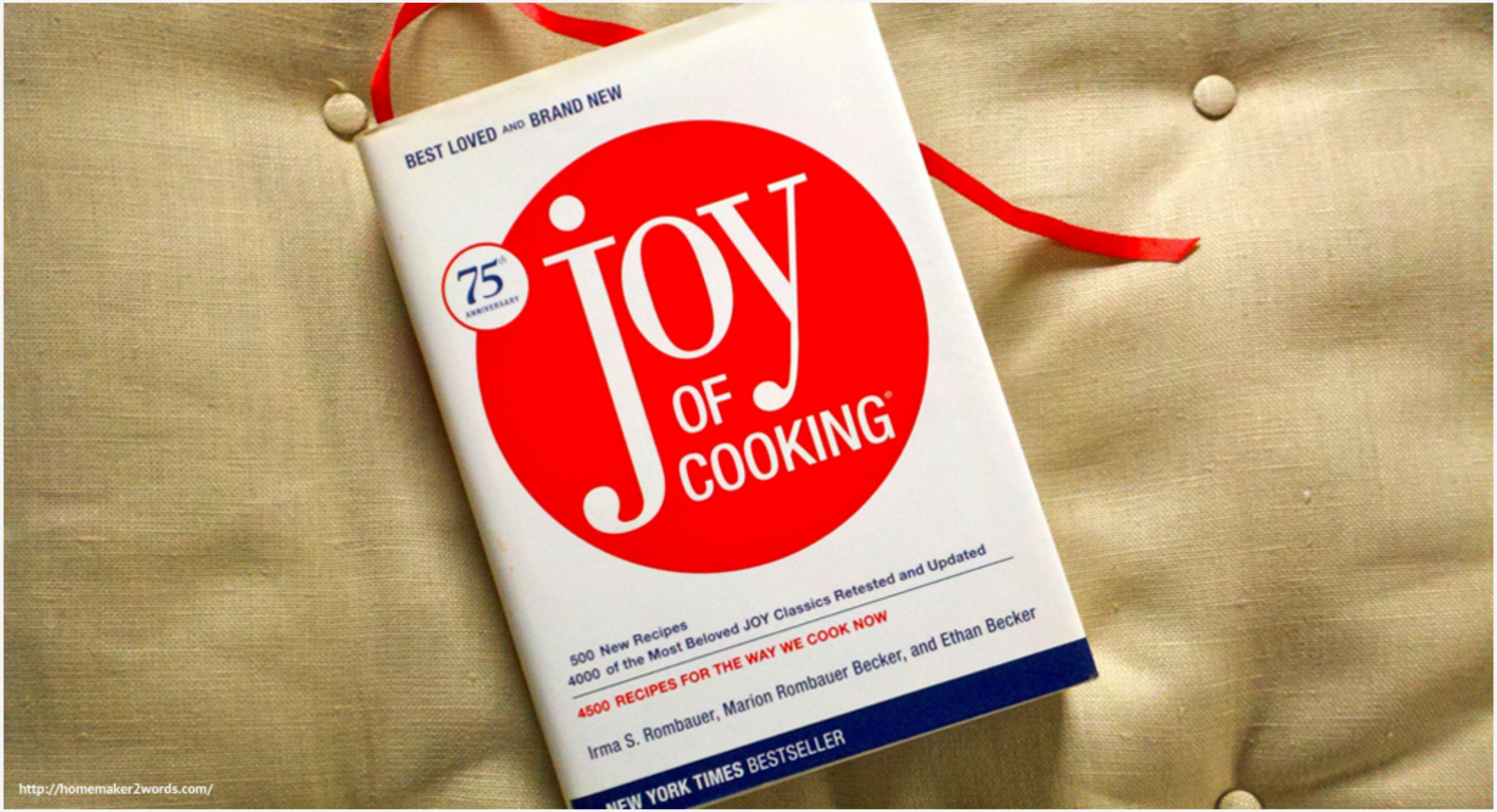
Indexing and Query Optimization

Agenda

- What are indexes?
- Why do I need them?
- Working with indexes in MongoDB
- Optimize your queries
- Avoiding common mistakes

What Are Indexes?

