min(), max(), avg(), count(), sum(), <, >, =, +, -, *, /, %, &, ^, AND, IN, NOT, ANY, EXISTS...

name	author-id	language	publication-date	pages	description
origin	0000001	English	2017-10-03	461	When billionaire researcher Edmond Kirsch is killed...

The basics of SQL language

Commands

- **CREATE TABLE** books (id int, name varchar(50), author-id int, language varchar(20), publication-date DATE, pages int, description varchar(400), PRIMARY KEY(id), FOREIGN KEY (author-id))

id	name	author-id	language	publication-date	pages	description

- **INSERT INTO** books (0000001, origin, 0000001, English, 2017-10-03, 461, When billionaire researcher...)

id	name	author-id	language	publication-date	pages	description
0000001	origin	0000001	English	2017-10-03	461	When billionaire researcher...

- **- SELECT** language **FROM** books

language
English
Spanish

- SELECT publication-date **FROM** books **WHERE** pages **<** 480

publication-date
2017-10-03

Designing with Normalization in mind

- Reduce redundancy and unnecessary duplication.
- Helps produce database systems that are cost-effective and have better security models.
- Seven levels of normalization (1NF, 2NF, ..., 6NF, BCNF). Most database systems are normalized up to 3NF.
- What's relevant under the course's scope:
  - Each table cell should contain a single value.
  - No duplication of rows.
  - Has <u>no transitive functional</u> dependencies (A → B & B → C = A→ C).
    - Author_id = 1, Book_id = 3, Nationality = Israeli

      Book_id → Author_id: if we know the book's name, we can learn the author's name

      Author → Nationality: If we know the Author's name, we can determine his nationality.

      Book → Nationality: if we know the book's name, we can determine the author's nationality.
    - Instead:

      Auther_id = 1, Nationality = Israeli | Book_id = 3, Auther_id = 1.
- Functional dependencies are a **very important** component of the normalize data process.

Designing with Normalization in mind

How will we get all the books made by Terry Pratchett?

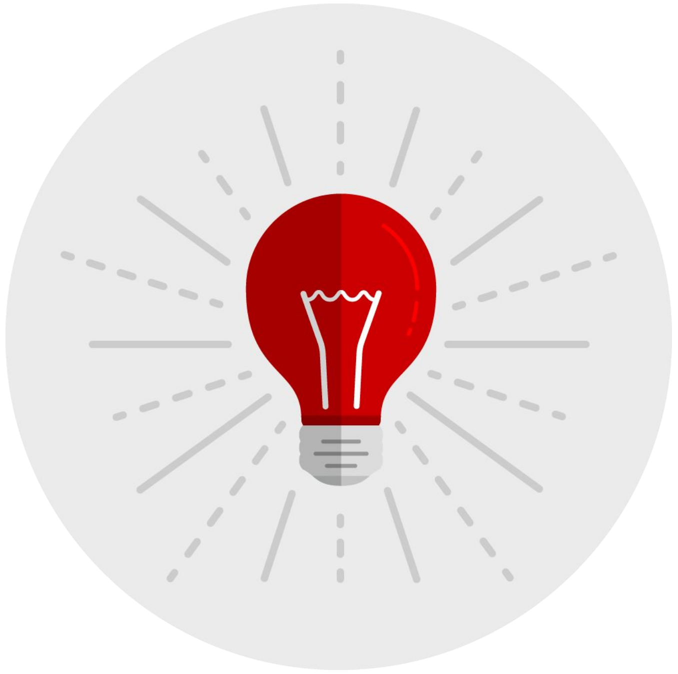
- If I want to find all the books made by Terry Pratchett, I will…
  - X “look at all the books listed under him”
  - ✓ “look at all the books, and filter filter them by Terry’s Author-id”:

    SELECT * FROM books WHERE Author_id==1

Id (PK)	First-name	Last-name	Date-of-birth
000001	Terry	Pratchett	1948-04-28

Book-id (PK)	name	Author-id (FK)	language	publication-date
000001	The Colour of Magic	000001	English	1983-10-03
000001	The Shepherd's Crown	000001	English	2015-08-27

Revision Questions



Question #1
What is a primary key in a relational database?

Question #2
What are the 4 main functions of a Database?

Unified Modeling Language (ERD, DFD)

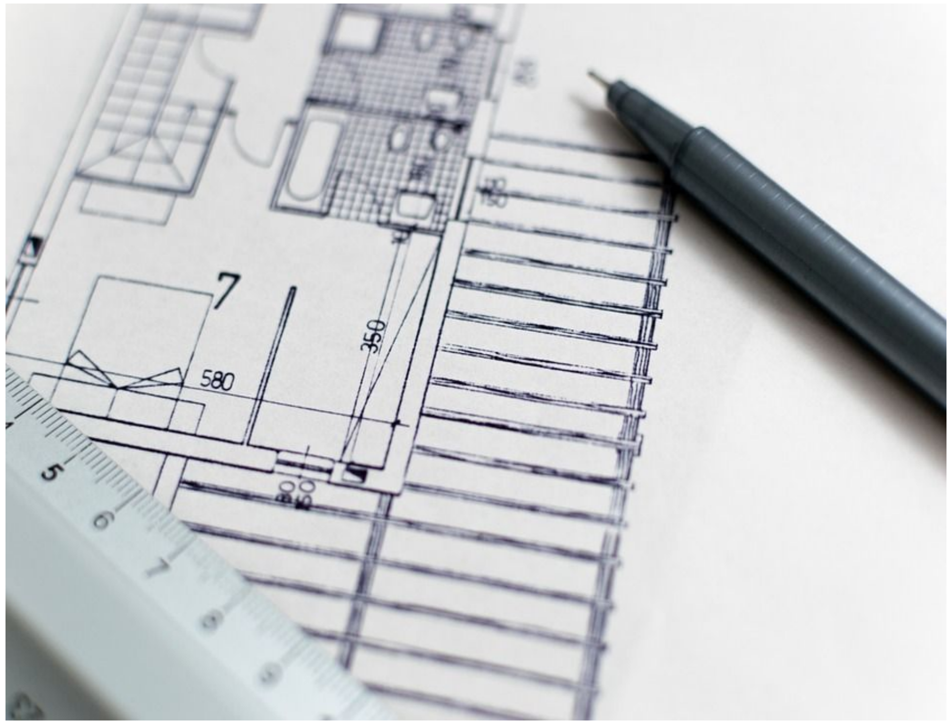
UML - ERD

Entity **R**elationship **D**iagram

A Blueprint of our database and entities.

Shows the **design** of the database
and the **relationship** between different entities in the system

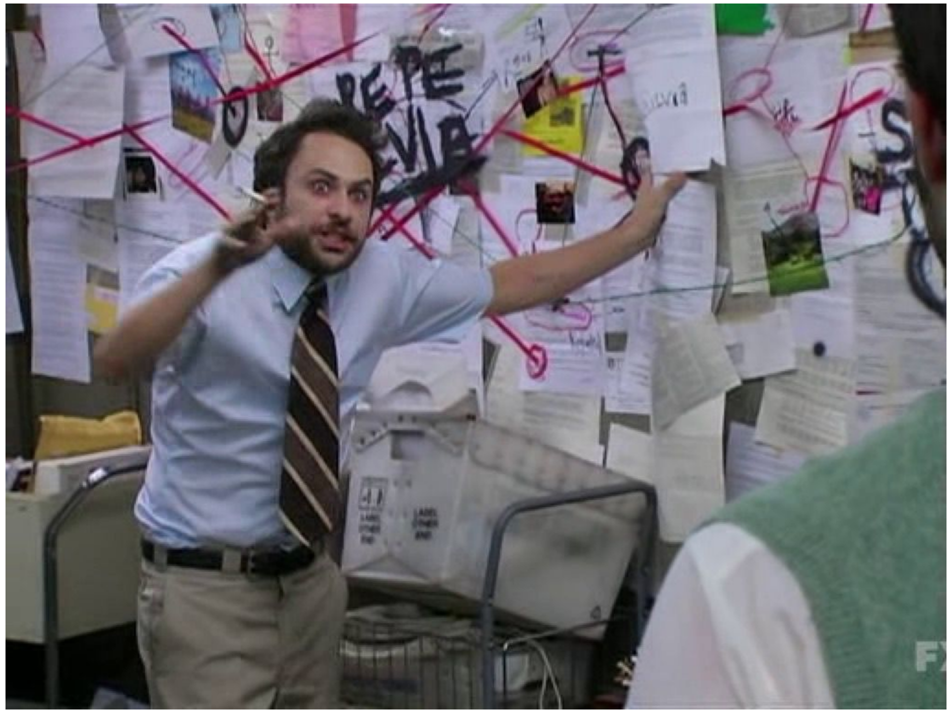
- Tools: lucidcharts, draw.io, word, pen and paper…



source: https://pixabay.com

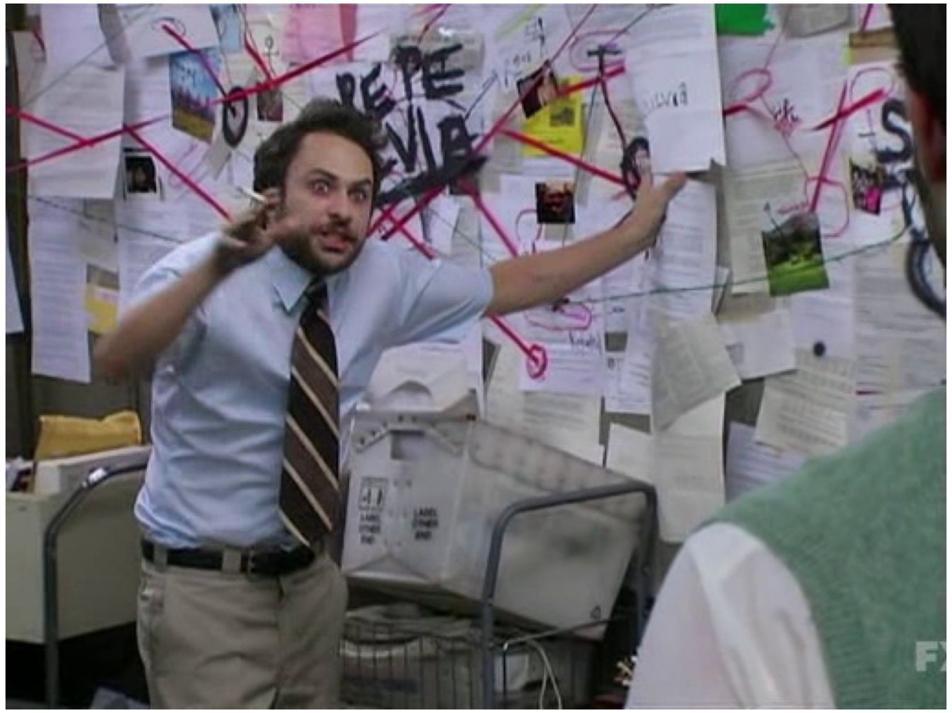
UML - ERD

- Describe all the entities and their relationships.
- Easy to understand (non-technical).
    - Readable.
- Take into account Django's builtins.



