Week 4: SQL, Tables and Migrations

1. SQL

Syntax for creating tables ->

CREATE TABLE flights (

Id INTEGER PRIMARY KEY AUTOINCREMENT,

Origin TEXT NOT NULL,

DESTINATION TEXT NOT NULL,

DURATION INTEGER NOT NULL

);

Types of Constraints

* Check
* Default
* Not Null
* Primary Key
* Unique

INSERT INTO flights (origin, destination, duration) VALUES (“New York”, “London”, 415);

1. SQL Queries
2. Retrieve Data

SELECT \* FROM flights -> \* means everything

SELECT \* FROM flights WHERE id = 3; or SELECT \* FROM flights WHERE origin = “New York”

SELECT \* FROM flights WHERE duration > 500 AND destination = “Paris”;

SELECT \* FROM flights WHERE origin IN (“New York”, “Lima”);

**More Complicated Searches in SQL**

What if you want to find a partial match for example you want origin in flights where it contains the letter a

SELECT \* FROM flights WHERE origin LIKE “%a%” -> % is a wildcard character in SQL where it means match any character. The LIKE command tells SQL we are not looking for an exact match

1. Update Data

UPDATE flights SET duration = 430 WHERE origin = “New York” AND destination = “London”;

1. DELETE Data

DELETE FROM flights WHERE origin = “Tokyo”;

1. Useful Clauses

* LIMIT
* ORDER BY
* GROUP BY
* HAVING

1. Foreign Keys

Tables in SQL have a tendency to spiral out of control and increase in size because we have too much data. Therefore, impt for us to design the right tables to have in any application. However, these tables have to be linked to one another and that is through foreign keys.

Foreign keys are any column that contains a column to another table. This column would be the foreign key. Splitting tables therefore make our data more organized and joining them would make it easy to get the right data

1. Relationships
2. One to Many

1 to many relationship is used to describe the relationship where 1 row in a SQL table can be linked to many rows in another table.

1. Many to Many

Many to many refers to the idea where many rows in 1 table can be linked to many rows in another table.

1. Example

How to implement Many To Many

If you have a table called people storing first name, last name and id

Another table called locations containing id, destination and airport names

Another table called flights, which contains origin, destination and id. Note that origin and destination here can be foreign keys linking to locations table

A new table called passengers which contains 2 columns, person\_id and flight\_id. It is an association table which associates rows from 1 table to rows form another table. This is called a join table.

Summary

* Table for airports (location) -> Storing location details
* Table for flights -> Storing flight details. Any flight maps to 2 different airports one for origin one for destination. Any airport might appear for multiple flights. This is an example of a 1 to many relationship. Where 1 row of airports can be seen in multiple rows of flights.
* Table of persons -> Storing passenger details. Each row is meant to represent 1 person
* Table of passengers -> Storing relationship between persons and flights i.e. tracking which passengers goes to what flight. This is called an association table as it will contain only 2 columns, person\_id and flight\_id. This is an example of a many to many relationship between person table and flight table because 1 person can take many flights.

1. JOIN

The above design in summary demonstrates why JOIN is impt as it now helps you visualize multiple tables together. Data can be held in multiple different tables and therefore the use of a single table is unintuitive. Look at the passengers table where it only contains person\_id and flight\_id.

1. Syntax

Assuming that we only have 2 tables, flights and passengers and every passenger is only linked to 1 flight.

SELECT first, origin, destination FROM flights JOIN passengers ON passengers.flight\_id = flights.id;

The output of a JOIN statement is a completely new table. To create that table we would need to first indicate the columns of the first table that we want in the new table. That would be first, origin and destination.

Next, we would need to show how these 2 tables are related to one another. In our case, the flight\_id column in passenger means the same as flights.id in flight table.

1. Types of Join

**Inner JOIN**

SQL will table 2 tables and compare them and will only match them if their columns are a match.

**Left Outer Join**

**Right Outer Join**

**Full outer join**

1. Index

Sometimes we would need to make queries of our databases more efficient because we have to lookup databases often. We use index to achieve this. An index of a database works the same way an index in the library does. For example, if library organizes books by their title alphabetically, then the titles act as their index. In SQL, index acts the same way. We have to define 1 column in the table to be an index and when we search the column via this specific index, that query will be faster.

Note: Creating an index by a column requires more memory as it is creating a unique data structure to create this. The exact reasons of how Index works are not explained in CS50 must find yourself.

Syntax

CREATE INDEX name\_index ON passengers (last) -> This means create an index called name\_index on the passengers last name.

