

GemStone/S Indexed Collection Primer

Dale Henrichs
GemTalk Systems
ESUG 2014

Why Indexes?

```
"Find people born between 20 and 30 years ago"  
| population today twentyYearsAgo thirtyYearsAgo |  
today := Date today.  
twentyYearsAgo := today - (20 * 365) days.  
thirtyYearsAgo := today - (30 * 365) days.  
^ population  
select: [ :each |  
    thirtyYearsAgo <= each birthday &  
    (each birthday <= twentyYearsAgo) ]
```

Efficient Query Execution

1.6M element population (8300 element result set):

3800ms - select block query

10ms - indexed query

GemStone/S

Indexes

- Indexes are built against UnorderedCollections
- Indexes are based on the instance variables of the objects in the collection.
- Indexes are automatically updated when instance variables are changed
- Indexes use B-trees for efficient equality-based lookup

Query derived from select expression

```
"Find people born between 20 and 30 years ago"  
| population today twentyYearsAgo thirtyYearsAgo |  
today := Date today.  
twentyYearsAgo := today - (20 * 365) days.  
thirtyYearsAgo := today - (30 * 365) days.  
^ population  
select: [ :each |  
    thirtyYearsAgo <= each birthday &  
    (each birthday <= twentyYearsAgo) ]
```

```
"Find people born between 20 and 30 years ago"  
| population today twentyYearsAgo thirtyYearsAgo query |  
today := Date today.  
twentyYearsAgo := today - (20 * 365) days.  
thirtyYearsAgo := today - (30 * 365) days.  
^ ('thirtyYearsAgo <= each.birthday <= twentyYearsAgo'  
asQueryOn: population)  
bind: 'twentyYearsAgo' to: twentyYearsAgo;  
bind: 'thirtyYearsAgo' to: thirtyYearsAgo;  
queryResult
```

```
"Create index on population"  
GsIndexSpec new  
equalityIndex: 'each.birthday' lastElementClass: Date;  
createIndexesOn: population
```

Query Predicates

```
"Find people born between 20 and 30 years ago"  
| population today twentyYearsAgo thirtyYearsAgo |  
today := Date today.  
twentyYearsAgo := today - (20 * 365) days.  
thirtyYearsAgo := today - (30 * 365) days.  
^ population  
select: [ :each |  
    thirtyYearsAgo <= each birthday &  
    (each birthday <= twentyYearsAgo) ]
```

```
"replace message sends with path dot terms"  
(thirtyYearsAgo <= each.birthday) & (each.birthday <= twentyYearsAgo)
```

```
"Convert to a range query predicate - more efficient"  
(thirtyYearsAgo <= each.birthday <= twentyYearsAgo)
```

Query variable binding and execution

```
"Find people born between 20 and 30 years ago"  
| population today twentyYearsAgo thirtyYearsAgo |  
today := Date today.  
twentyYearsAgo := today - (20 * 365) days.  
thirtyYearsAgo := today - (30 * 365) days.  
^ population  
select: [ :each |  
    thirtyYearsAgo <= each birthday &  
    (each birthday <= twentyYearsAgo) ]
```

```
"Bind query variables to values"  
^ ('thirtyYearsAgo <= each.birthday <= twentyYearsAgo'  
asQueryOn: population)  
bind: 'twentyYearsAgo' to: twentyYearsAgo;  
bind: 'thirtyYearsAgo' to: thirtyYearsAgo;  
queryResult
```

Index derived from Query

```
"Find people born between 20 and 30 years ago"  
| population today twentyYearsAgo thirtyYearsAgo query |  
today := Date today.  
twentyYearsAgo := today - (20 * 365) days.  
thirtyYearsAgo := today - (30 * 365) days.  
^ ('thirtyYearsAgo <= each.birthday <= twentyYearsAgo'  
asQueryOn: population)  
bind: 'twentyYearsAgo' to: twentyYearsAgo;  
bind: 'thirtyYearsAgo' to: thirtyYearsAgo;  
queryResult
```

```
"Create index on population"  
GsIndexSpec new  
equalityIndex: 'each.birthday' lastElementClass: Date;  
createIndexesOn: population
```

Last ElementClass

```
"Create index on population"  
GsIndexSpec new  
  equalityIndex: 'each.birthday' lastElementClass: Date;  
  createIndexesOn: population
```

- Expected class of last element in path term
- Any class that responds to comparison messages (< > <= >= = ~=) can be used
- “Basic” Classes cache representative values in B-tree and are restricted to #isKindOf:..

```
"Basic Classes"  
Character  
CharacterCollection  
Date  
DateAndTime  
DateTime  
Number  
Time
```


Additional Path Terms and Operators

```
"Find people who have at least one child  
with 2 children."  
(each.children.*.numberOfChildren = 2)
```

```
"Find people whose firstName or lastName  
is 'Martin'."  
(each.firstName|lastName = 'Martin')
```

```
"Find females."  
(each.gender == #'female')
```