UML - ERD

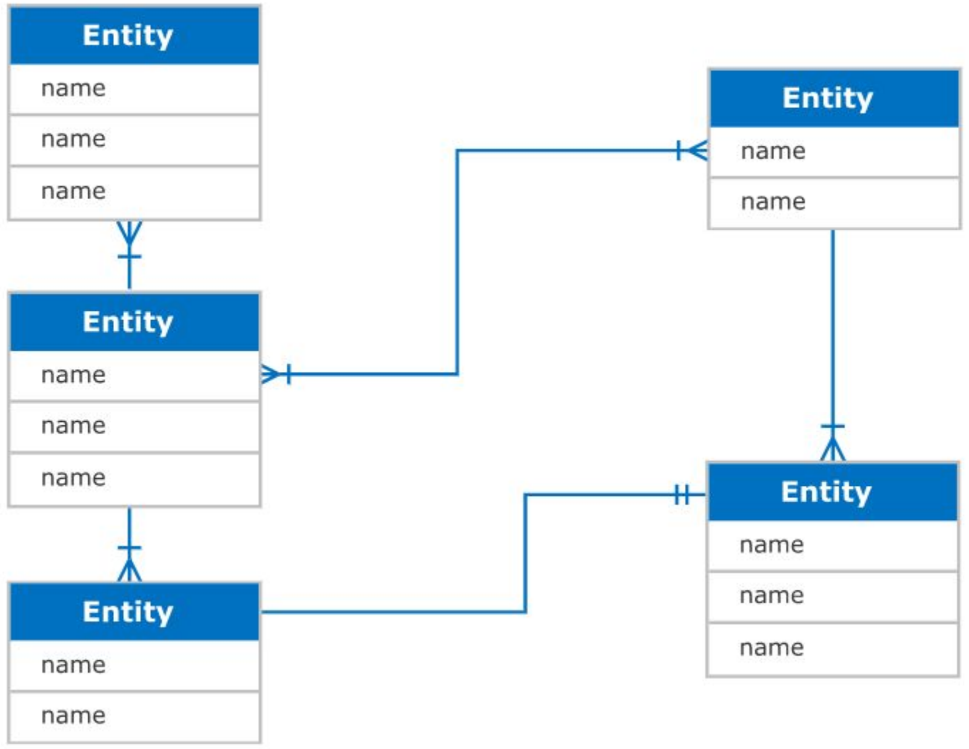
- **Use cases:**
  - Designing an application with multiple entities.
  - Choosing between multiple designs.
  - Multiple developers working on a project.
  - Describe your application (e.g: to stakeholders).
- **Why use it**
  - Ease the development of the application.
  - Reach decisions and conclusions faster.
  - indicate issues in the system before they occur.
  - Make structure and topology clear to both. developers and stakeholders.



