

57. Retrieve the creator of a post

[#typeorm](#) [#entity](#) [#graphql](#) [#onetomany](#) [#manytoone](#) [#backend](#) [#frontend](#) [#query](#) [#resolver](#) [#sql](#) [#postgresql](#)

Expose the creator column to the client

- We already added `creator` column to the `Post` entity in [40. Implement Post.creator and User.posts columns](#)
- We were not exposing the `creator` column to the client (there's no `@Field()` attribute). We add the `@Field()` attribute to it
- Since `creator` of type `User` and that's an `@ObjectType()` GraphQL automatically knows which fields `creator` has

/entities/Post.ts

```
@Field()
@Column()
creatorId!: number;

@Field()
@ManyToOne(() => User, (user) => user.posts)
creator: User;
```

Update posts query in Resolver

- We implemented an `SQL` query to pull `posts` from the `database` in [51. Pagination for Posts - Resolver post.ts / Query_posts\(\)](#)
- We update this `SQL` query with an `inner join`, to provide the `creator` field by pulling a `user` from `user` table via `creatorId`
- typeorm.io/#/select-query-builder/joining-relations

/resolvers/post.ts

```
const qb = getConnection()
  .getRepository(Post)
  .createQueryBuilder("p")
  .innerJoinAndSelect("p.creator", "u", "u.id = p.creatorId")
```


```

.orderBy("p.createdAt", "DESC") // mind the double quotes '"' ... '"'
.take(realLimitPlusOne);

if (cursor) {
  qb.where("p.createdAt < :cursor", { cursor: new Date(parseInt(cursor)) });
}

```

Alternative raw SQL query implementation

- We could also do this with [raw sql query](#) as follows
- The drawback of this method is that the [sql](#) query will always return the [creator](#), even if the end user doesn't need it and the [graphql query](#)  does not ask for it
- [Note that](#) `json_build_object()` is a [PostgreSQL](#) featurer

/resolvers/post.ts

```

const replacements: any[] = [realLimitPlusOne];

if (cursor) {
  replacements.push(new Date(parseInt(cursor)));
}

const posts = await getConnection().query(
  `
  select p.*,
  json_build_object(
    'id', u.id,
    'username', u.username,
    'email', u.email
  ) creator
  from post p
  inner join public.user u on u.id = p."creatorId"
  ${cursor ? `where p."createdAt" < $2` : ""}
  order by p."createdAt" DESC
  limit $1
  `,
  replacements
);

```

Update GraphQL query

- Now we update the `posts` query in `graphql` based on the changes we made on the `backend`

`/graphql/queries/posts.graphql`

```
query Posts($limit: Int!, $cursor: String) {  
  posts(limit: $limit, cursor: $cursor) {  
    hasMore  
    posts {  
      id  
      createdAt  
      updatedAt  
      title  
      textSnippet  
      creator {  
        id  
        username  
        email  
      }  
    }  
  }  
}
```

- Run codegen to generate the `TypeScript` code for `graphql`

```
yarn gen
```

- and now we have the updated `usePostsQuery()` hook in `/generated/graphql.tsx`