TypeORM Operations

Course Code: CSC 4182 Course Title: Advanced Programming In Web Technologies

Dept. of Computer Science
Faculty of Science and Technology

Lecture No:	**2**	**Week No:**	**04**	**Semester:**	
Lecturer:	*Sazzad Hossain; sazzad@aiub.edu*				

Lecture Outline

- ✓ Entity
- ✓ TypeORM Decorators
- ✓ CRUD Operation
- ✓ TypeORM Find Option
- ✓ TypeORM One to One
- ✓ TypeORM One to Many
- ✓ TypeORM Many to Many

Entity

Entity is a **class** that maps to a database table.

Entity can be created by defining a new class with @Entity() class.

```
import { Entity, PrimaryGeneratedColumn, Column } from "typeorm"

@Entity() // entity representing a database table
export class User {
    @PrimaryGeneratedColumn() // Defines the primary key column of the table
    id: number; //automatically generated Unique identifier

    @Column() // database column
    firstName: string; // column name

    @Column() // database column
    lastName: string; // column name

    @Column() // database column
    isActive: boolean; // column name

     @Column() // database column
     @Generated('uuid') // automatically generated using the UUID strategy
     code: string;
```

Entity

@Column decorator and specify the type property with the desired data type.

```
@Column("int")
id: number;

@Column({name: “fullname" type: "int" })
name: string;

@Column({ type: 'varchar', length: 50 })
name: string;
```

The supported data type can be found here;
https://typeorm.io/entities#column-types-for-postgres

TypeORM Decorators

1.@**Entity**: Defines a class as an entity representing a database table.
2.@**PrimaryGeneratedColumn**: Specifies a column as the primary key and determines its generation strategy.
3.@**Column**: Specifies a property as a column in the database table.
4.@**CreateDateColumn**: Automatically sets the creation date for a column.
5.@**UpdateDateColumn**: Automatically updates the column with the current date on each update.
7.@**OneToMany**: Establishes a one-to-many relationship between entities.
8.@**ManyToOne**: Establishes a many-to-one relationship between entities.
9.@**ManyToMany**: Establishes a many-to-many relationship between entities.
10.@**JoinColumn**: Specifies the join column for a relationship.
11.@**JoinTable**: Specifies the join table for a many-to-many relationship.
12.@**Index**: Creates an index on one or multiple columns.
13.@**Unique**: Ensures that the column values are unique.
14.@**PrimaryColumn**: Specifies a column as a primary key without auto-generation.
15.@**Generated**: Specifies that a column is generated with a custom value or expression.

Custom Autogenerate


```
import { Entity, Column, PrimaryGeneratedColumn, BeforeInsert } from
'typeorm';

@Entity()
export class User {
  @PrimaryGeneratedColumn()
  id: number;

  @Column()
  name: string;

  @Column()
  randomNumber: number;

  @BeforeInsert()
  generateRandomNumber() {
    this.randomNumber = Math.floor(Math.random() * 1000);
  }
}
```

CRUD Operation

To perform CRUD operations using TypeORM in NestJS, need to Inject the repository into the service:

- Use the **@InjectRepository** or **@Inject** decorator to inject the repository for the entity into your service class.
- Example: constructor(@InjectRepository(**User**) private userRepository: Repository<**User**>) {}
- **User** is the entity class

CRUD Operation


```
import { Injectable } from '@nestjs/common';
import { InjectRepository } from '@nestjs/typeorm';
import { Repository } from 'typeorm';
import { User } from './user.entity'; // change this to your entity class

@Injectable()
export class UserService {
  constructor(@InjectRepository(User) private userRepository: Repository<User>) {}
// userRepository is the local repository
  async createUser(user: User): Promise<User> {
    return this.userRepository.save(user);
  }
  async getAllUsers(): Promise<User[]> {
    return this.userRepository.find();
  }
  async getUserById(id: number): Promise<User> {
    return this.userRepository.findOneBy({id:id});
  }
  async updateUser(id: number, updatedUser: User): Promise<User> {
    await this.userRepository.update(id, updatedUser);
    return this.userRepository.findOneBy({id:id});   }
  async deleteUser(id: number): Promise<void> {
    await this.userRepository.delete(id);
  }
}
```

CRUD Operation


```
// Create a new user
async createUser(user: User): Promise<User> {
    return this.userRepository.save(user);
 }

// Get all users
async getAllUsers(): Promise<User[]> {
    return this.userRepository.find();
 }

// Get a user by ID
async getUserById(id: number): Promise<User> {
    return this.userRepository.findOneBy({id:id});
 }
```