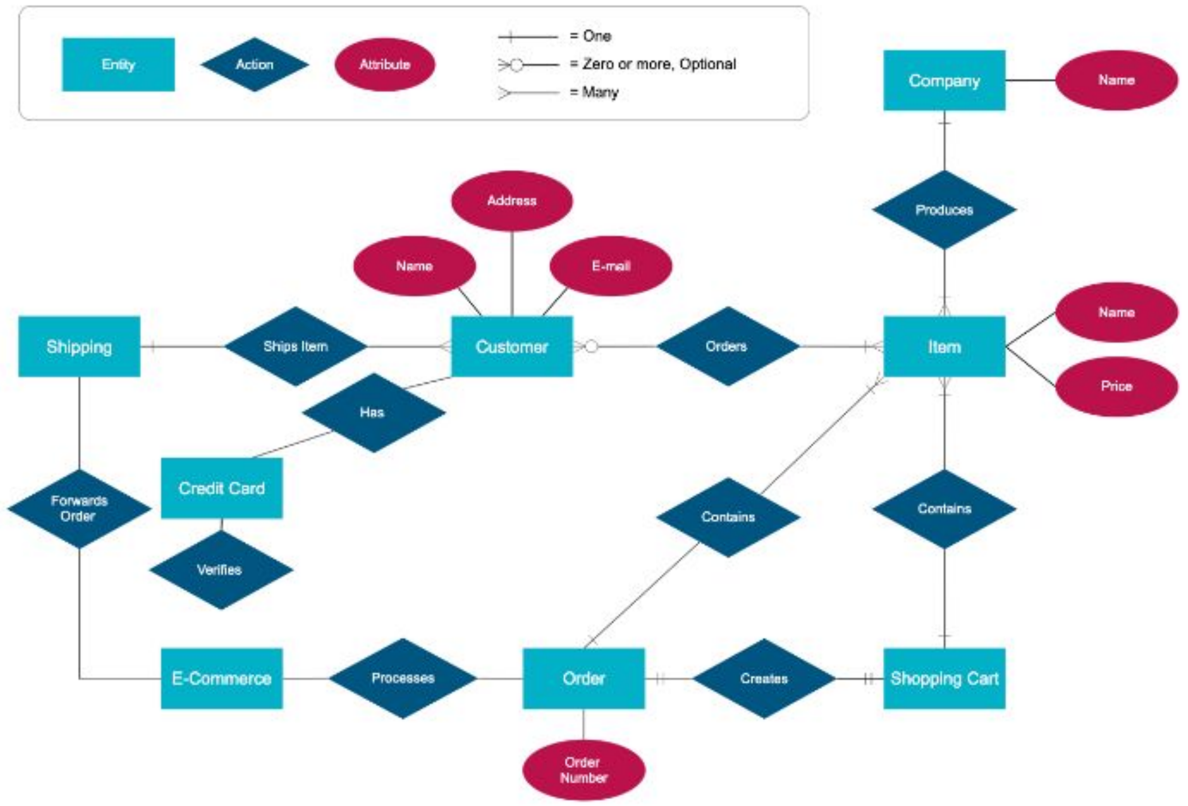
UML - ERD

How does it look

IDEF1X Notation ERD - Relational Schema

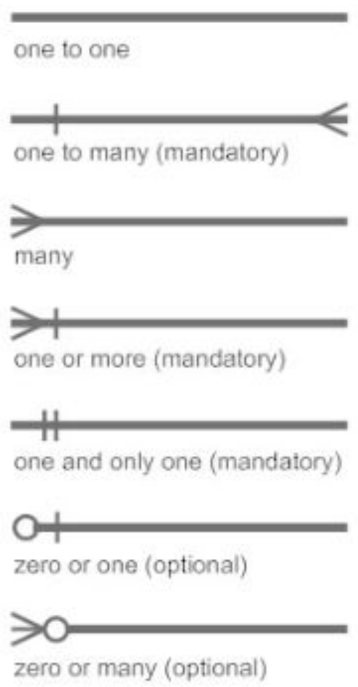


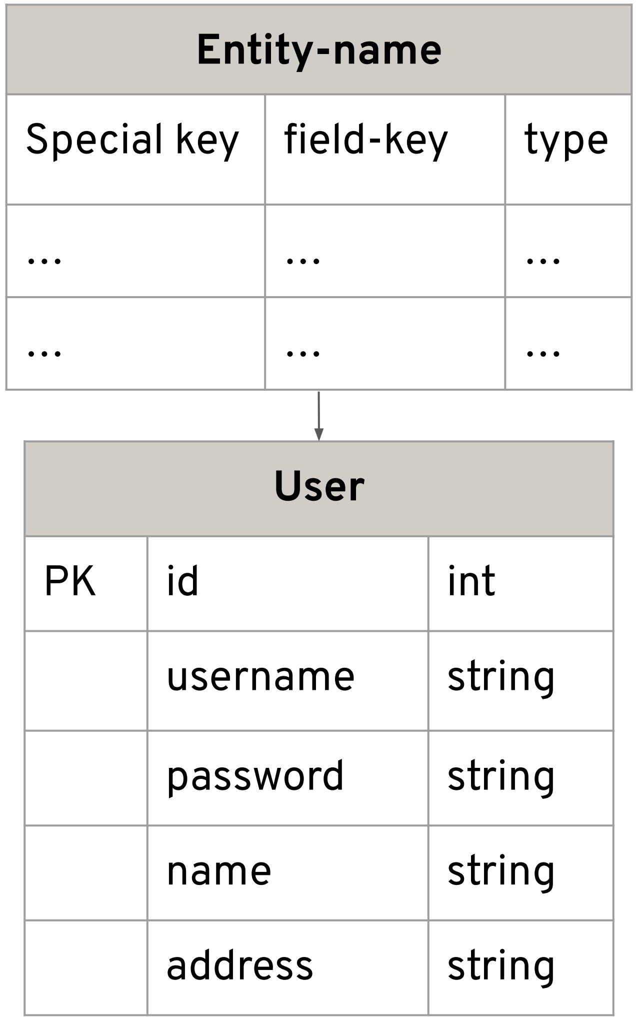
Traditional ERD



UML - ERD

How does it look





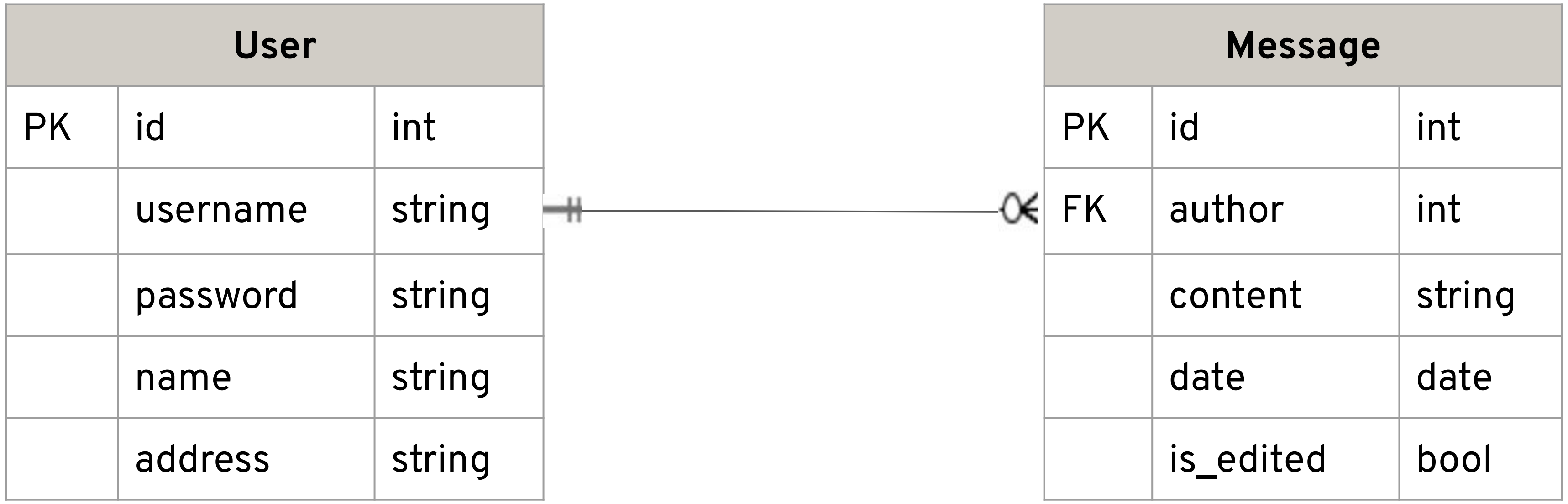
Entity-name		
Special key	field-key	type
…	…	…
…	…	…

User		
PK	id	int
	username	string
	password	string
	name	string
	address	string

UML - ERD

Examples

“A system in which users will be able to post textual content to a message board. Users should have a username, password, name, and address. Each message on the board should include the author’s information, message’s content, date, and a flag for marking if message is edited ”



User		
PK	id	int
	username	string
	password	string
	name	string
	address	string

Message		
PK	id	int
FK	author	int
	content	string
	date	date
	is_edited	bool

Hotel reservation ERD - Exercise

Create an ERD for an Hotel reservation system.

1. Guests can reserve a room at the hotel.
2. Guests sign up using their full name, email and password.
3. Each room can be reserved only to one guest at a given time.
4. Each reservation include 1 room only.
5. Guests can leave a review for their stay (unless it is still pending).
6. We can assume all the rooms in the hotel are the same in terms of size and type.

Countdown - 5 minutes

source: https://pixabay.com

Hotel reservation ERD - Solution

1. **Guest**: email, first name, last name, password
2. **Room**: number, floor, description, price per night
3. **Reservation**: id, guest email, room number, start date, end date, price per night, status
4. **Review**: id, guest id, reservation id, rating, description

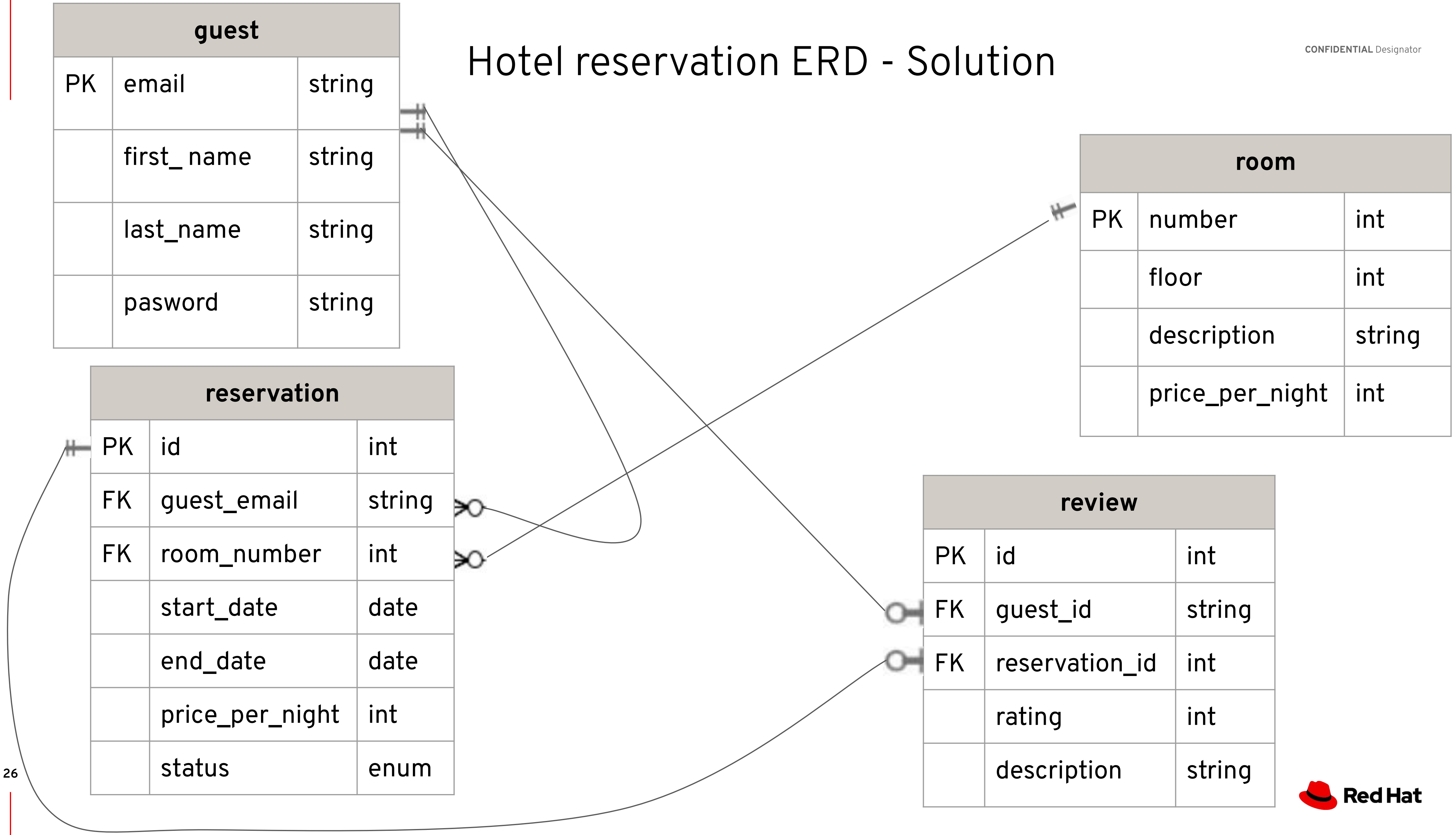
guest		
PK	email	string
	first_ name	string
	last_name	string
	password	string

room		
PK	number	int
	floor	int
	description	string
	price_per_night	int

reservation		
PK	id	int
FK	guest_email	string
FK	room_number	int
	start_date	date
	end_date	date
	price_per_night	int
	status	enum

review		
PK	id	int
FK	guest_email	string
FK	reservation_id	int
	rating	int
	description	string

Hotel reservation ERD - Solution



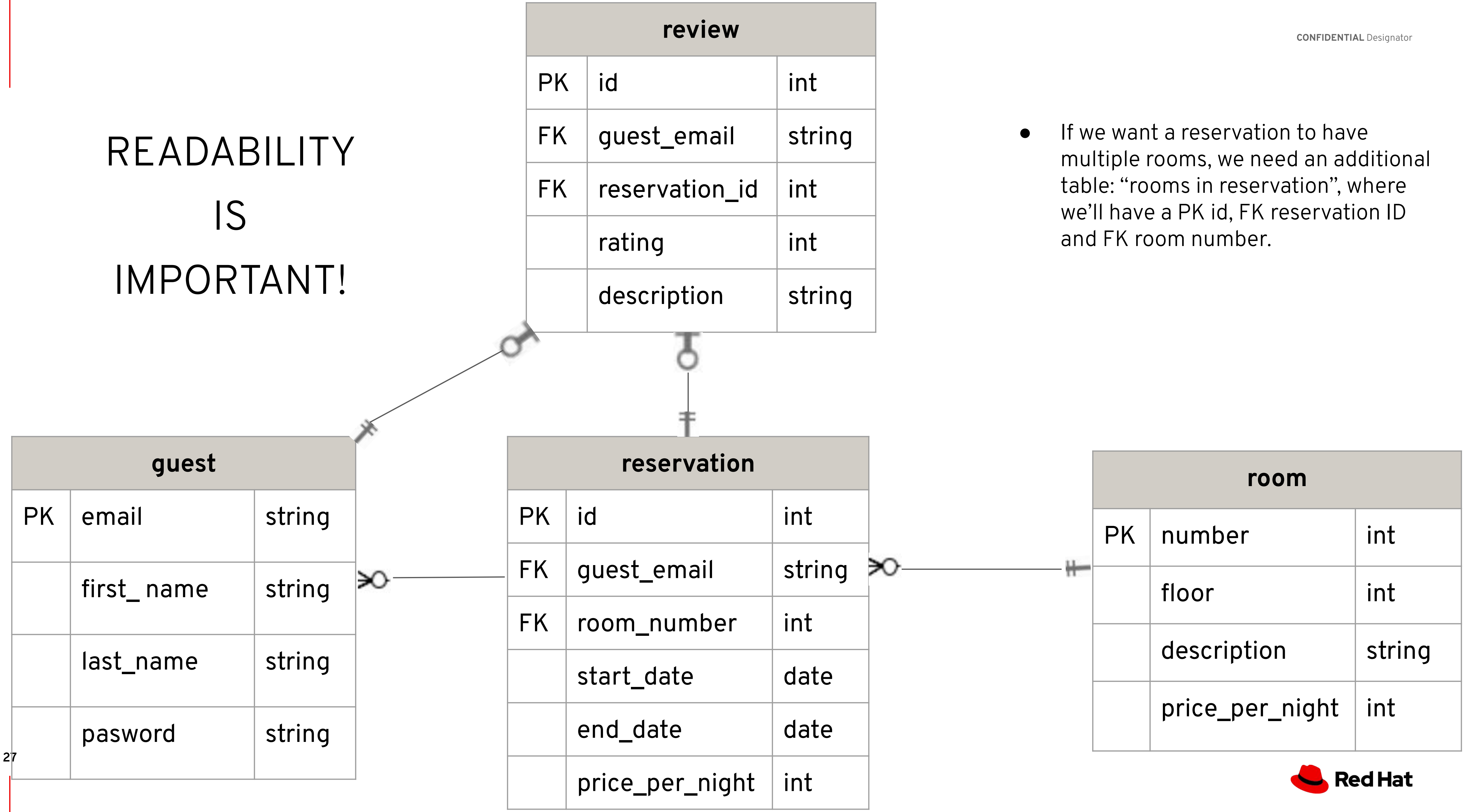
guest		
PK	email	string
	first_ name	string
	last_name	string
	pasword	string

