INTERMEDIATE

CORE DATA

HANDS-ON CHALLENGES

Intermediate Core Data

Luke Parham

Notice of Rights

Notice of Liability

Trademarks

Challenge #4: Improving Performance

By Luke Parham

In the demo, you saw a few tactics for monitoring and fixing performance problems.

Your challenge this time is to look into the performance of the fetch requests on the **Departments** tab of the sample app you saw in the demo.

This time, you'll go back to the world of **XCTest** and learn about writing performance tests!

Analyzing the Department List

First, go to **DepartmentListViewController.swift** and take a look at how the list is built up in `totalEmployeesPerDepartment()`.

```
//1
let fetchRequest: NSFetchRequest<Employee> =
NSFetchRequest(entityName: "Employee")
var fetchResults: [Employee] = []
do {
  fetchResults = try coreDataStack.mainContext.fetch(fetchRequest)
} catch let error as NSError {
  print("ERROR: \(error.localizedDescription)")
  return [[String: String]]()
}

//2
var uniqueDepartments: [String: Int] = [:]
for employee in fetchResults {
  if let employeeDepartmentCount =
      uniqueDepartments[employee.department] {
    uniqueDepartments[employee.department] = employeeDepartmentCount + 1
  } else {
    uniqueDepartments[employee.department] = 1
  }
}

//3
var results: [[String: String]] = []
```

```
for (department, headCount) in uniqueDepartments {
let departmentDictionary: [String: String] =
["department": department,
"headCount": String(headCount)]
results.append(departmentDictionary)
}
return results
```

1. A fetch request is used to retrieve all the employees in the persistent store.
2. A dictionary of department names to employee counts is built up.
3. Then, an array of result dictionaries are built with two keys, "department" and "headCount" pointing to their respective values which can be dropped into the UI.

If you completed challenge 1, you should already know that doing manual counting like this is completely unnecessary. It's also a bit slower since calculations are being done in code instead of on the SQL level. Now you'll see how to measure just how much slower it is.

Testing Performance

Navigate to **DepartmentListViewControllerTests.swift** and add the following test.

```
func testTotalEmployeesPerDepartment() {
    measureMetrics([XCTPerformanceMetric_WallClockTime],
        automaticallyStartMeasuring: false) {

      let departmentList = DepartmentListViewController()
      departmentList.coreDataStack =
          CoreDataStack(modelName: "EmployeeDirectory")
      self.startMeasuring()
      _ = departmentList.totalEmployeesPerDepartment()
    self.stopMeasuring()
  }
}
```

Here, you're using `measureMetrics(_:automaticallyStartMeasuring:for:)` which allows you to to provide a block containing code you want to profile.

You could have the test measure the performance of the entire block by passing true to the `automaticallyStartMeasure` argument, but instead you only want to measure the method you just looked at, `totalEmployeesPerDepartment()`.

Run the test with ⌘**U** and you'll see that the test passes and then provides you with the time it took along with a comparison to the average so far.

The current speed may not seem too bad, but there's definitely room for improvement.

Go back to **DepartmentListViewControllerTests.swift** and add the following alternative method below the original.

```
func totalEmployeesPerDepartmentFast() -> [[String: String]] {
  //1
  let expressionDescription = NSExpressionDescription()
  expressionDescription.name = "headCount"
  //2
  expressionDescription.expression =
      NSExpression(forFunction: "count:",
          arguments:[NSExpression(forKeyPath: "department")])
  //3
  let fetchRequest: NSFetchRequest<NSDictionary> =
  NSFetchRequest(entityName: "Employee")
  fetchRequest.propertiesToFetch =
      ["department", expressionDescription]
  fetchRequest.propertiesToGroupBy = ["department"]
  fetchRequest.resultType = .dictionaryResultType
  //4
  var fetchResults: [NSDictionary] = []
  do {
    fetchResults =
        try coreDataStack.mainContext.fetch(fetchRequest)
  } catch let error as NSError {
    print("ERROR: \(error.localizedDescription)")
    return [[String: String]]()
  }
  return fetchResults as! [[String: String]]
}
```

This code still uses a fetch request, but its a much more efficient one thanks to the custom `NSExpression` you've added.

Here's what's going on.

1. You create an `NSExpressionDescription()` to represent the **"headCount"** in the dictionary you're going to receive.
2. You create the accompanying `NSExpression` applying the `count:` function to the **"department"** keypath.
3. Then you set up the fetch request, this time asking for a dictionary as the result and telling it to return the "department" property for each object, and also the "count" property you just created.
4. Execute the fetch request as usual.

Next, go to `viewDidLoad()` and replace

```
items = totalEmployeesPerDepartment()
```

with

```
items = totalEmployeesPerDepartmentFast()
```

Verifying the Improvement

Now that you've attempted to speed things up, go back to **DepartmentListViewControllerTests.swift**.

Add another test, so you can compare the performance of the two methods.

```
func testTotalEmployeesPerDepartmentFast() {
    measureMetrics([XCTPerformanceMetric_WallClockTime],
        automaticallyStartMeasuring: false) {

     let departmentList = DepartmentListViewController()
     departmentList.coreDataStack =
          CoreDataStack(modelName: "EmployeeDirectory")
     self.startMeasuring()
     _ = departmentList.totalEmployeesPerDepartmentFast()
     self.stopMeasuring()
   }
}
```

Then run with ⌘**U** again to see the results.

As you can see, the second method takes almost half of the time of the original took on my iPhone 5!