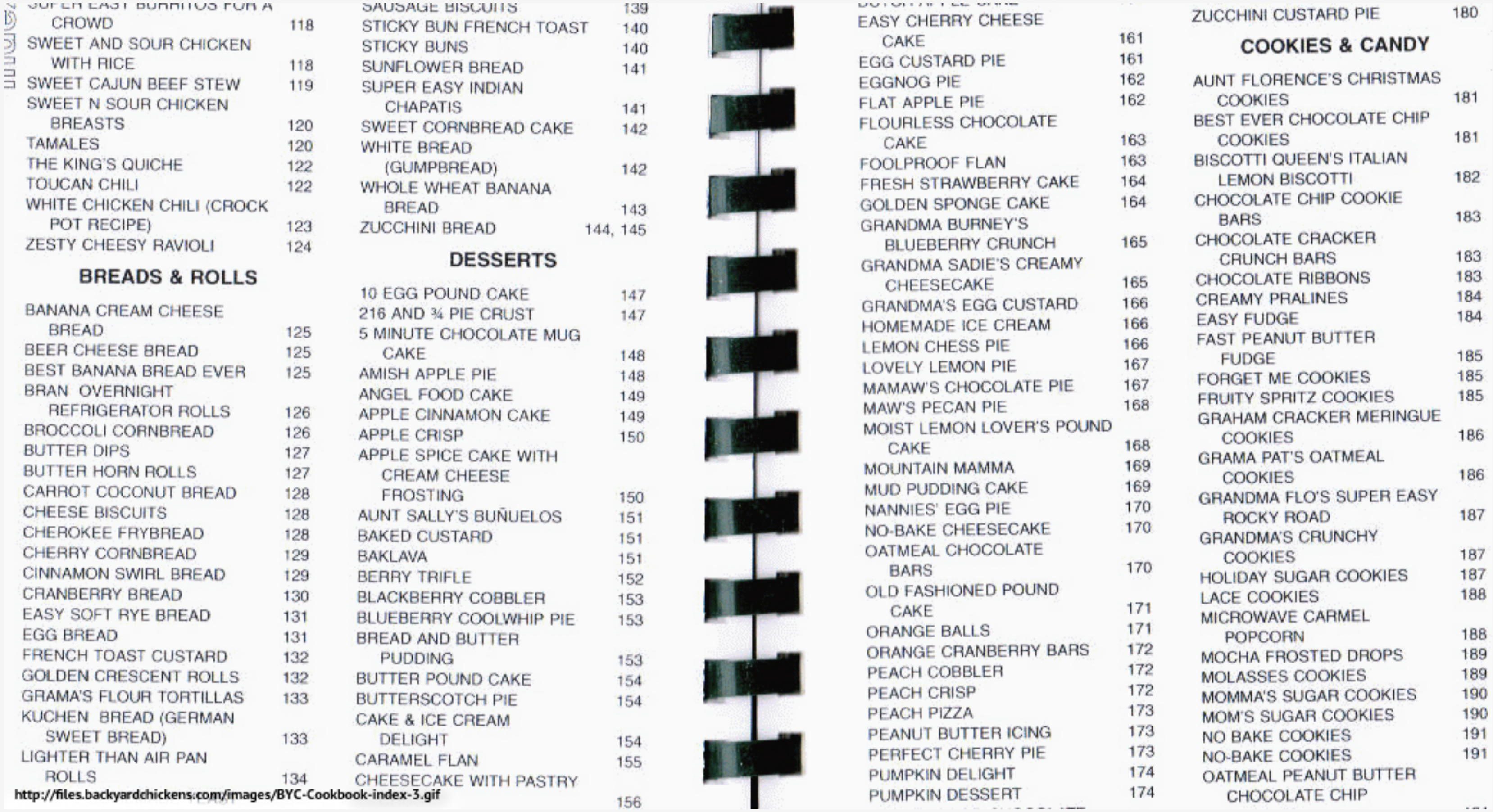

BEST LOVED AND BRAND NEW
75th ANNIVERSARY
joy OF COOKING
500 New Recipes
4000 of the Most Beloved JOY Classics Retested and Updated
4500 RECIPES FOR THE WAY WE COOK NOW
Irma S. Rombauer, Marion Rombauer Becker, and Ethan Becker
NEW YORK TIMES BESTSELLER

http://homemaker2words.com/

What Are Indexes?

Imagine you're looking for a recipe in a cookbook ordered by recipe name. Looking up a recipe by name is quick and easy.