reservation		
PK	id	int
FK	guest_email	string
FK	room_number	int
	start_date	date
	end_date	date
	price_per_night	int
	status	enum

room		
PK	number	int
	floor	int
	description	string
	price_per_night	int

review		
PK	id	int
FK	guest_id	string
FK	reservation_id	int
	rating	int
	description	string

READABILITY IS IMPORTANT!



review

PK	id	int
FK	guest_email	string
FK	reservation_id	int
	rating	int
	description	string

guest

PK	email	string
	first_ name	string
	last_name	string
	pasword	string

reservation

PK	id	int
FK	guest_email	string
FK	room_number	int
	start_date	date
	end_date	date
	price_per_night	int

room

PK	number	int
	floor	int
	description	string
	price_per_night	int

- If we want a reservation to have multiple rooms, we need an additional table: "rooms in reservation", where we'll have a PK id, FK reservation ID and FK room number.

Hotel reservation ERD - Solution

1. Get all currently available rooms → **SELECT room_number FROM reservation WHERE status != "ACTIVE"**
2. Is room 1234 in use currently (if there is a result - it is not in use) → **SELECT * FROM reservation WHERE status != "ACTIVE" and room_number=1234**
3. Get all reviews by a user → **SELECT * FROM review WHERE guest_email="mail@gamil.com"**

guest		
PK	email	string
	first_ name	string
	last_name	string
	password	string

room		
PK	number	int
	floor	int
	description	string
	price_per_night	int

reservation		
PK	id	int
FK	guest_email	string
FK	room_number	int
	start_date	date
	end_date	date
	price_per_night	int
	status	enum

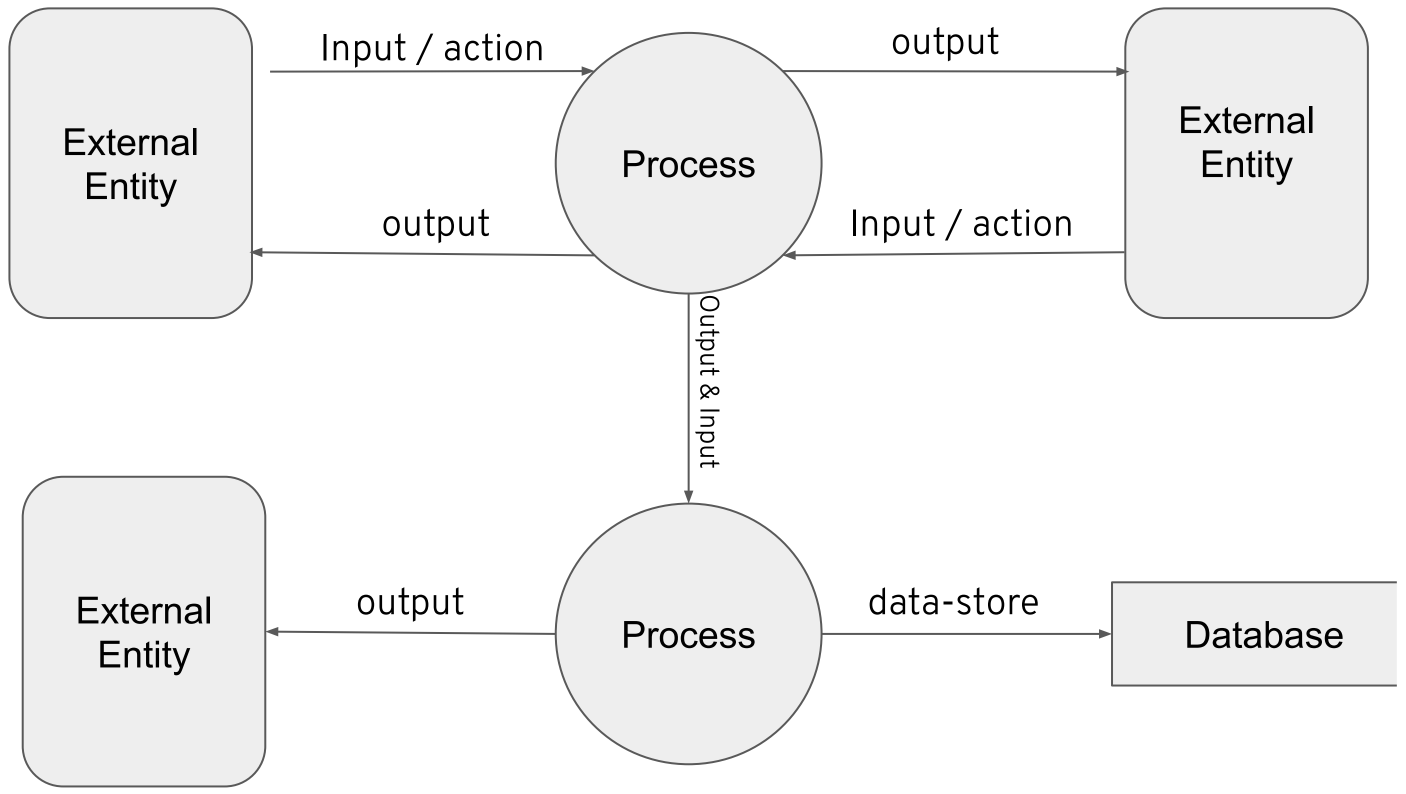
review		
PK	id	int
FK	guest_email	string
FK	reservation_id	int
	rating	int
	description	string

UML - DFD

Data **F**low **D**iagram

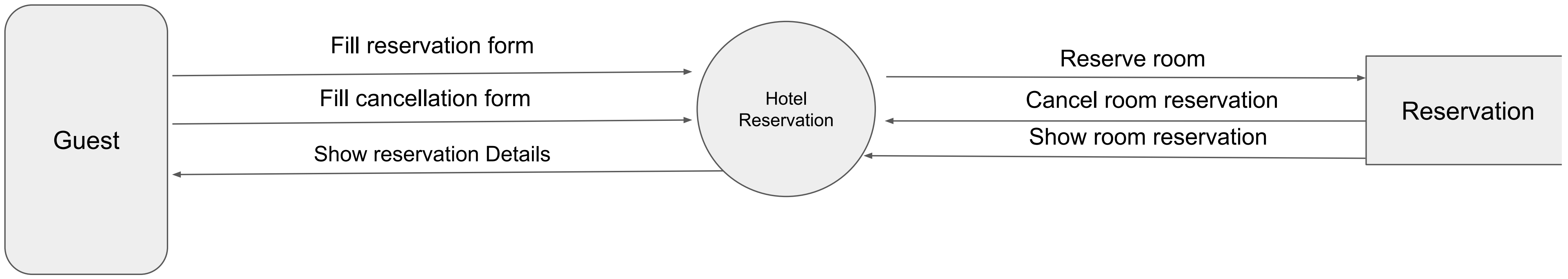
- Make it clear what the expected user stories are.
- Maps out the flow of information for any process or system.
- Shows the functionality the application is expected to have, which entity uses it and what inputs or outputs it has.
    - Useful in writing functions and especially **tests.**
    - Communicate what functionality the application is expected to have.

UML - DFD


External Entity
Input / action
Process
output
External Entity
output
Input / action
Output & Input
External Entity
output
Process
data-store
Database

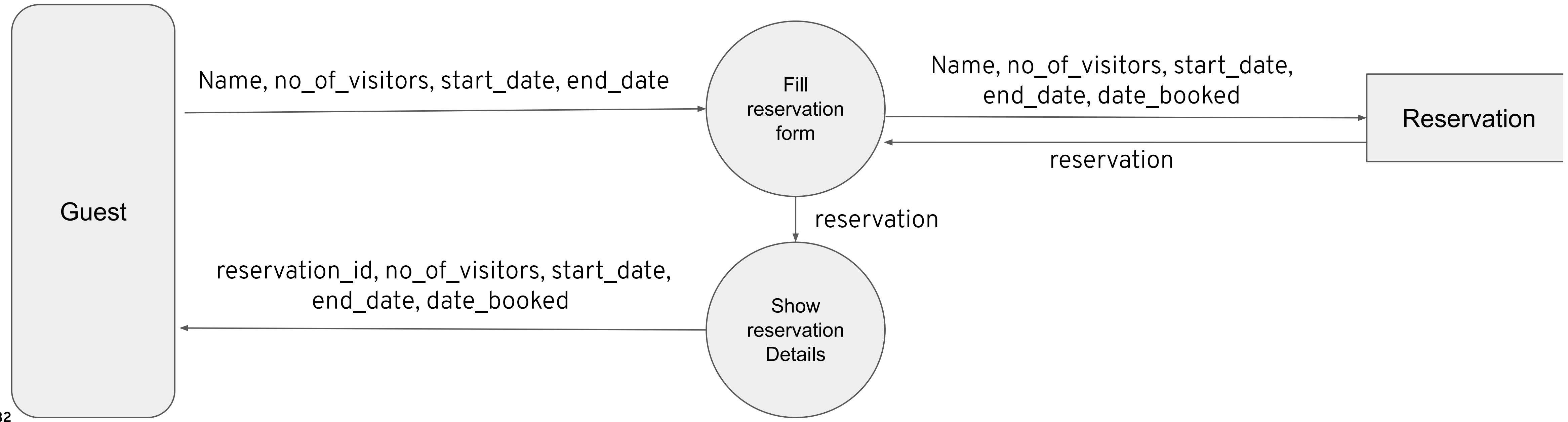
UML - DFD

- Level 0 - Context diagram - at-a-glance view, showing the system as a single high-level process, with its relationship to external entities.
- A simple visual layout of multiple functions and process in the application.