Index Options

```
"Identity index"  
GsIndexSpec new  
  identityIndex: 'each.gender';  
  createIndexesOn: population
```

```
"Unicode equality index"  
GsIndexSpec new  
  unicodeIndex: 'each.gender'  
  collator: (IcuCollator forLocaleNamed: 'en_GB');  
  createIndexesOn: population
```

```
"Reduced conflict equality index"  
GsIndexSpec new  
  equalityIndex: 'each.birthday'  
  lastElementClass: Date;  
  options: GsIndexOptions reducedConflict;  
  createIndexesOn: population
```

```
"Heterogeneous equality index"  
GsIndexSpec new  
  equalityIndex: 'each.birthday'  
  lastElementClass: Date;  
  options: GsIndexOptions optionalPathTerms;  
  createIndexesOn: population
```

Query Evaluation Options

```
| population query |  
query := 'each.firstName = ''Eve''' asQueryOn: population.
```

```
query queryResult.
```

```
query  
  do: [ :each |  
    each lastName = 'Addams'  
    ifTrue: [ ^ each ] ].
```

```
query  
  detect: [ :each | each lastName = 'Addams' ]  
  ifNone: [ nil ].
```

```
query select: [ :each | each numberOfChildren = 3 ].
```

```
query reject: [ :each | each numberOfChildren = 3 ]
```

Query Object Model

```
"Find people born between 20 and 30 years ago"  
| population today twentyYearsAgo thirtyYearsAgo |  
today := Date today.  
twentyYearsAgo := today - (20 * 365) days.  
thirtyYearsAgo := today - (30 * 365) days.  
^ ('thirtyYearsAgo <= each.birthday) &  
  (each.birthday <= twentyYearsAgo)'  
asQueryOn: population)  
bind: 'twentyYearsAgo' to: twentyYearsAgo;  
bind: 'thirtyYearsAgo' to: thirtyYearsAgo;  
queryResult
```

```
"Find people born between 20 and 30 years ago"  
| today twentyYearsAgo thirtyYearsAgo predicate1 predicate2 population |  
today := Date today.  
twentyYearsAgo := today - (20 * 365) days.  
thirtyYearsAgo := today - (30 * 365) days.  
predicate1 := GsQueryPredicate  
  variable: 'thirtyYearsAgo'  
  operator: #'<='  
  path: 'each.birthday'.  
predicate2 := GsQueryPredicate  
  path: 'each.birthday'  
  operator: #'<='  
  variable: 'twentyYearsAgo'.  
^ (GsQuery fromFormula: predicate1 & predicate2 on: population)  
bind: 'twentyYearsAgo' to: twentyYearsAgo;  
bind: 'thirtyYearsAgo' to: thirtyYearsAgo;  
queryResult
```

GsDevKit Tutorial

```
x - NAME lesson_05 - Family Tree Indexes SYNOPSIS lesson_05 [
LESSON 5
-----
In this lesson we look at creating indexes for specific queries. As in
lesson 04, you
can list and execute queries with the following commands:

`./lesson_05 --queries`
`./lesson_05 --run=15`

Equality Indexes
-----
Given the following equality query:

(each.firstName = 'Eve')

To create an index for this query, you extract the path term from the
query, in this case
`each.firstName`, and use it in the equalityIndex creation message:

GsIndexSpec new
  equalityIndex: 'each.firstName' lastElementClass: String;
  createIndexesOn: collection.

The lastElementClass argument should be chosen to closely match the
class of the instances
that will be found in the last instance variable in the pathTerm
(firstName). In general
the only requirement is that the instances found in the instance
variable be comparable
(i.e., messages using these messages: =, ~=, <, >, <=, and >= run
without error. However,
```

```
query 3/ (4)
"String equality query."

| query |
query := '(each.firstName = 'Eve')'
asQueryOn: (Smalltalk at: #'INDEXING_TUTORIAL') population.
query queryOptions: query queryOptions - GsQueryOptions autoOptimize.
GsIndexSpec new
  equalityIndex: 'each.firstName' lastElementClass: String;
  createIndexesOn: (Smalltalk at: #'INDEXING_TUTORIAL') population

x - anArray( ( each.isFemale ), (each.numberOfChildren > 3), (each
.      -> anArray( ( each.isFemale ), (each.numberOfChildren > 3), (each
(class)@ -> Array
(oop)@   -> 267313921
(size)@  -> 16
1@       -> ( each.isFemale )
2@       -> (each.numberOfChildren > 3)
3@       -> (each.firstName = 'Eve')
4@       -> (each.firstName = 'Eve') | (each.firstName = 'Brian')
5@       -> ((each.firstName = 'Eve') | (each.firstName = 'Brian')) not
6@       -> (each.father.numberOfChildren > 3)
7@       -> (2 <= each.numberOfChildren <= 3)
8@       -> (each.gender == #'female')
9@       -> (each.sons.*.numberOfChildren = 2)
10@      -> (each.tags.* = 'soccer') & (each.tags.* = 'magic')
```

Futures

- We intend to open source the Query API and implementation
- Hasso Plattner Institut Senior project to port the Query API to Squeak

Resources

- GemTalk Systems
 - <http://gemtalksystems.com/>
- GsDevKit GitHub project
 - <https://github.com/GsDevKit/gsDevKitHome#open-source-development-kit-for-gemstones-64-bit->
- GsDevKit Indexing tutorial (mail me for availability of tutorial)
 - <https://github.com/GsDevKit/gsDevKitHome/projects/indexSample/README.md#index-sample>