Consult the Index



SUPER EASY BURRITOS FOR A CROWD 118
SWEET AND SOUR CHICKEN WITH RICE 118
SWEET CAJUN BEEF STEW 119
SWEET N SOUR CHICKEN BREASTS 120
TAMALES 120
THE KING'S QUICHE 122
TOUCAN CHILI 122
WHITE CHICKEN CHILI (CROCK POT RECIPE) 123
ZESTY CHEESY RAVIOLI 124

BREADS & ROLLS

BANANA CREAM CHEESE BREAD 125
BEER CHEESE BREAD 125
BEST BANANA BREAD EVER 125
BRAN OVERNIGHT REFRIGERATOR ROLLS 126
BROCCOLI CORNBREAD 126
BUTTER DIPS 127
BUTTER HORN ROLLS 127
CARROT COCONUT BREAD 128
CHEESE BISCUITS 128
CHEROKEE FRYBREAD 128
CHERRY CORNBREAD 129
CINNAMON SWIRL BREAD 129
CRANBERRY BREAD 130
EASY SOFT RYE BREAD 131
EGG BREAD 131
FRENCH TOAST CUSTARD 132
GOLDEN CRESCENT ROLLS 132
GRAMA'S FLOUR TORTILLAS 133
KUCHEN BREAD (GERMAN SWEET BREAD) 133
LIGHTER THAN AIR PAN ROLLS 134

SAUSAGE BISCUITS 139
STICKY BUN FRENCH TOAST 140
STICKY BUNS 140
SUNFLOWER BREAD 141
SUPER EASY INDIAN CHAPATIS 141
SWEET CORNBREAD CAKE 142
WHITE BREAD (GUMPBREAD) 142
WHOLE WHEAT BANANA BREAD 143
ZUCCHINI BREAD 144, 145

DESSERTS

10 EGG POUND CAKE 147
216 AND ¾ PIE CRUST 147
5 MINUTE CHOCOLATE MUG CAKE 148
AMISH APPLE PIE 148
ANGEL FOOD CAKE 149
APPLE CINNAMON CAKE 149
APPLE CRISP 150
APPLE SPICE CAKE WITH CREAM CHEESE FROSTING 150
AUNT SALLY'S BUÑUELOS 151
BAKED CUSTARD 151
BAKLAVA 151
BERRY TRIFLE 152
BLACKBERRY COBBLER 153
BLUEBERRY COOLWHIP PIE 153
BREAD AND BUTTER PUDDING 153
BUTTER POUND CAKE 154
BUTTERSCOTCH PIE 154
CAKE & ICE CREAM DELIGHT 154
CARAMEL FLAN 155
CHEESECAKE WITH PASTRY 156

DUTCH APPLE CAKE
EASY CHERRY CHEESE CAKE 161
EGG CUSTARD PIE 161
EGGNOG PIE 162
FLAT APPLE PIE 162
FLOURLESS CHOCOLATE CAKE 163
FOOLPROOF FLAN 163
FRESH STRAWBERRY CAKE 164
GOLDEN SPONGE CAKE 164
GRANDMA BURNEY'S BLUEBERRY CRUNCH 165
GRANDMA SADIE'S CREAMY CHEESECAKE 165
GRANDMA'S EGG CUSTARD 166
HOMEMADE ICE CREAM 166
LEMON CHESS PIE 166
LOVELY LEMON PIE 167
MAMAW'S CHOCOLATE PIE 167
MAW'S PECAN PIE 168
MOIST LEMON LOVER'S POUND CAKE 168
MOUNTAIN MAMMA 169
MUD PUDDING CAKE 169
NANNIES' EGG PIE 170
NO-BAKE CHEESECAKE 170
OATMEAL CHOCOLATE BARS 170
OLD FASHIONED POUND CAKE 171
ORANGE BALLS 171
ORANGE CRANBERRY BARS 172
PEACH COBBLER 172
PEACH CRISP 172
PEACH PIZZA 173
PEANUT BUTTER ICING 173
PERFECT CHERRY PIE 173
PUMPKIN DELIGHT 174
PUMPKIN DESSERT 174