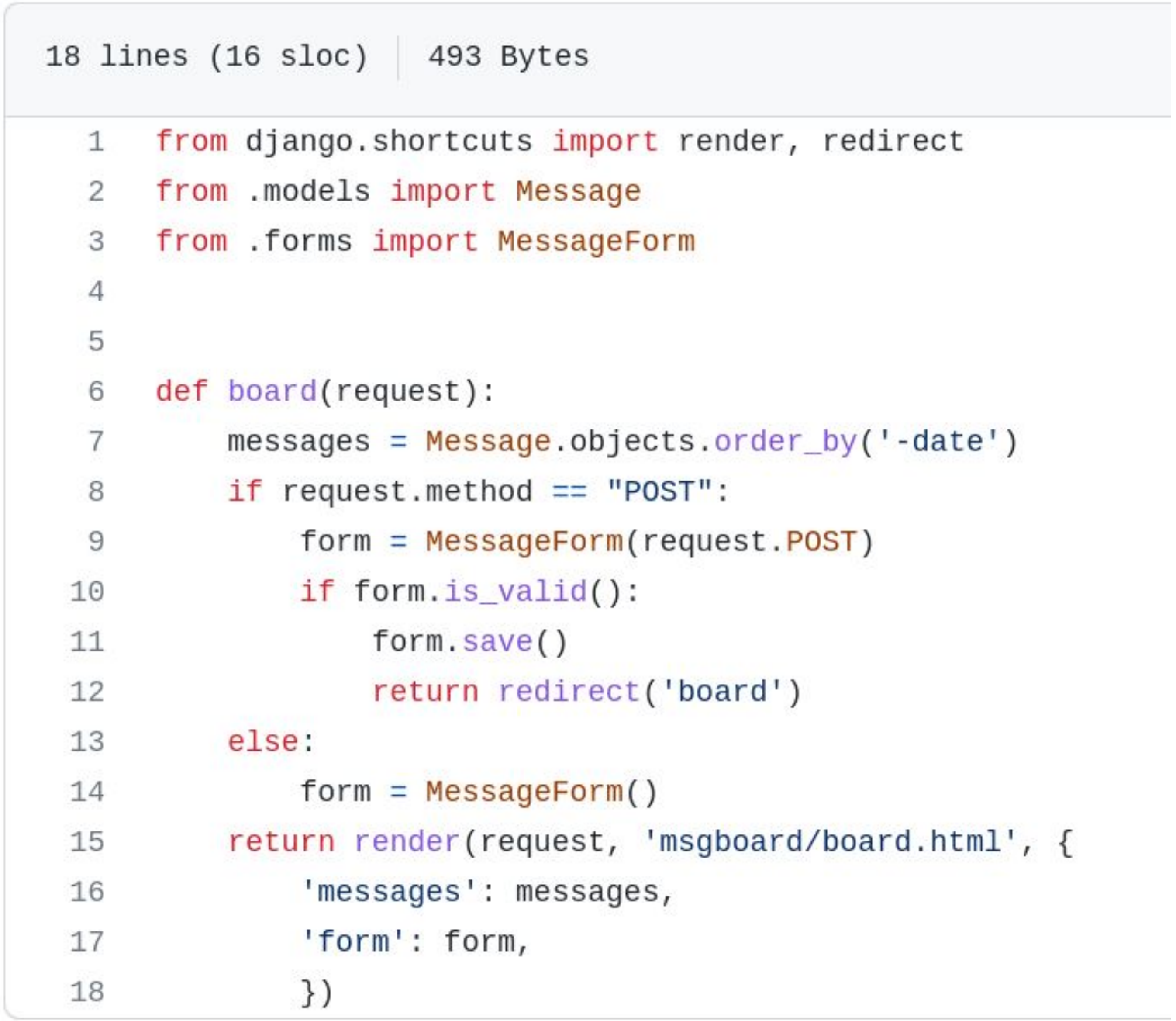


UML - DFD

- Level 1 - still a general overview.
- A simple (but more specific) layout of a single process in the application.


Guest
Name, no_of_visitors, start_date, end_date
Fill reservation form
Name, no_of_visitors, start_date, end_date, date_booked
Reservation
reservation
reservation
reservation_id, no_of_visitors, start_date, end_date, date_booked
Show reservation Details

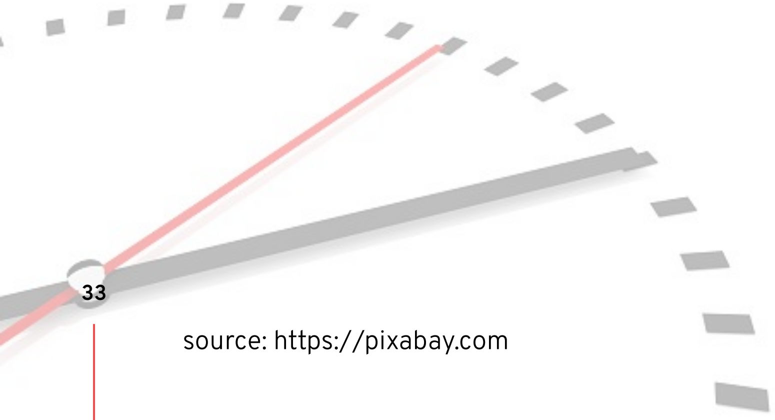
What does it do?



18 lines (16 sloc) | 493 Bytes

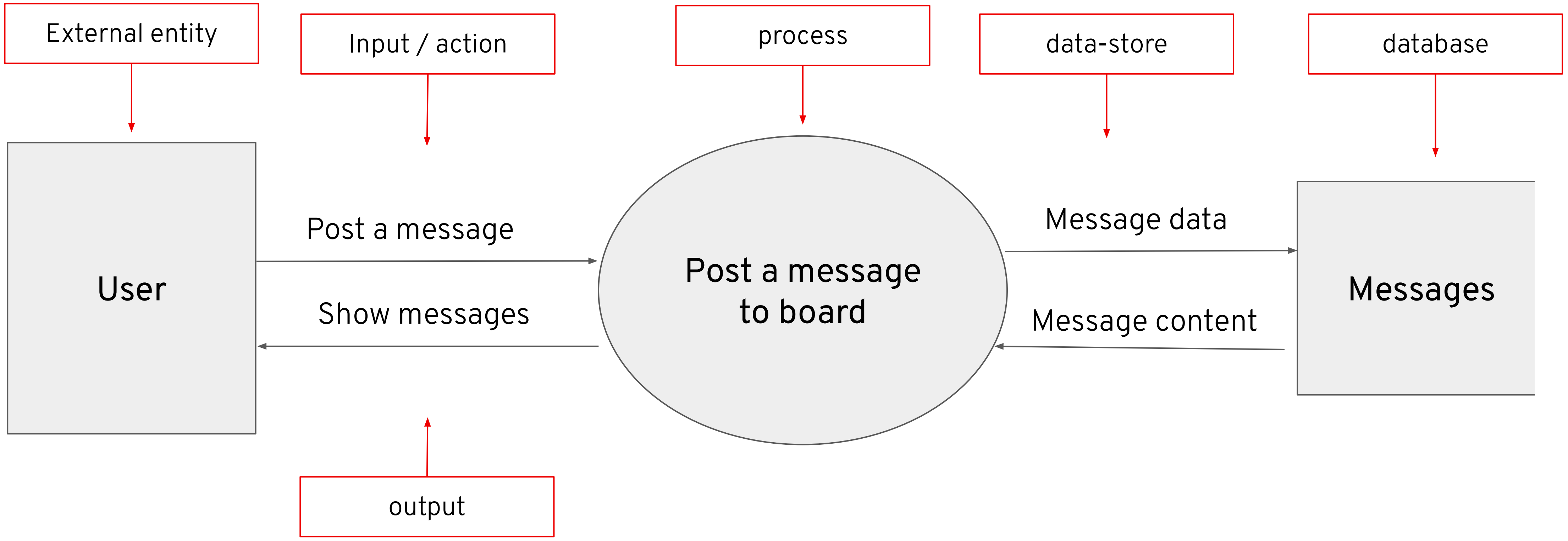
```
from django.shortcuts import render, redirect
from .models import Message
from .forms import MessageForm


def board(request):
    messages = Message.objects.order_by('-date')
    if request.method == "POST":
        form = MessageForm(request.POST)
        if form.is_valid():
            form.save()
            return redirect('board')
    else:
        form = MessageForm()
    return render(request, 'msgboard/board.html', {
        'messages': messages,
        'form': form,
        })
```

