Language Integrated Query

By Christine Bittle

Language Integrated Query allows us to Create, Read, Update, and Delete information from a database connection without having to explicitly write SQL. Instead, we work with entities.

Note: Make sure your database is built before trying to access information. Check using the SQL Server Object Explorer to find the database tables you've created using migrations and the update-database command. Follow this video guide to see step by step for constructing a database using Entity Framework.

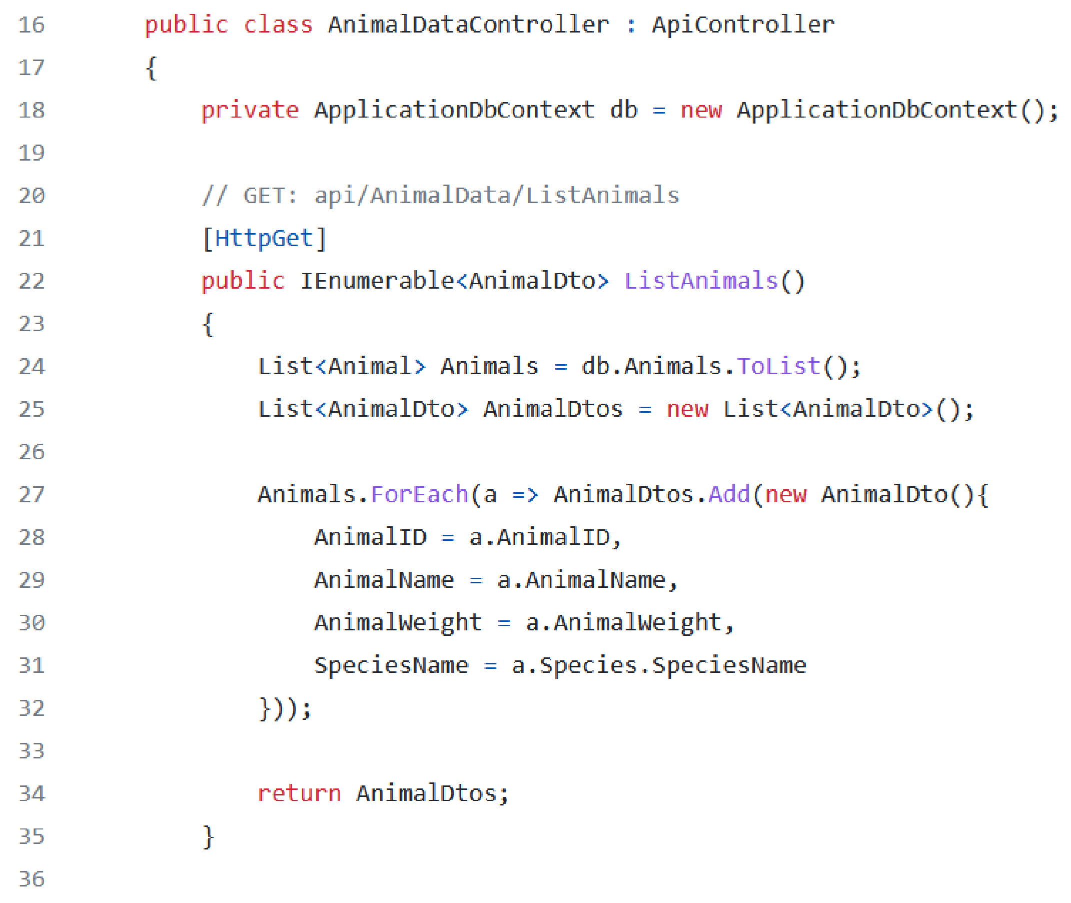
Using a Web API connection, you can now access information from the database as objects. See the following Github example and video guide as a reference.