CRUD Operation


```
 // Update a user
 async updateUser(id: number, updatedUser: User): Promise<User> {
    // Update the user with the provided ID using the updatedUser
data
    await this.userRepository.update(id, updatedUser);
    // Return the updated user
    return this.userRepository.findOneBy({id:id});
  }

 // Delete a user
 async deleteUser(id: number): Promise<void> {
    // Delete the user with the provided ID
    await this.userRepository.delete(id);
  }
```

TypeORM Find Option

Repository and manager .find* methods accept special options

```
userRepository.find({
    select: { //select - indicates which properties of the main
object must be selected
        firstName: true,
        lastName: true,
    },
})
```

This will execute following query:

```
SELECT "firstName", "lastName" FROM "user"
```

TypeORM Find Option

OR Operator

```
userRepository.find({
    where: [
        { firstName: "Timber"},
        { firstName: "Stan" },
    ],
})
```

This will execute following query:

```
SELECT * FROM "user" WHERE ("firstName" = 'Timber') OR
("firstName" = 'Stan')
```

TypeORM Find Option

AND Operator

```
userRepository.find({
    where:
        { firstName: "Timber",
         firstName: "Stan" },
    ,
})
```

This will execute following query:

```
SELECT * FROM "user" WHERE ("firstName" = 'Timber’) AND
("firstName" = 'Stan')
```

TypeORM Find Option

Order

```
userRepository.find({ //order - selection order.
    order: {
        name: "ASC",
        id: "DESC",
    },
})
```

This will execute following query:

```
SELECT * FROM "user"
ORDER BY "name" ASC, "id" DESC
```

TypeORM Find Option

LIKE Operator

```
userRepository.find({
    where: {
      name: Like('John%’), //Like operator is used to
perform a partial match search
    },
  });
```

More find options can be found here;
https://typeorm.io/find-options

TypeORM One to One

Suppose two entities: **User** and **UserProfile**. They have a one-to-one relationship

```
// User.entity.ts
import { Entity, PrimaryGeneratedColumn, Column, OneToOne, JoinColumn } from
'typeorm';
import { UserProfile } from './UserProfile.entity';

@Entity()
export class User {
  @PrimaryGeneratedColumn()
  id: number;

  @Column()
  name: string;

  @OneToOne(() => UserProfile, userProfile => userProfile.user, { cascade: true
})
  @JoinColumn()
  userProfile: UserProfile;
}
```

TypeORM One to One

- **@OneToOne** decorator is used to define a one-to-one relationship between entities
- **() => UserProfile:** This parameter specifies the target entity that the current entity (User) is being related to. In this case, it's the UserProfile entity.
- **userProfile => userProfile.user:** This parameter specifies the inverse side of the relationship. It defines the property on the UserProfile entity that references the User entity. In this example, it refers to the user property in the UserProfile entity.
- **{ cascade: true }:** This parameter specifies the cascade behavior for the relationship. When cascade is set to true, it means that any changes made to the User entity will be cascaded to the associated UserProfile entity. For example, if a User is deleted, the associated UserProfile will also be deleted.
- **@JoinColumn()** decorator specifies the join column in the database table.

TypeORM One to One


```typescript
// UserProfile.entity.ts
import { Entity, PrimaryGeneratedColumn, Column, OneToOne,
JoinColumn } from 'typeorm';
import { User } from './User.entity';

@Entity()
export class UserProfile {
  @PrimaryGeneratedColumn()
  id: number;

  @Column()
  age: number;

  @OneToOne(() => User, user => user.userProfile)
  user: User;
}
```

TypeORM One to One

- **@OneToOne():** Decorator to define a one-to-one relationship between the current entity and the User entity. It takes two parameters:
  - **() => User** specifies the target entity as User. It uses an arrow function to reference the User entity class.
  - **user => user.userProfile** specifies the inverse side of the relationship in the User entity. It indicates that the userProfile property in the User entity is the reference to the related entity.

TypeORM One to One


```
// user.service.ts
import { Injectable } from '@nestjs/common';
import { InjectRepository } from '@nestjs/typeorm';
import { Repository } from 'typeorm';
import { User } from './user.entity';
import { UserProfile } from './userProfile.entity';

@Injectable()
export class UserService {
  constructor(
    @InjectRepository(User) private userRepository: Repository<User>,
    @InjectRepository(UserProfile) private userProfileRepository: Repository<UserProfile>,
  ) {}

  async createUser(user: User, userProfile: UserProfile): Promise<User> {
    userProfile.user = user; //assign user object to UserProfile object
    await this.userProfileRepository.save(userProfile);
    return this.userRepository.save(user);
  }
  async getUserWithProfile(id: number): Promise<User> {
    return this.userRepository.findOneBy(id, { relations: ['userProfile'] });
  }
}
```

TypeORM One to Many

One-to-many relationship between a Customer and Order

```
// customer.entity.ts
import { Entity, PrimaryGeneratedColumn, Column, OneToMany } from
'typeorm';
import { Order } from './order.entity';

@Entity()
export class Customer {
  @PrimaryGeneratedColumn()
  id: number;

  @Column()
  name: string;

  @OneToMany(() => Order, order => order.customer, { cascade: true })
  orders: Order[];
}
```

TypeORM One to Many

@OneToMany(() => Order, order => order.customer): to define a one-to-many relationship between the Customer and Order entities.