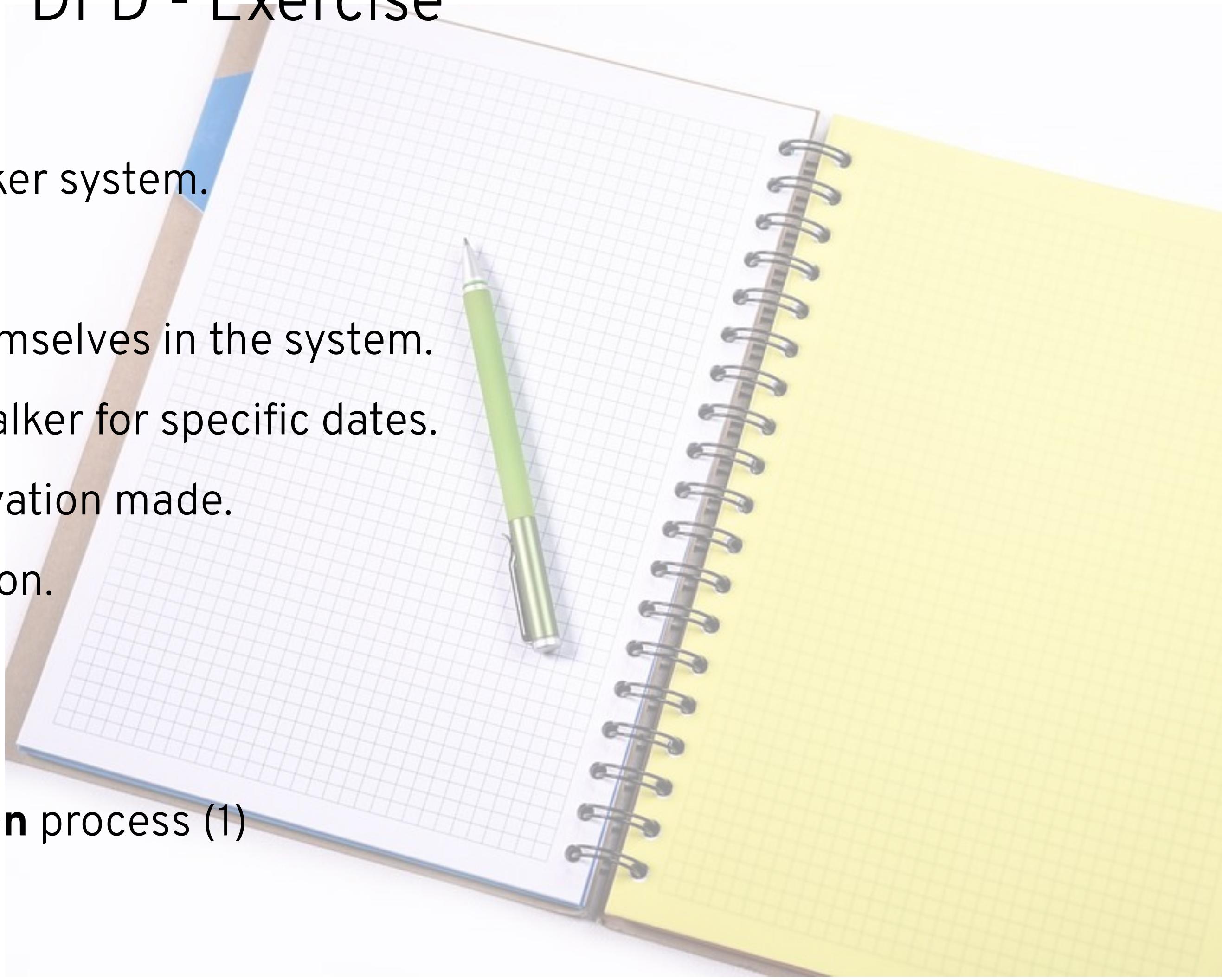


source: https://pixabay.com

What does it do?



‘Walk my dog’ DFD - Exercise



Create a level 0 DFD for the ‘walk my dog’ dogwalker system.

Cover the following user stories:

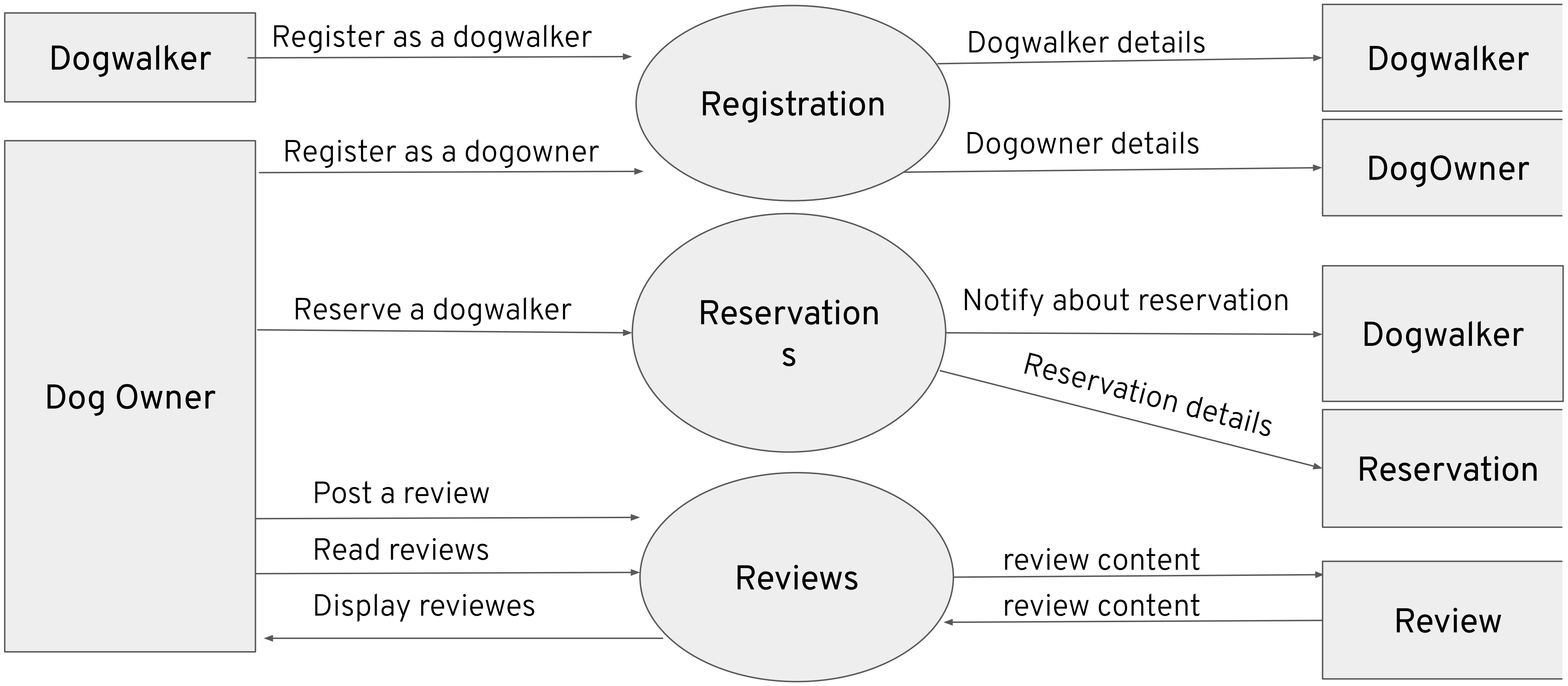
1. Dog walkers and dog owners can register themselves in the system.
2. Dog owner should be able to reserve a dog walker for specific dates.
3. Dog walkers should be notified about a reservation made.
4. Dog owner can post a review for the reservation.
5. Dog owner can read dogwalkers’ reviews.

- Bonus: Create a level 1 DFD for the **reservation** process (1)

source: https://pixabay.com

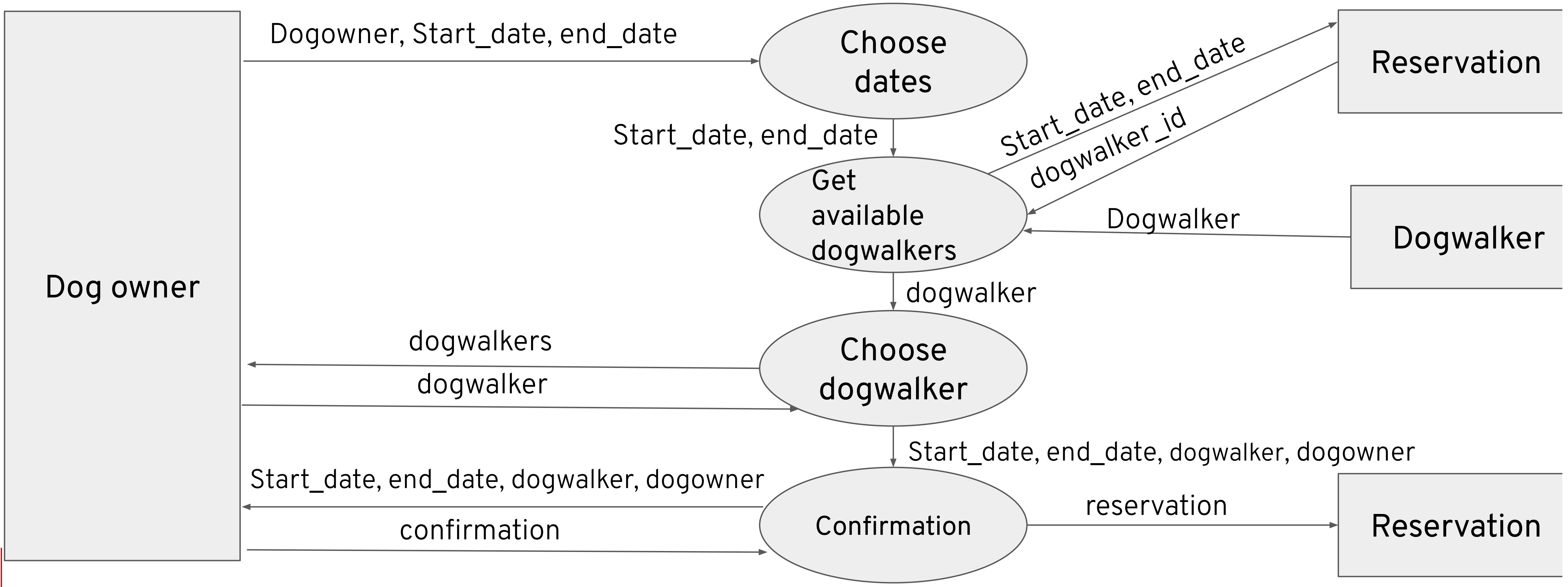
Hotel reservation DFD - Exercise

Level 0 DFD

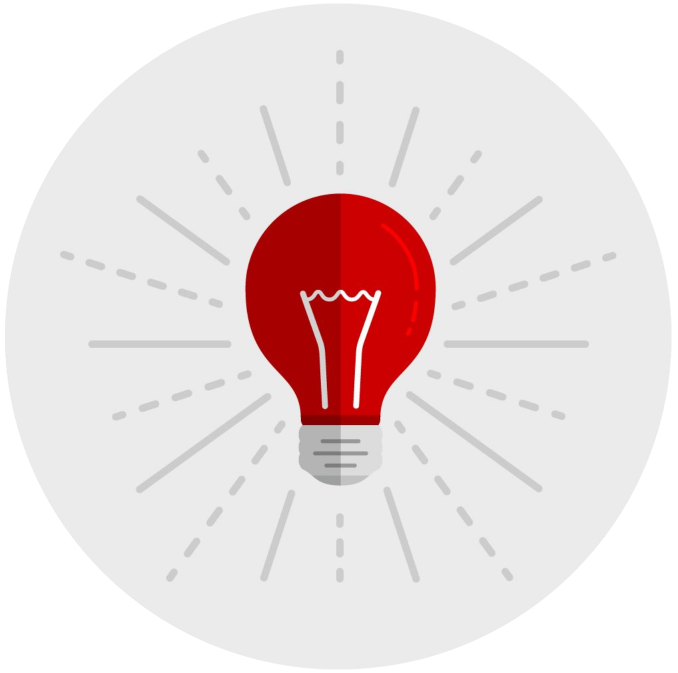


Hotel reservation DFD - Solution

1. Dog owner should be able to reserve a dog walker to specific dates.



Revision Questions



Question #1
When will we use UML?