Using a WebAPI method, our objective is to build CRUD functionality on the Animals table, which includes:

Method Name	Inputs	Outputs
ListAnimals	-	List of Animal Objects
FindAnimal	Animal ID	Animal Object
AddAnimal	Animal Object	-
UpdateAnimal	Animal Id, Animal Object	-
DeleteAnimal	Animal ID	-

READ Functionality

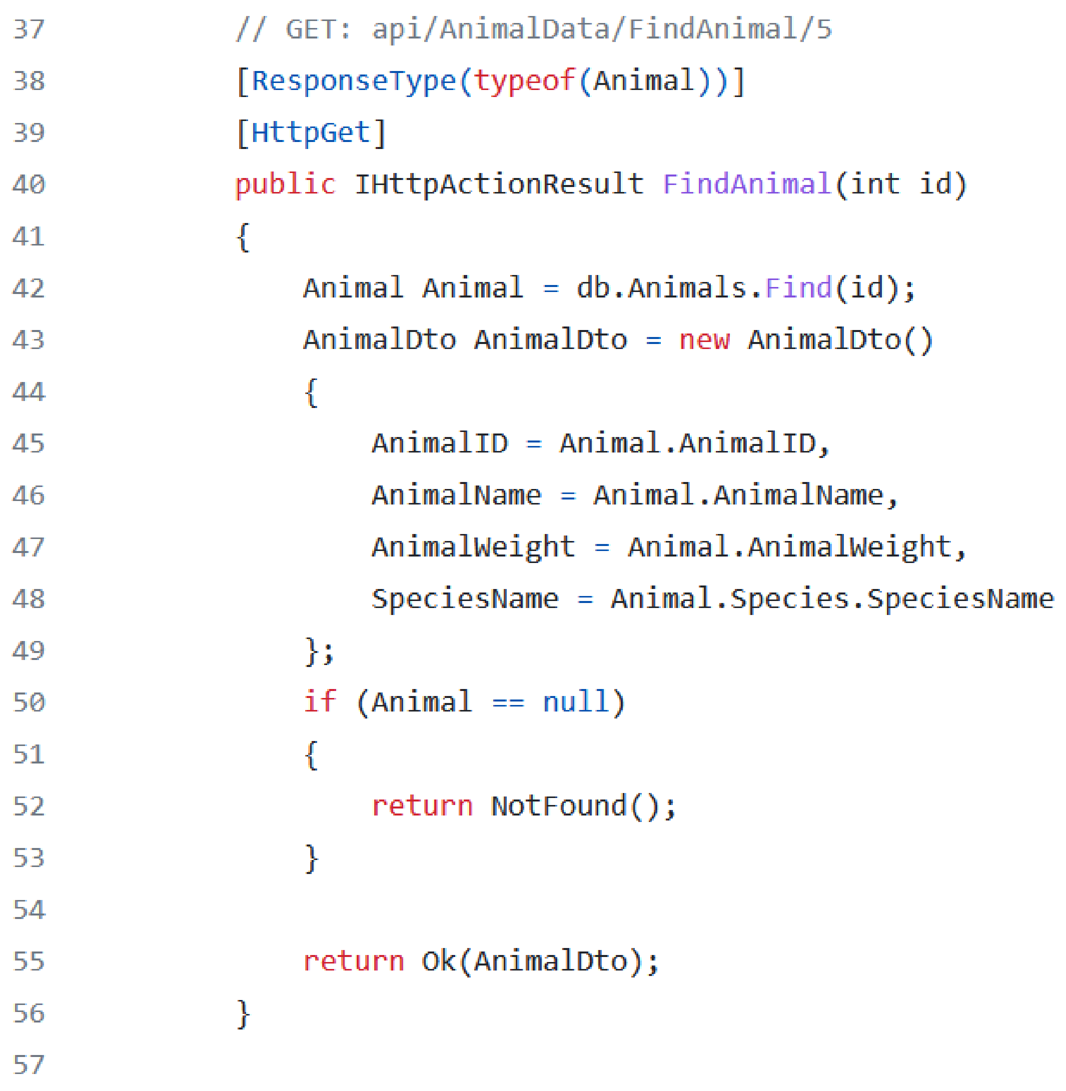
From ZooApplication/Controllers/AnimalDataController.cs



```
    public class AnimalDataController : ApiController
    {
        private ApplicationDbContext db = new ApplicationDbContext();

        // GET: api/AnimalData/ListAnimals
        [HttpGet]
        public IEnumerable<AnimalDto> ListAnimals()
        {
            List<Animal> Animals = db.Animals.ToList();
            List<AnimalDto> AnimalDtos = new List<AnimalDto>();

            Animals.ForEach(a => AnimalDtos.Add(new AnimalDto(){
                AnimalID = a.AnimalID,
                AnimalName = a.AnimalName,
                AnimalWeight = a.AnimalWeight,
                SpeciesName = a.Species.SpeciesName
            }));

            return AnimalDtos;
        }