1. SQL Injection Attacks

When dealing with databases we must be cognizant of how hostile actors can attack them. Most commonly this is via SQL injection attacks. This occurs when a form posts something and the backend uses those inputs to execute a SQL query. Hackers could try to comment out authentication via - - as that stands for commenting out things in SQL.

Best way to avoid this is to just use an ORM and let the ORM handle security like with Django.

1. Race Conditions

Basically, occurs when a multi-threaded application try to query the same database simultaneously with different threads. To solve we can either lock the database or do something else.

1. How Django use Models
2. Migrations

Django gives you the ability to model SQL tables as Python classes within the models.py file. However, when you create python classes, nothing is created within the databases yet. We still need 1 more step to change these Python classes into SQL tables. This process is called migrations.

Command for creating migrations are:

python manage.py makemigrations -> This tells Django to create instructions to implement the changes done to models.py. I.E. now there are changes to models.py please make those changes legible to SQL

Once you do this, steps for migrations can be found in migrations.py

Python manage.py migrate -> actually starts the migrations. The default without any settings is that a SQLlite3 db file is created within the models.py where changes was made.

1. Interacting with SQlite3 files

We can enter the Django shell to write Python code to interact with SQLite3 files that have been created. Alternatively, we can use a GUI like DB browser or write actual SQL commands in Python. However, Django shell contains an in-built ORM that would effectively change Python code into SQL commands using Django’s own library.

Enter Django file with command

Python manage.py shell -> Use this cause has access to the ORM

ORM Commands

**Create row in table**

F = Flight(origin = “New York”, destination = “London”, duration = 415)

f.save()

**Query all rows in table**

Flights = Flights.objects.all.() -> Would receive a QuerySet

**Pop first row**

Flight = Flights.first()

**Get specific row**

Flights = Airport.objects.filter(city=”New York”).first() -> First means pop the first result in Query Set. If we only have 1 result then we will always get the result we want.

Alternatively can also do

Flights = Airport.objects.get(city=”New York”) \_> We do this when we know that there is only 1 result

1. Django Admin Page

To help use add things in models without us going into the shell and manually adding rows, Django has the admin panel which is a GUI that helps us do this.

Steps to do so

1. Create super user account in the application. Do this by running python manage.py createsuperuser
2. Add models to the admin.py file in app
3. How to implement the various relationships

**One to One**

Use a foreign key to the referenced table. You must also put a unique constraint on the foreign key column to prevent multiple rows in the child table from relating to the same row in the referenced (parent) table.

I.E. If you have a student table and an address table. The address table should contain a foreign key to the student\_id. Unique constraint should be on. The parent table is the student table and child table is the address table.

**One-To-Many**

Use foreign key on the many side of the relationship linking back to the one side.

**Many-To-Many**

1. SQL

Use a junction table.

student: student\_id, first\_name, last\_name

classes: class\_id, name, teacher\_id

student\_classes: class\_id, student\_id # the junction table

Example Queries

-- Getting all students for a class:

SELECT s.student\_id, last\_name

FROM student\_classes sc

INNER JOIN students s ON s.student\_id = sc.student\_id

WHERE sc.class\_id = X

-- Getting all classes for a student:

SELECT c.class\_id, name

FROM student\_classes sc

INNER JOIN classes c ON c.class\_id = sc.class\_id

WHERE sc.student\_id = Y

1. Django

Use ManyToManyField to define a many-to-many relationship. You can use this ManyToMany field on either table. Can also create a recursive relationship i.e. an object with a many-to-many r/s to itself and relationshipos to models not yet defined.

Suggested that the nbame of a ManyToManyFIeld be plural.

Doesn’t matter which model has ManyTOManyField but you should only put it on one not both.

Rule of thumb is ManyToManyField instnaces should go in the object that is going to be edited on a form.

Extra Fields on many-to-many relationships

If you need to associate data with the relationships between 2 models – such as an application tracking the musical group which musician belongs to. A person can be a member of multiple groups and a group can have multiple members. However, there are other details you might want to collect such as the date at which a person joins a group.

You can explicitly specify the model that will be used to govern the many-to-many relationship i.e. like building an associate table in SQL explicitly. You can then put extra fields on this intermediate model. To associate the intermediate model with the ManyToManyField, you need to add a through argument to point to the model that will act as intermediary. I.E. In the Group table you must add a through="”ameOfAssociateTable"

1. Additional Model Field Info in Django

**Meta**

You can add metadata to models using an inner class. Metadata is anything that is not a field such as database table names, ordering options, and human-readable singular and plural names.