Question #2
What is the difference between ERD and DFD?

Question #3
In what occasions will I have a FK as a field in an entity?

Data modeling in Django ORM

Object Relational Mapping (ORM)

Why?

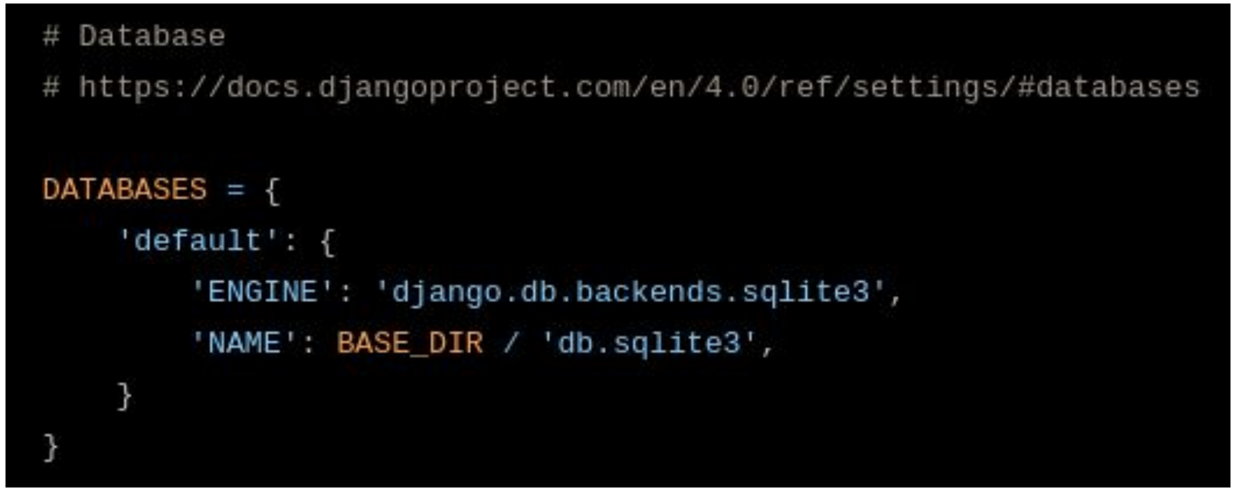
- Database as code
- Can be tracked in version control
- Enables code review process
- Enables usage as a native OOP structure
- Encapsulates SQL complexity
- Less worry about SQL injection
- Database abstraction, switch DBS when you want

Challenges

- A different mindset than software development
- Different ORMs has unique way of working
- Complex functionality likely to be inefficient or impossible

Django ORM - step by step

Django **settings.py**



```
# Database
# https://docs.djangoproject.com/en/4.0/ref/settings/#databases

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'db.sqlite3',
    }
}
```

Note: UTF-8 coding is automatic with sqlite,

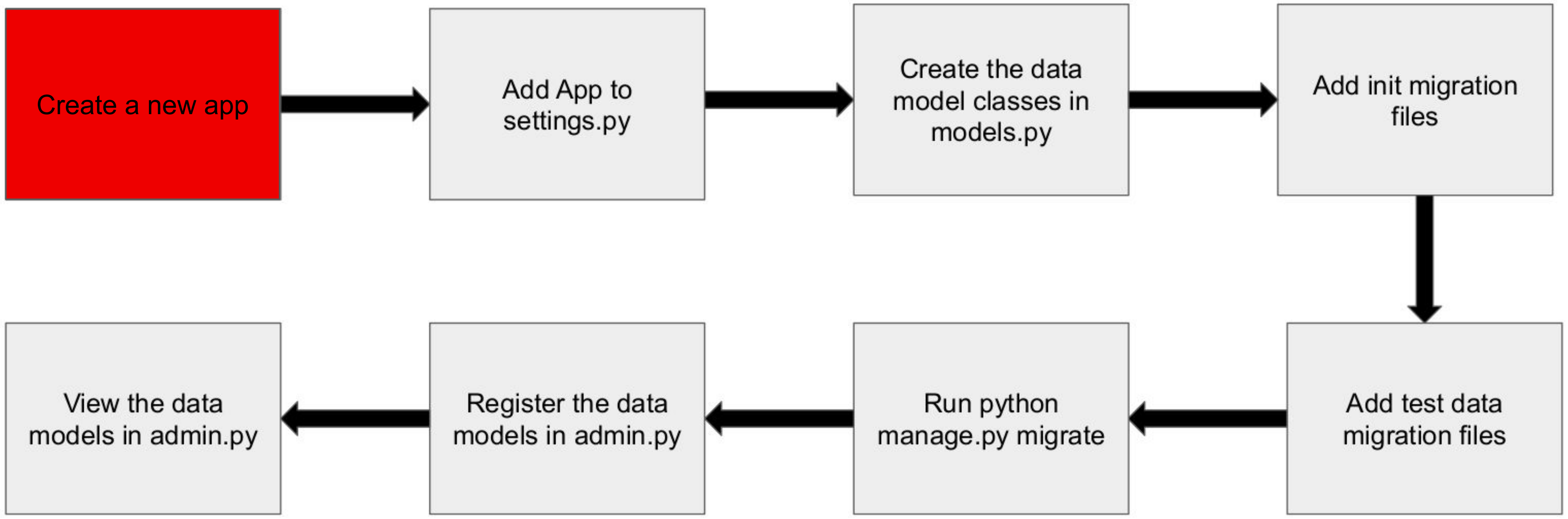
But needs to be defined in other databases such as mysql/postgresql

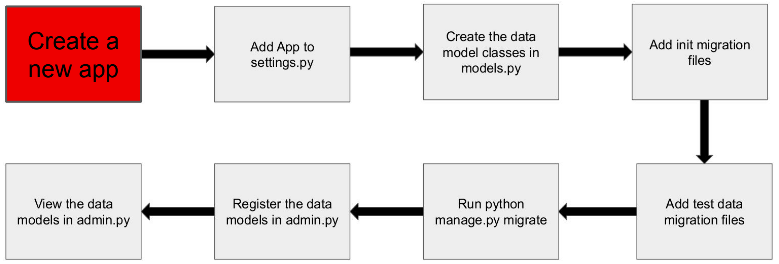
Django ORM - step by step

Run SQLite Command-line with django:

```
$ sudo dnf install sqlite
$ pipenv run python
$ manage.py dbshell
```

Django ORM - step by step


Create a new app
Add App to settings.py
Create the data model classes in models.py
Add init migration files
Add test data migration files
Run python manage.py migrate
Register the data models in admin.py
View the data models in admin.py

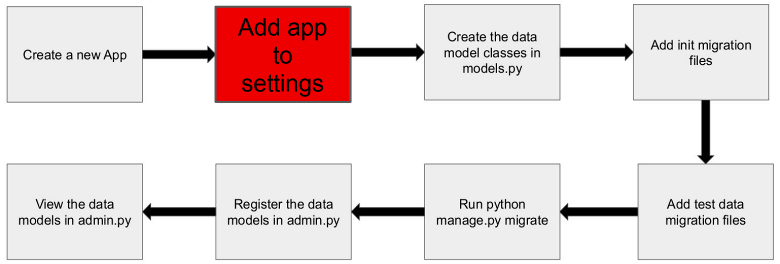

Create a new app
Add App to settings.py
Create the data model classes in models.py
Add init migration files
Add test data migration files
Run python manage.py migrate
Register the data models in admin.py
View the data models in admin.py

step by step - Create a new app

- Start by creating a new app
- An app can represent an entity, a functionality, or the whole application
- We recommend creating an app per entity (e.g: student, orders)
    - Scalability
    - Teamwork
    - Managing migrations
    - Readability

- `$ python manage.py startapp student`

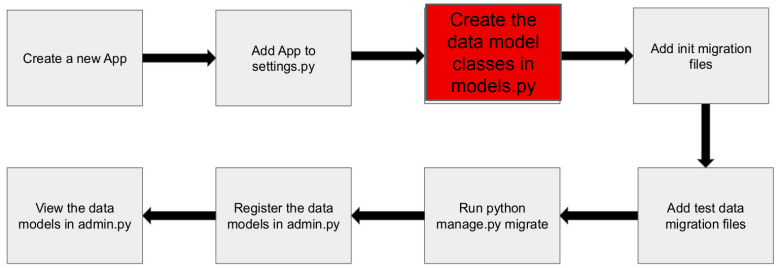
```
student/
    __init__.py
    admin.py
    apps.py
    migrations/
        __init__.py
    models.py
    Tests.py
    views.py
```


Create a new App
Add app to settings
Create the data model classes in models.py
Add init migration files
Add test data migration files
Run python manage.py migrate
Register the data models in admin.py
View the data models in admin.py

step by step - Add app to settings

- Add the new app ('student') to settings.py.
  There should already be a list of default apps.
- Django won't be aware of new apps without adding them there.