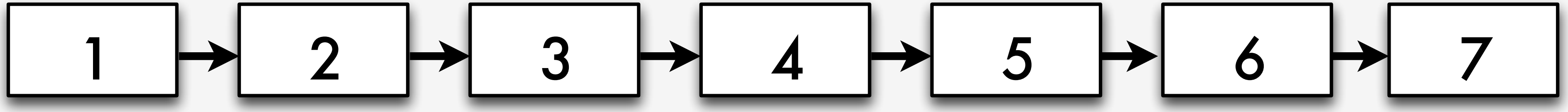
ZUCCHINI CUSTARD PIE 180

COOKIES & CANDY

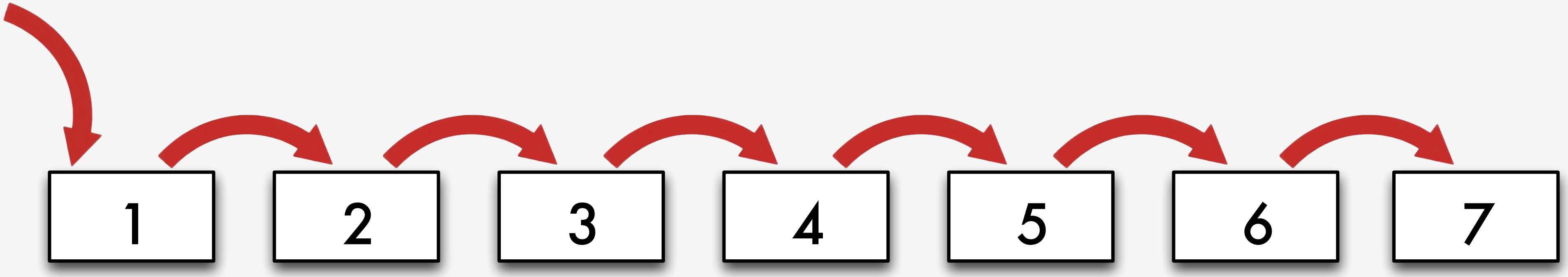
AUNT FLORENCE'S CHRISTMAS COOKIES 181
BEST EVER CHOCOLATE CHIP COOKIES 181
BISCOTTI QUEEN'S ITALIAN LEMON BISCOTTI 182
CHOCOLATE CHIP COOKIE BARS 183
CHOCOLATE CRACKER CRUNCH BARS 183
CHOCOLATE RIBBONS 183
CREAMY PRALINES 184
EASY FUDGE 184
FAST PEANUT BUTTER FUDGE 185
FORGET ME COOKIES 185
FRUITY SPRITZ COOKIES 185
GRAHAM CRACKER MERINGUE COOKIES 186
GRAMA PAT'S OATMEAL COOKIES 186
GRANDMA FLO'S SUPER EASY ROCKY ROAD 187
GRANDMA'S CRUNCHY COOKIES 187
HOLIDAY SUGAR COOKIES 187
LACE COOKIES 188
MICROWAVE CARMEL POPCORN 188
MOCHA FROSTED DROPS 189
MOLASSES COOKIES 189
MOMMA'S SUGAR COOKIES 190
MOM'S SUGAR COOKIES 190
NO BAKE COOKIES 191
NO-BAKE COOKIES 191
OATMEAL PEANUT BUTTER CHOCOLATE CHIP