**Blank vs. Null**

If null is True, Django wil store empty values as Null. Default is False.

Don’t use null on string-based fields such as Char or Text. If string-based field has null=True, then it has 2 possible values for no data. Null or empty string. This is redundant and Django convention is to use empty string.

If blank=-True, the field is allowed to be blank. Default is False. Null is purely database related while Blank is validation related. If blank-True, form validation will allow entry of an empty value. If blank=False, field is required.

When creating fields in Django very common for null=true and blank=true to come together. Rationale is for most fields, if you submit an empty field, Django would save it as Null. Therefore, if you are allowing a field to be blank you would have to accept that it would be null as well.

Exception is Text or Char field where empty fields would be submitted as empty strings.

**Personal Notes**

**Request object in Django**

1. User

Request.user in Django refers to the standard User model. This means that just calling it like request.user would get the username but you can get other attributes from the model such as request.user.id to get the id.

1. POST

Request.POST would be a queryDictionary that contains all the information that the client sends in a key-value pair format. This includes which button was clicked as only buttons that are clicked would be included in the POST request.

**Displaying Error Messages with Forms**

Django form fields have input validation errors. Sometimes to show lack of validation, best way is to redirect user back to the html page with the form and this helps you to display the error message.

Example:

1. In views.py

If request.method == “POST”:

Form = formName(request.POST) => Gives form data from posted form including validations

If form.is\_valid():

Do something …

Else:

Return render(request, <html\_page>, { “form”: form})

1. In template
2. {% if form.errors %}
3. {% for field in form %}
4. {% for error in field.errors %}
5. <div class="alert alert-danger">
6. <strong>{{ error | escape }}</strong>
7. </div>
8. {% endfor %}
9. {% endfor %}
10. {% else %}

**Handling Templates with Multiple Forms or Buttons**

1. Multiple Buttons

Buttons will only submit their data if they are clicked on. Button name and value will be stored as a key-value pair in request.POST. Request.POST is a queryDict. You can also add name to the buttons on template to make them identifiable. Then in views.py you just need to check whether which button was the one submitted in the request sent by client.

Example:

If request.method == “POST:

Form = <whateverForm>(request.POST)

If “button\_name” in request.POST.:

Do something ….

Elif “another\_button\_name” in request.POST:

Do Something

Note: It is far better practice to have each button surrounded by their own form. This is because if you use is\_valid() method in view.py it means that to submit all buttons, they must pass the form’s validation process. This would cause problems as it leads to an unintuitive user interface.

Therefore, multiple forms should have multiple buttons. The distinction is whether they should be sent to multiple routes or a single route. This depends on the business logic in question but in general it is important to ensure readability and 1 view should not have too many if statements inside which is required when buttons are there.

**Where to put logic?**

Avoid putting logic in template or views.py. So either have all of them in separate files OR have “fat” models although fat models cause performance issues and is bad for large apps.

**How to update ManyToManyField?**

1. Need to create instance of row that is to be updated
2. Then you need to create item to be added to that row
3. Then you need to save

Example:

            listing = Listing.objects.get(listing\_id)

            watchlist\_item = Listing.objects.create(watchers=request.user.id)

            listing.watchers.add(watchlist\_item)

Here, listing is the parent row, watch\_list item to be added. Watchers is the name on the listing model that is the ManyToMany field.

**How to retrieve Objects?**

**Messages Framework (How to send pop up messages to users)**

* Achieved using messaging framework in Django
* Don’t need to explicitly pass messages into Django template via dictionary it is automatically done
* Still need to style alert with bootstrap

**How Django read filters for ManyToManyField or a reverse ForeignKey**

When we query a table we often also want to include more than 1 filter. Django has 2 main methods to do this depending on what results you are looking for.

Example 1

Blog.objects.filter(entry\_headline\_contains=”Lennon”, entry\_pub\_date\_year=”2008”)

Example 2

Blog.objects.filter(entry\_headline\_contains=”Lennon”).filter(entry\_pub\_date\_year=”2008”)

Example 1 would return any row in Blog that contains a headline Lennon AND has a publishing date of 2008.

Example 2 would return any row in Blog that contains a headline Lennon OR a publishing date of 2008.

<https://docs.djangoproject.com/en/3.2/topics/db/queries/>

Find Spanning multi-valued relationships

**Formulating Tables Tips and Tricks**

**Error List**

* Cannot unpack non-iterable int object
  + Django speak for saying you need to add id=\*