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'student.apps.StudentConfig',
]
```


Create a new App
Add App to settings.py
Create the data model classes in models.py
Add init migration files
Add test data migration files
Run python manage.py migrate
Register the data models in admin.py
View the data models in admin.py

step by step - Data model classes

- student/models.py

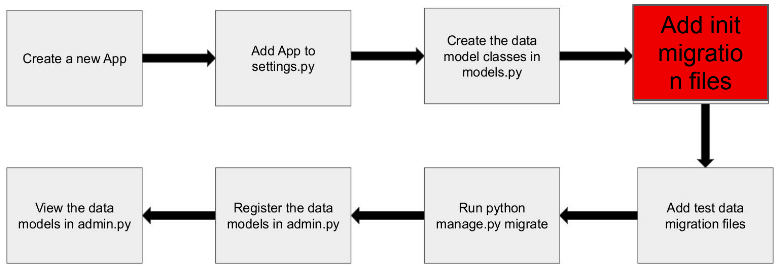
```
class Gender(models.TextChoices):
   Male = 'M', 'Male'
   Female = 'F', 'Female'
   Unspecified = 'UN', 'Unspecified'

class Student(models.Model):
   user = models.OneToOneField(settings.AUTH_USER_MODEL,
                               on_delete=models.CASCADE)
   student_id = models.IntegerField(primary_key=True)
   nickname = models.CharField(max_length=30, blank=True)
   phone_number = models.IntegerField(null=True, blank=True)
   gender = models.CharField(max_length=2, choices=Gender.choices, default='UN', blank=True)
```

- SQL → ORM

○ Varchar	→	CharField
○ Int	→	IntegerField
○ foreign key	→	OneToOneField
○ primary key	→	primary_key=True
○ Deletion	→	on_delete=...
○ ...		

➢ Use built-in Django authentication system (user) for the basic fields (name, email, password, …) and an additional table for class-specific info..


Create a new App
Add App to settings.py
Create the data model classes in models.py
Add init migration files
Add test data migration files
Run python manage.py migrate
Register the data models in admin.py
View the data models in admin.py

step by step - init migration files

The way of propagating changes you make to your models (adding a field, deleting a model, etc.) into your database schema

- Saving changes as migrations:

```
$ python manage.py makemigrations student
```

```
student/
    migrations/
        __init__.py
        0001_initial.py
    …
```

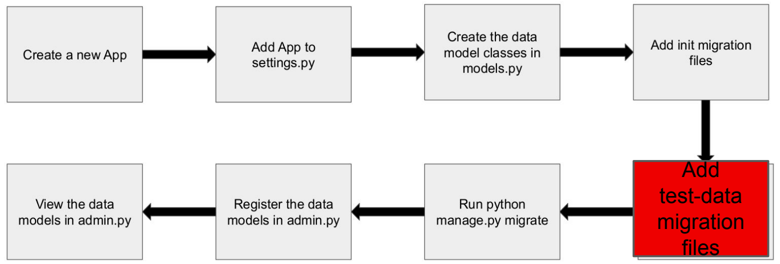
Django ORM - init migration files

IMPORTANT:

NEVER EDIT MIGRATION FILES MANUALLY!

migrations are entirely derived from your models file, and are essentially a history that Django can roll through to update your database schema to match your current models.
Updated are built upon each other, perform changes by adding layers instead of editing the existing ones.

step by step - test-data migrations

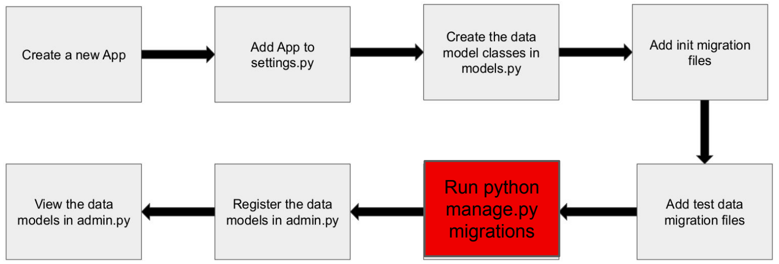

Create a new App
Add App to settings.py
Create the data model classes in models.py
Add init migration files
Add test-data migration files
Run python manage.py migrate
Register the data models in admin.py
View the data models in admin.py

- Exception to the rule: those migrations are written manually.
- Pre-made data for reliable tests
- Will be touched on more later.

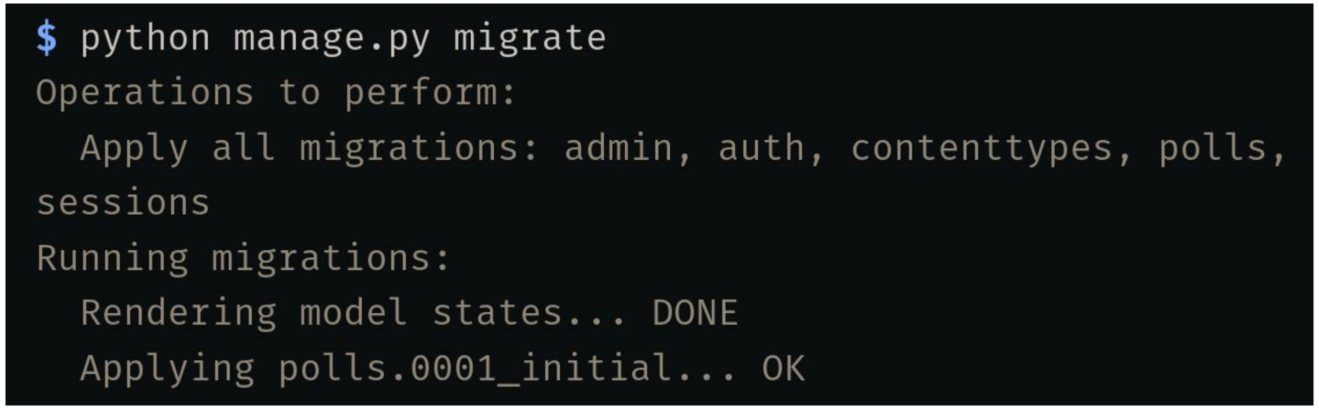
```
from django.db import migrations, transaction

class Migration(migrations.Migration):
  dependencies = [
    (student, '0001_initial'),
  ]
  def generate_data(apps, schema_editor):
    from student.models import Student
    test_data = [ (00001, 'testUser1', 0524677545, 'M'),  (00002, 'testUser2', 0524755849, 'F')]
    with transaction.atomic():
      for (student_id, nickname, phone_number, gender) in test_data:
        Student.create(student_id=student_id, nickname=username,
                           phone_number=phone_number,  gender=gender)
  operations = [
    migrations.RunPython(generate_data),
  ]
```

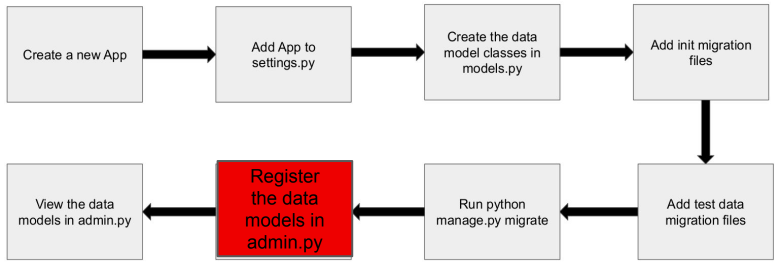
step by step - apply migrations


Create a new App
Add App to settings.py
Create the data model classes in models.py
Add init migration files
Add test data migration files
Run python manage.py migrations
Register the data models in admin.py
View the data models in admin.py

- Letting Django know you've made changes to your models (or made new ones) and those be stored as a migration.
- Run the migrations and manage the database schema automatically:
`$ python manage.py migrate`



```
$ python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, polls,
sessions
Running migrations:
  Rendering model states... DONE
  Applying polls.0001_initial... OK
```

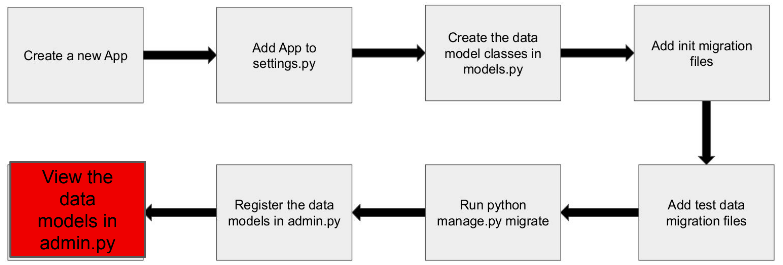

Create a new App
Add App to settings.py
Create the data model classes in models.py
Add init migration files
Add test data migration files
Run python manage.py migrate
Register the data models in admin.py
View the data models in admin.py

step by step - register the models

- Have your model show up in Django's admin interface.

```
from django.contrib import admin
from .models import Student

admin.site.register(Student)
```


Create a new App
Add App to settings.py
Create the data model classes in models.py
Add init migration files
Add test data migration files
Run python manage.py migrate
Register the data models in admin.py
View the data models in admin.py

Django ORM - Admin panel

- Reads metadata from your models to provide a quick, model-centric interface to manage content on your site.
- Let us watch our DB in the admin panel in real-time using admin.com as the admin user.

1. Create an admin user: `django-admin createsuperuser`
2. Enter the admin site (/admin/ endpoint by default)

More info here:
https://docs.djangoproject.com/en/4.0/intro/tutorial02/#creating-an-admin-user

Django ORM - Playing with objects

- Adding new records to the table ➝ creating an object

```
student = Student(user=User.objects.create_user(…), student_id=”00003”,  nickname=”oamsalem”,
                              phone_number=”0544866302”, gender=”M”)
student.user.save()
student.save()
```

- Pull information from the table ➝ call an object’s property

```
my_student = Student.objects.filter(gender=”M”)
print(my_student[0].student_id)
```

- Modify a record & Remove a record

```
my_student.nickname = “other nickname”
my_student.save()
my_student.delete()
```

Questions?

Thank you

Red Hat is the world's leading provider of enterprise open source software solutions. Award-winning support, training, and consulting services make Red Hat a trusted adviser to the Fortune 500.

linkedin.com/company/red-hat

facebook.com/redhatinc

youtube.com/user/RedHatVideos

twitter.com/RedHat