```

- AnimalDataController.cs is a WebAPI controller. It is configured to listen for requests to localhost:xx/api/AnimalData/{MethodName}
- In this example, the List Animals method can be called with a GET request to api/AnimalData/ListAnimals.
- Use the command prompt to send a curl request to the resource. (i.e. curl localhost:xx/api/AnimalData/ListAnimals
- What did you observe?
- Navigate to the database in the SQL server object explorer.
- Go to the Species table, add two species (1.Monkey, 2.Lion)
- Go to the Animals Table and add three animals. (George, Sam, Leo). George and Sam are monkeys (Species ID 1) Leo is a lion (Species ID 2)
- Try the API method again. You will see a list of animal objects. Use it to build out the summary block!

From [ZooApplication/Controllers/AnimalDataController.cs]()

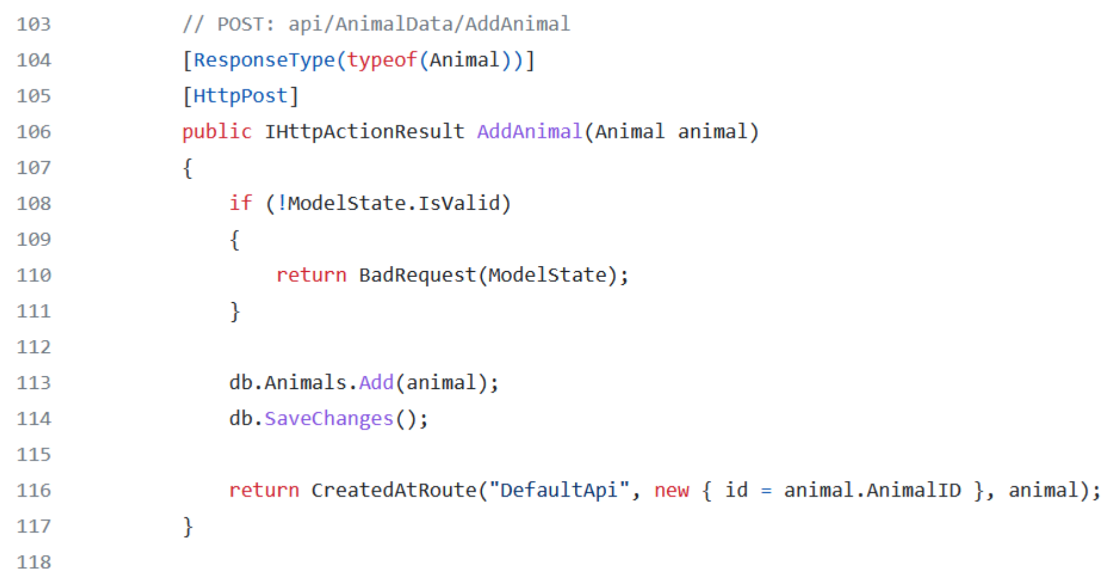


```
37          // GET: api/AnimalData/FindAnimal/5
38          [ResponseType(typeof(Animal))]
39          [HttpGet]
40          public IHttpActionResult FindAnimal(int id)
41          {
42              Animal Animal = db.Animals.Find(id);
43              AnimalDto AnimalDto = new AnimalDto()
44              {
45                  AnimalID = Animal.AnimalID,
46                  AnimalName = Animal.AnimalName,
47                  AnimalWeight = Animal.AnimalWeight,
48                  SpeciesName = Animal.Species.SpeciesName
49              };
50              if (Animal == null)
51              {
52                  return NotFound();
53              }
54
55              return Ok(AnimalDto);
56          }
57
```

- Find Animal is similar to ListAnimals
- Instead of `db.Animals.ToList()`, we use `db.Animals.Find(id)`
- The return type we use is IHTTPActionResult, which allows us to be more flexible.
- On line 52, we can return an HTTP Status code of 404 (Not Found), if the Animal doesn't exist in the database.
- On Line 55, we return an Animal Object
- In Both ListAnimals and ShowAnimal, we return a "Data Transfer Object". This is defined in [ZooApplication/Models/Animal.cs](). We can use this to "package" the data in a way that doesn't affect the database.

