CoreData Performance

Where and how data is stored?

Store Type

- NSInMemoryStoreType
- NSSQLiteStoreType
- NSBinaryStoreType
- NSXMLStoreType

More memory - more speed

Scheme design

- Denormalization is OK
- Column indexing

Fetching

NSFetchRequest

- fetchLimit
- fetchBatchSize
- relationshipKeyPathsForPrefetching
- includesSubentities
- returnsObjectsAsFaults
- includesPropertyValues

NSFetchRequest example

```
NSEntityDescription *ed = [NSEntityDescription entityForName:@"Employee"
                                      inManagedObjectContext:context];
NSFetchRequest *fetchRequest = [NSFetchRequest new];
fetchRequest.entity = ed;
fetchRequest.predicate = ...;
fetchRequest.fetchBatchSize = 10;
fetchRequest.relationshipKeyPathsForPrefetching = @[@"department"];
fetchRequest.returnsObjectsAsFaults = NO;
...
```

Fetching distinct values

```
NSFetchRequest *fetchRequest = [NSFetchRequest new];
fetchRequest.entity = ...;
fetchRequest.resultType = NSDictionaryResultType;
fetchRequest.returnsDistinctResults = YES;
fetchRequest.propertiesToFetch = @[@"name"]];
NSArray *objects = [managedObjectContext executeFetchRequest:fetchRequest error:NULL];
for(NSDictionary *dict in objects) {
    NSLog(@"Employee name: %@", dict[@"name]);
}
```

NSPredicate

- Light comparisons go first
- String comparisons are expensive

NSPredicate light comparisons

👎

```
[NSPredicate predicateWithFormat:@"name == %@ && age > %d", @"Tom", 20]
```

👍

```
[NSPredicate predicateWithFormat:@"age > %d && name == %@", 20, @"Tom"]
```

NSPredicate string comparisons

- beginswith and endswith
- ==
- contains
- matches
- [cd] 😢

NSExpression

```
NSExpression *outcomeExpr = [NSExpression expressionForKeyPath:@"outcome"];
NSExpression *incomeExpr = [NSExpression expressionForKeyPath:@"income"];
NSExpression *profitExpr = [NSExpression expressionForFunction:@"from:subtract:"
                                                     arguments:@[incomeExpr, outcomeExpr]];

NSExpressionDescription *expressionDescription = [NSExpressionDescription new];
[expressionDescription setName:@"profit"];
[expressionDescription setExpression:profitExpr];
[expressionDescription setExpressionResultType:NSDoubleAttributeType];

NSFetchRequest *request = [NSFetchRequest fetchRequestWithEntityName:@"Department"];
[request setPropertiesToFetch:@[expressionDescription]];
[request setResultType:NSDictionaryResultType];
NSArray *profits = [context executeFetchRequest:request error:&error];
/*
[
    { "profit": 100 },
    { "profit": 200 },
    { "profit": -10 },
    ...
]
*/
```

NSExpression

average:, sum:, count:, min:, max:, median:, mode:, stddev:, add:to:, from:subtract:, multiply:by:, divide:by:, modulus:by:, sqrt:, log:, ln:, raise:toPower:, exp:, ceiling:, abs:, trunc:, random, random:, now, floor:, uppercase:, lowercase:, bitwiseAnd:with:, bitwiseOr:with:, bitwiseXor:with:, leftshift:by:, rightshift:by:, onesComplement:, noindex:

Fetch in background

```
[backgroundContext performBlock:^{
    NSFetchRequest *fetchRequest = [NSFetchRequest new];
    ...
    fetchRequest.predicate = ...;
    fetchRequest.resultType = NSManagedObjectIDResultType;
    NSArray *objectIDs = [backgroundContext executeFetchRequest:fetchRequest error:NULL];
    [mainContext performBlock:^{
        for (NSManagedObjectID *objectID in objectIDs) {
            NSManagedObject *object = [mainContext objectWithID:objectID];
            // Update UI
        }
    }];
}];
```

NSAsynchronousFetchRequest

```
NSFetchRequest *fetchRequest = [NSFetchRequest new];
...
NSPersistentStoreAsynchronousFetchResultCompletionBlock resultBlock =
                                            ^(NSAsynchronousFetchResult *result) {
                                            // Access result.finalResult
                                            };
NSAsynchronousFetchRequest *asyncFetch = [[NSAsynchronousFetchRequest alloc]
                                          initWithFetchRequest:fetchRequest
                                               completionBlock:resultBlock]
[context performBlock:^{
    NSAsynchronousFetchResult *result = [context executeRequest:asyncFetch
                                                           error:NULL];
    // Access result.progress
}];
```

NSAsynchronousFetchRequest

- Progress and cancellation (KVO)
- Fetch in background vs. asynchronous fetch

Importing data

Common way (fetch-or-insert)

While iterating through data

1. Fetch object with predicate
2. If not exists, insert one
3. Update

Efficient way

1. Sort import object (if possible)
2. Execute a single fetch request
3. Iterate though fetched and import objects
4. Perform operations (update, insert, delete)

Hands on!

Assignment I

- Checkout branch: performance-assignment-1
- Go to ModelController and see parseResponseData:completion method
- Improve performance of existing parsing implementation

Assignment I Solution

Branch: performance-assignment-1-solution

Assignment II (additional)

- Checkout branch: performance-assignment-2
- Go to EmployeesItemsProvider and see loadItems: method
- Play with fetch request properties to improve performance of fetching

Thanks!