# Django Basics

|  |  |  |
| --- | --- | --- |
| Topic | Command | Info |
| SQL Create Table | CREATE TABLE tablename (  id INTEGER PRIMARY KEY AUTOINCREMENT, origin TEXT NOT NULL, …); | First entry: id column, second entry: type of data, Primary key = unique identifier, NOT NULL🡪cannot be 0.  Possible cosntraints: NOT NULL, PRIMARY KEY, UNIQUE, CHECK etc. |
| SQL add data to TABLE | INSERT INTO tablename  (coulmnname1, coulmname2, columnname3)  VALUES (“Berlin”, “Germany”, 200); | SQL command to add data to a table |
| SQL QUERIES | SELECT col1, col2 FROM tablename  FROM tablename  WHERE col1=”Berlin  ORDER BY something; | WHERE lable like “%a%” -> % means a number of signs |
| SQLite3 create db | Touch mydb.sql  Sqlite3 mydb.sql | Opens the SQLite cmd and start writing in the selected db.  .tables shows all available tables  Better formatting: .mode columns, .headers yes |
| SQL UPDATE | Update tablename  SET location=”Berlin”  WHERE district=”Kreuzberg |  |
| SQL DELETE | DELETE FROM tablename  WHERE location=”Berlin |  |
| SQL join Tables | SELECT name, origin, destination  FROM flights\_table  JOIN passengers\_table  ON passengers.flight\_id=flights.id | Joining tables allows for an intelligent table design in terms of one-to-many or many-to-many relations. We design independent tables for a single structure of data, i.e. person, flight, airport and design the relationship between them using association tables like person\_id|flight\_id which are created using the JOIN command.  There are 4 types of JOINS:  - JOIN/INNER JOIN -> joins two tables and gives out all entries where both tables have values  - LEFT/RIGHT OUTER JOIN -> joins two tables and shows all entries of the LEFT/RIGHT table even if no match |
| SQL Injection | Username=”hack”—" | SQL Queries can be bypassed depending on the syntax. Django however stops this. Here – will be interpreted as a comment in SQL |
| SQL Race Conditions |  | It can happen that multiple requests are executed in parallel and that |
| Django Models | Models.py | Django uses the models defined here to interact with the SQL database defining a python class that represents the way the data should be stored into a database. Django will figure out what SQL syntax to use to create and manipulate the table. In general we have one Model per Table |
| Django Migrations | 1. Python manage.py makemigrations Name 2. Python mange.py migrate | Django uses migrations to update the database based on changes made in modely.py. It is a two step process where 1) we create migrations - the changes that we want to apply to the database- and afterwards apply |
| Django SQL Shell | python manage.py shell  f = Flight(from=”a”,to=”b”,dur=20)  f.save | Used to interact with the Database using python code – SQL code is not required –  First we create a database object and then we save it with .save |
| Django table print | def \_\_str\_\_(self): print(someformat) | String representation of the object called in the print(object) case |
| Django select \* | Query = dbModel.objects.all() | Retuns a query list of containing all objects |
| Django objects | dbClass.objects.filter(destination=”Berlin”)  dbClass.objects.get(destination=”Berlin”)  dbClass.objects.all()  dbClass.objects.exclude(destination=”Berlin”).all(() | Filter returns a list which matches the query. Using .first() on top gets the top entry  Get returns the EXPECTED single element from the query. It will raise alarms if more than one element exists |
| Django db entry values | Entry = Query.first();  Entry.id, Entry.origin etc., Entry.delete() |  |
| Django join tables | models.ForeignKey(ReferrenceTable, on\_delete=models.CASCADE,  related\_name = “departures” | Table linking is done in the model class using the ForeignKey class.  Models.CASCADE 🡪 if I delete the reference table, also delete the entries which reference it in this table.  Related name is used for the reverse reference, i.e. if my models is Flight and entry is origin which links to Airport then the reverse is all the airports who with a specific origin – all departures from this airport. |
| Django Admin Create | Python manage.py createsuperuser | Django Admin app is used to define create and manipulate models. For that we first need a admin account |
| Django Admin App | In admin.py admin.site.register(ClassName)  class MyClassAdmin(admin.ModelAdmin) | In admin.py we write what should be manipulable in the admin app, i.e. classes. In admin.py we can also choose customizable admin display views by designing a new class and choosing different settings |
| Many-To-Many | Models.ManyToManyField(ReferenceClass, blank=True, related\_name=”related”) | Implements a Many to Many relationship between table 🡪 one entry can have multiple references and one reference can have multiple entries. Blank allows the field to be Empty, i.e. passenger has no flights at all. To access the relate name you use ReferenceClass.related.all() |
| Authent. request | If not request.user.isauthenticated: | Check if user is signed in when visiting the page or not |
| Authenticate credentials | In views:py  Authenticate(request, username=user, password=pw)  , login, logout | Authenticate checks if username and password are valid and returns a username or None. Continue with if user is not None: … |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |