Voting / Likes System :-

9:21:20

- Users should be able to like a post only once
- Retrieving posts should fetch total no. of likes

- Like system is one-direction, i.e, we cannot downvote a post that we liked

youtube.com is now full screen Exit Full Screen (Esc)

Vote Model

- Column referencing post id
- Column refererencing id of user who liked the post
- A user should only be able to like a post once so this means we need to ensure every post_id/voter_id is a unique combination

Post_id	User_id
12	4
28	9
55	2
12	9
18	4
55	2

We cannot have a user like a post twice. Duplicate of row 3

Composite Key :-

- Composite Key is primary key that spans multiple columns.
- ∵ Primary Keys are unique, so by this, we can implement it, to restrict duplicate Post_id and User_id combination.

9:26:30

- In PostgreSQL (GUI), to set composite key, we just need to select multiple columns as Primary Key. And then, these primary key columns will be set as Foreign Key, referenced to other tables Primary Key.

=> We will make votes table (that will count votes by users against posts) where column post_id & user_id will exist, where both will form the composite key, and will reference the posts & users table.

To make this table, we will make a model class (in sqlalchemy), named Vote.

```
class Vote(Base):
    __tablename__ = "votes"
    user_id = Column(Integer, ForeignKey("users.id",
        ... ondelete="CASCADE"), primary_key = True)
    post_id = Column(Integer, ForeignKey("posts.id",
        ...ondelete = "CASCADE"), primary_key= True)
```

Vote Route

- Path will be at "/vote"
- The user id will be extracted from the JWT token
- The body will contain the id of the post the user is voting on as well as the direction of the vote.

```
{
    post_id: 1432
    vote_dir: 0
}
```

- A vote direction of 1 means we want to add a vote, a direction of 0 means we want to delete a vote.

=> So, with the above requirements, the path operation for the voting system will be as follow :-

```python
@router.post("/", status_code = status.HTTP_201_CREATED )
async def vote( vote: schemas.Vote, db: Session = Depends(database.get_db), current_user = Depends(oauth2.get_current_user) ):
    vote_query = db.query( models.Vote ).filter(
            models.Vote.user_id == current_user.id,
            models.Vote.post_id == vote.post_id)
    existing_vote = vote_query.first()
    if vote.dir == True:
        # Post is being liked/voted by user -- number of votes increases
        if not existing_vote:
            # If the user post combination does not exist, then vote will be added for the post
            vote_to_be_added = models.Vote( post_id=vote.post_id, user_id=current_user.id )
            db.add( vote_to_be_added )
            db.commit()
            db.refresh( vote_to_be_added )
            return {"vote": vote_to_be_added, "detail": "Successfully voted for the post!"}
        else:
            # Vote exists, and still voting with dir 1 is attempted
            raise HTTPException( status_code = status.HTTP_409_CONFLICT,
                detail = f"User with user id = {current_user.id} has already voted post with id = {vote.post_id}." )
    else:
        # Vote is being removed from the post by user -- number of votes decreases
        if existing_vote:
            # Vote exists, thus needs to be deleted.
            vote_query.delete( synchronize_session = False )
            db.commit()
            return {"detail": "Successfully removed the vote for the post!"}
        else:
            # Vote does not exists, then too attempted to be deleted.
            raise HTTPException( status_code = status.HTTP_404_NOT_FOUND,
                detail = "Vote does not exists!" )
```