ADD Functionality

From [ZooApplication/Controllers/AnimalDataController.cs]()

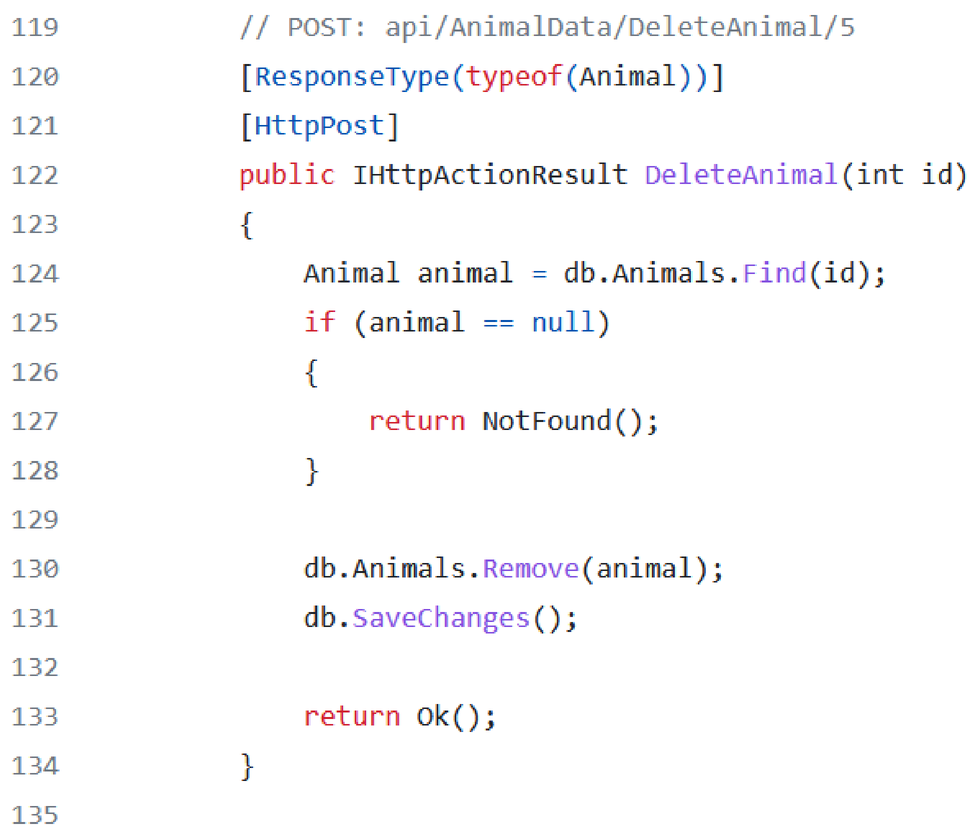


```
        // POST: api/AnimalData/AddAnimal
        [ResponseType(typeof(Animal))]
        [HttpPost]
        public IHttpActionResult AddAnimal(Animal animal)
        {
            if (!ModelState.IsValid)
            {
                return BadRequest(ModelState);
            }

            db.Animals.Add(animal);
            db.SaveChanges();

            return CreatedAtRoute("DefaultApi", new { id = animal.AnimalID }, animal);
        }

```

- Call this method using a POST request from the command prompt.
- Create an animal.json file with the following structure:
- `{“AnimalName”:”George”, “AnimalWeight”:30, “SpeciesID”:1}`
- In your command prompt, use a CURL request to send the POST data
- `cd ../path/to/animal.json`
- `curl -d animal.json -H “Content-Type:application/json” localhost:xx/api/AnimalData/AddAnimal`
- Check the database to see if the new animal is added
- `db.Animals.Add(animal) and db.SaveChanges()` adds the animal to the database
- In the github example, there is a json data folder with an [animal object as a reference.]()