() => Order: It specifies the target entity for the relationship, which is the Order entity in this case.

order => order.customer: It defines the inverse side of the relationship. It means that the Order entity has a reference to the Customer entity through its customer property.

orders: Order[]: It represents the property in the Customer entity that will hold the collection of related Order entities.

TypeORM One to Many


```
// order.entity.ts
import { Entity, PrimaryGeneratedColumn, Column, ManyToOne } from
'typeorm';
import { Customer } from './customer.entity';

@Entity()
export class Order {
  @PrimaryGeneratedColumn()
  id: number;

  @Column()
  orderNumber: string;

  @ManyToOne(() => Customer, customer => customer.orders)
  customer: Customer;
}
```

TypeORM One to Many


```typescript
import { Injectable } from '@nestjs/common';
import { InjectRepository } from '@nestjs/typeorm';
import { Repository } from 'typeorm';
import { Customer } from './customer.entity';
import { Order } from './order.entity';

@Injectable()
export class CustomerService {
  constructor(
    @InjectRepository(Customer) private customerRepository: Repository<Customer>,
    @InjectRepository(Order) private orderRepository: Repository<Order>,
  ) {}

  async getAllCustomers(): Promise<Customer[]> {
    return this.customerRepository.find({ relations: ['orders'] });
  }

  async createOrder(customerId: number, order: Order): Promise<Order> {
    const customer = await this.customerRepository.findOneBy(customerId);
    order.customer = customer;
    return this.orderRepository.save(order);
  }

  async getOrdersByCustomerId(customerId: number): Promise<Order[]> {
    return this.orderRepository.find({ where: { customer: { id: customerId } } });
  }
}
```

TypeORM Many to Many

Many-to-many relationship between Product and Category

```
// product.entity.ts
import { Entity, PrimaryGeneratedColumn, Column, ManyToMany, JoinTable }
from 'typeorm';
import { Category } from './category.entity';

@Entity()
export class Product {
  @PrimaryGeneratedColumn()
  id: number;

  @Column()
  name: string;

  @ManyToMany(() => Category, category => category.products)
  @JoinTable()
  categories: Category[];
}
```

TypeORM Many to Many


```typescript
// category.entity.ts
import { Entity, PrimaryGeneratedColumn, Column, ManyToMany } from
'typeorm';
import { Product } from './product.entity';

@Entity()
export class Category {
  @PrimaryGeneratedColumn()
  id: number;

  @Column()
  name: string;

  @ManyToMany(() => Product, product => product.categories)
  products: Product[];
}
```

TypeORM Many to Many


```typescript
import { Injectable } from '@nestjs/common';
import { InjectRepository } from '@nestjs/typeorm';
import { Repository } from 'typeorm';
import { Product } from './product.entity';
import { Category } from './category.entity';

@Injectable()
export class ProductService {
  constructor(
    @InjectRepository(Product)
    private productRepository: Repository<Product>,
    @InjectRepository(Category)
    private categoryRepository: Repository<Category>,
  ) {}

  async addProductToCategory(productId: number, categoryId: number): Promise<void> {
    const product = await this.productRepository.findOneBy(productId);
    const category = await this.categoryRepository.findOneBy(categoryId);

    if (product && category) {
      product.categories = [...product.categories, category];
      await this.productRepository.save(product);
    }
  }

  async removeProductFromCategory(productId: number, categoryId: number): Promise<void> {
    const product = await this.productRepository.findOneBy(productId);
    const category = await this.categoryRepository.findOneBy(categoryId);

    if (product && category) {
      product.categories = product.categories.filter((c) => c.id !== category.id);
      await this.productRepository.save(product);
    }
  }

  async getProductsWithCategories(): Promise<Product[]> {
    return this.productRepository.find({ relations: ['categories'] });
  }
}
```

References

1. **W3Schools Online Web Tutorials, URL: http://www.w3schools.com**
2. **Node.js, URL: https://nodejs.org/en/**
3. **Next.js, URL: https://nextjs.org/**
4. **TypeScript URL: https://www.typescriptlang.org/**
5. **MDN Web Docs URL: https://developer.mozilla.org/**
6. **TypeORM URL: https://typeorm.io/**


AMERICAN INTERNATIONAL UNIVERSITY
PRAESIDIUM
DISCIPLINA
CIVITATIS
1994
BANGLADESH

Thank You!