Setting up Relation using Foreign Key Column :-

⇒ Each post must be linked to the user (who created it). This is done using Foreing Keys.

⇒ Posts → Users will have a Many → One Relation

Thus, in Posts table, we create a foreign key column, named user_id. Can be done in 2 ways:-

i) Directly, using Postgres GUI.
ii) Using sqlalchemy ORM, using Python.

i) Using Postgres GUI :- Create column, giving NOT NULL constraint, and same data type as in Users table in Post table.

⇒ Set foreign key constraint.
⇒ By convention, name of foreign key should be posts_users_fkey [only for CLI purpose]
⇒ Now, in it, set the user_id as Local Column
⇒ In References, select public.users table.
⇒ In Referencing, select the id column, i.e, the column in Users table that will be the unique column in other (Users) table.

⇒ Then, under the Actions tab, select the following:-
On update - NO ACTION.
On delete - CASCADE.

CASCADE actually means when the user is deleted (the one which has ONE in relation), then all the post (Post table records) [the table in which foreign key exists] will get deleted, which was related to the user being deleted.

Other options include :- RESTRICT, NULL, DEFAULT.

⇒ Hit Save to make the Foreign key.

08:07:30

ii) Using sqlalchemy ORM :-

Within the Post model class, we need to create a new static variable, that will correspond to the user_id foreign key column in posts table.

```
from sqlalchemy import ForeignKey

class Post(Base):
    id = ...
    title = ....
    ---------
    owner_id = Column(Integer, ForeignKey("users.id", ondelete="CASCADE"), nullable=False)
```

Foreign Key column

should match exactly with the datatype of the column it is referencing to

"users.id" → other table name (not the class name)

"users.id" → exact name of the column which this FKey is referencing to

ondelete="CASCADE" → on delete action

Updation in our API accordingly:

08:14:00

In the response schema for posts, we need to add the foreign key column owner_id data, so that frontend can know to which user the created or fetched posts belongs to.

Though we need owner_id column data while creating post to, but we will not add this column in the request schema PostCreate, as the owner id will be obtained by extracting current user data from the token.

```
class PostBase(BaseModel):
    title: str
    content: str
    published: bool = True
```

```
class PostCreate(PostBase):
    id: int
    created_at: datetime
    owner_id: int
    class Config:
        orm_mode = True
```

8:18:00

While creating (or updating Post), in the SQL query statement we must add the owner id as the current user id, alongwith the post record.

Thus, in the path operation for creating posts,

we just need to make the following addition :-

```
...
async def create_post(...):
    new_post = models.Post(**post.dict())
    new_post.owner_id = current_user.id
```

This object is then added to the database

object obtained from access token verification

8:20:00

⇒ We need to apply a logic in the update_post and delete_post path operation, such that a user can only be able to update and delete posts that he/she created.

user object obtained by validating JWT

The logic is as follow :-

```
...
fetched_post.owner_id != current_user.id:
    raise HTTPException(status_code = 403,
    ...detail = "Not authorised to perform req. operation")
...
```

post fetched from database

8:34:00

⇒ Performing SQL join using Sqlalchemy :-

* When retrieving posts (single or multiple) we are returning owner_id only, till now.
But, when designing frontend, it needs to show the associated user name (and details), with the posts, as that is actually how social media apps work.

For this, we again need to perform another SQL query to fetch creator's name, with the associated user id, or else perform SQL join.

* In sqlalchemy, we have a way to do this simply, without any SQL join or changes in database tables.
We just need to set another variable in the Post model class as follows :-

```
class Post(Base):
    id = Column(...)
    owner_id = Column(...)
    owner = relationship("User")
```

sqlalchemy model class

from sqlalchemy.orm import relationship

owner: contain an object of other class type which will act as join, as in SQL, table has column from other table

"User": model class name of that class with whom this table is related with, using foreign key constraint

2) Thus, the relationship() method takes the name of model class, which represents the table with which we want to have SQL join (the table must be linked with a Foreign Key)

It will return an object of the other class type (User here), and will be assigned to the member static variable (owner here).

3) Thus, when a Post class object will be fetched, it will contain the details of associated User class object, in the owner variable.

4) We need to change the PostResponse schema accordingly, so that the owner variable is also returned :-

```
class PostResponse(PostBase):
    id: int
    created_at: datetime
    owner_id: int
    owner: UserResponse
    - -
    - -
```

UserResponse → Pydantic model using which User details are being sent in the response.