Delete Functionality

From [ZooApplication/Controllers/AnimalDataController.cs]()

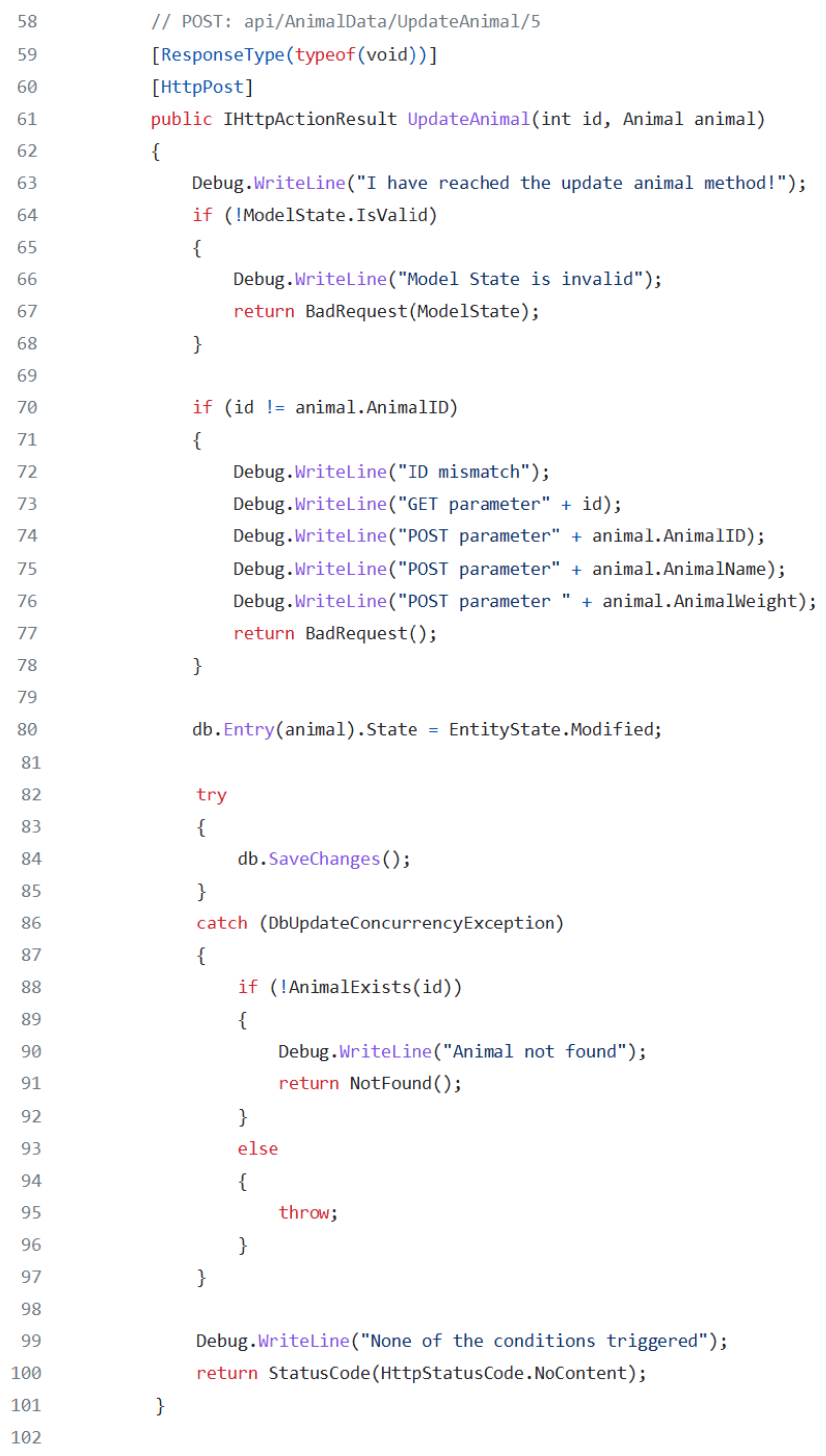


```
        // POST: api/AnimalData/DeleteAnimal/5
        [ResponseType(typeof(Animal))]
        [HttpPost]
        public IHttpActionResult DeleteAnimal(int id)
        {
            Animal animal = db.Animals.Find(id);
            if (animal == null)
            {
                return NotFound();
            }

            db.Animals.Remove(animal);
            db.SaveChanges();

            return Ok();
        }

```

- Use a curl command to delete the animal that was added into the system.
- curl -d “” localhost/api/AnimalData/DeleteAnimal/1
- Check that the animal is deleted
- `db.Animals.Find(id)` is the same as the command used in FindAnimal
- `db.Animals.Remove(animal)` takes an animal object to remove

Update Functionality

From ZooApplication/Controllers/AnimalDataController.cs



```csharp
        // POST: api/AnimalData/UpdateAnimal/5
        [ResponseType(typeof(void))]
        [HttpPost]
        public IHttpActionResult UpdateAnimal(int id, Animal animal)
        {
            Debug.WriteLine("I have reached the update animal method!");
            if (!ModelState.IsValid)
            {
                Debug.WriteLine("Model State is invalid");
                return BadRequest(ModelState);
            }

            if (id != animal.AnimalID)
            {
                Debug.WriteLine("ID mismatch");
                Debug.WriteLine("GET parameter" + id);
                Debug.WriteLine("POST parameter" + animal.AnimalID);
                Debug.WriteLine("POST parameter" + animal.AnimalName);
                Debug.WriteLine("POST parameter " + animal.AnimalWeight);
                return BadRequest();
            }

            db.Entry(animal).State = EntityState.Modified;

            try
            {
                db.SaveChanges();
            }
            catch (DbUpdateConcurrencyException)
            {
                if (!AnimalExists(id))
                {
                    Debug.WriteLine("Animal not found");
                    return NotFound();
                }
                else
                {
                    throw;
                }
            }

            Debug.WriteLine("None of the conditions triggered");
            return StatusCode(HttpStatusCode.NoContent);
        }

```

Update Functionality

- Create another CURL request, similar to the add functionality. This time, make sure to include an Animal ID as part of the JSON data.
- `{"AnimalID": 1,“AnimalName”:”George”, “AnimalWeight”:35, “SpeciesID”:1}`
- In your command prompt, use a CURL request to send the POST data
- `cd ../path/to/animal.json`
- `curl -d animal.json -H “Content-Type:application/json” localhost:xx/api/AnimalData/UpdateAnimal/1`
- Check if the animal with an ID of 1 is updated with the new weight of 35. You can modify any information EXCEPT the Animal ID. For example, you can make George a lion by changing the Species ID.
- `db.Entry(animal).State = EntityState.Modified` and `db.SaveChanges()` change the animal in the database.
- If something goes wrong, you can check the View > Output before sending the CURL request. The `Debug.WriteLine()` messages will appear there, similar to `console.log()` in JavaScript.

Summary

- Language Integrated query can be used to Create, Read, Update, and Delete data in a database
- Use the Command Prompt and CURL to test API methods for Create, Read, Update, and Delete
- Check the database through the Tools > SQL Server Object Explorer to confirm changes to the resource. You can also check if a record is changed by using the Find or List API commands
- If something goes wrong, you can use Debug.WriteLine() to confirm that you have the information you need to execute your function.

Function	SQL	LINQ
`ListAnimals`	`select * from animals`	`db.Animals.ToList()`
`FindAnimal`	`select * from animals where animalid=@id`	`db.Animals.Find(id)`
`AddAnimal`	`insert into Animals (animalname, animalweight, speciesid) values (@animalname, @animalweight, @speciesid)`	`db.Animals.Add(animal)` `db.SaveChanges()`
`DeleteAnimal`	`delete from animals where animalid=@id`	`db.Animals.Remove(animal)` `db.SaveChanges()`
`UpdateAnimal`	`update animals set animalname=@animalname, animalweight=@animalweight, speciesid=@speciesid where animalid=@animalid`	`db.Entry(animal).EntityState = EntityState.Modified` `db.SaveChanges()`

What's Next?

This guide is a good starting point for the CRUD operations in your project. However, more methods are needed to express the full range of data. See the github example ZooApplication_3 for a more full expression of CRUD between related entities.