http://files.backyardchickens.com/images/BYC-Cookbook-index-3.gif

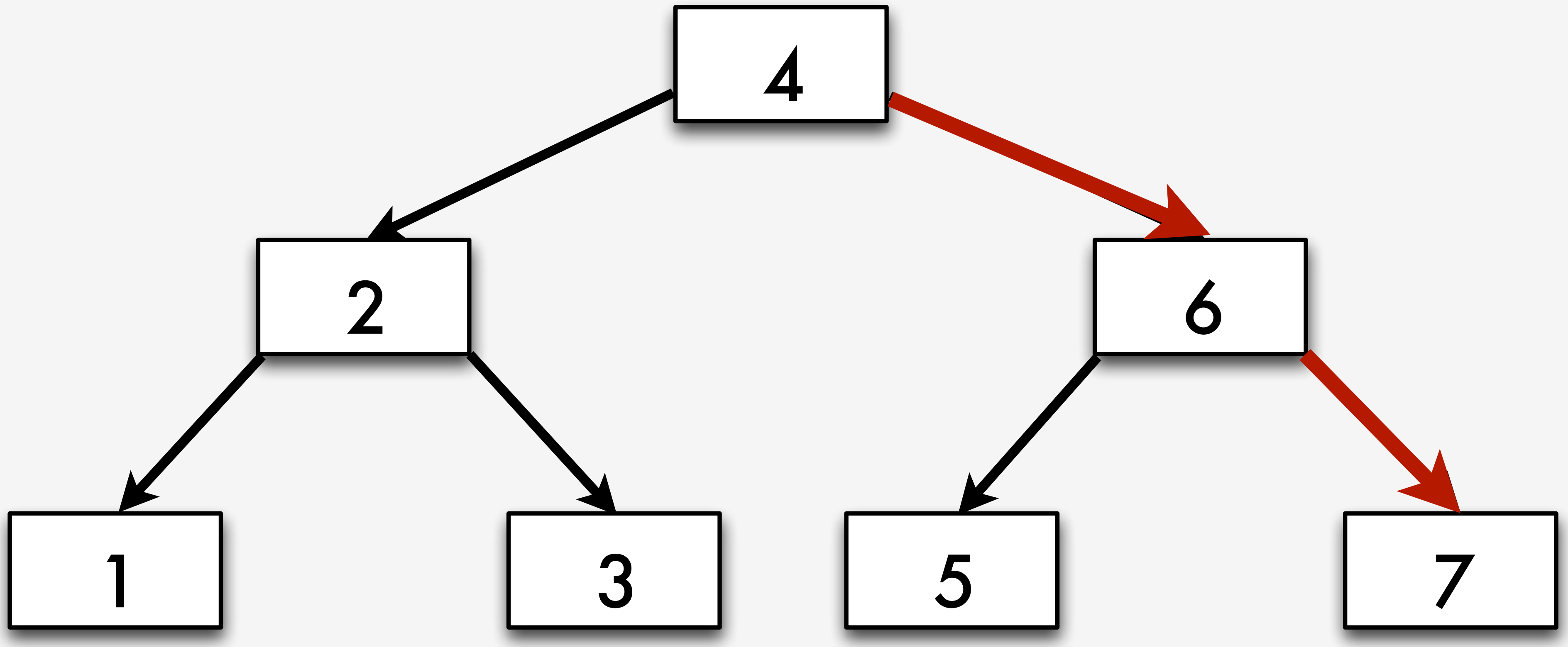
Linked List



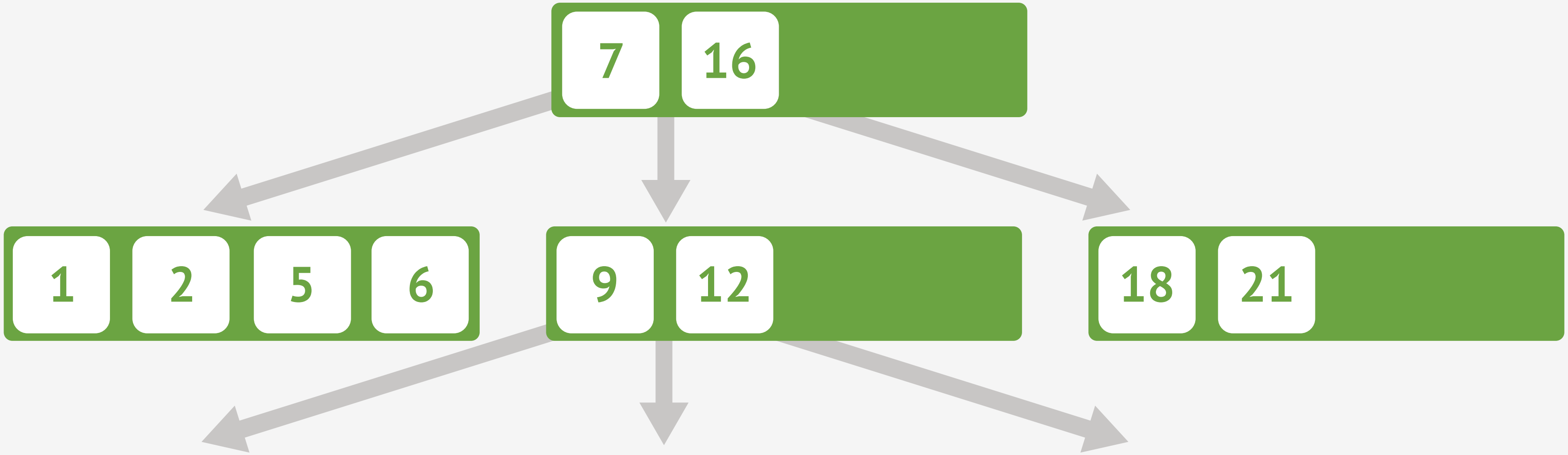
Finding 7 in a Linked List



Finding 7 In a Tree

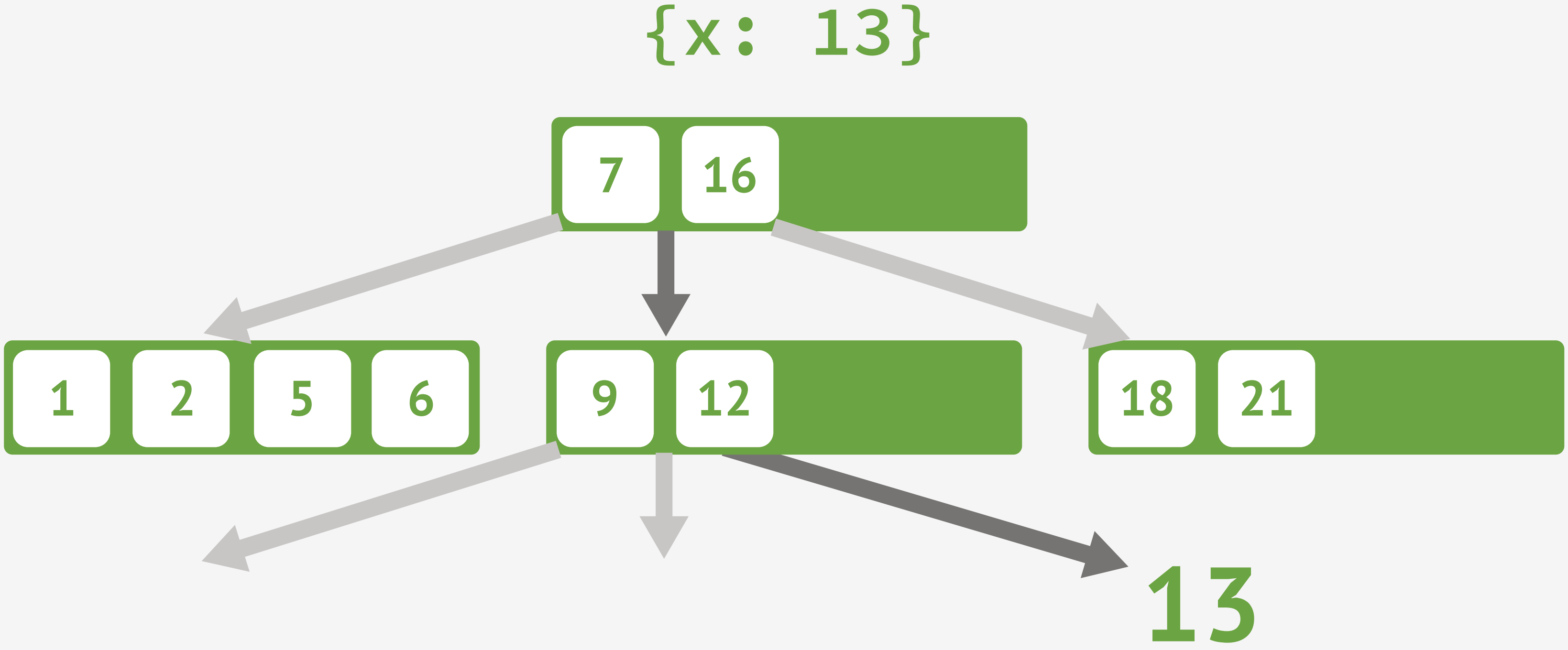


Indexes in MongoDB are B-Trees



Queries, inserts and deletes: O(log(n)) time

B-Trees



Indexes are the single biggest tunable performance factor in MongoDB

Absent or suboptimal indexes are the most common avoidable MongoDB performance problem.

Why do I need indexes?

A brief story

Working with Indexes in MongoDB

How Do I Create Indexes?

```
// Create an index if one does not exist
db.recipes.ensureIndex ( { main_ingredient : 1 } )



// The client remembers the index and raises no errors
db.recipes.ensureIndex( { main_ingredient : 1 } )
```

* 1 means ascending, -1 descending

What Can Be Indexed?

```
// Multiple fields (compound key indexes)
db.recipes.ensureIndex({
    main_ingredient: 1,
    calories: -1
})

// Arrays of values (multikey indexes)
{
    name: 'Chicken Noodle Soup’,
    ingredients : [ 'chicken', 'noodles’ ]
}

db.recipes.ensureIndex( { ingredients : 1 } )
```

What Can Be Indexed?

```
// Embedded documents
{
    name : 'Apple Pie',
    contributor : {
        name : 'Joe American',
        id : 'joea123'
    }
}

db.recipes.ensureIndex ( { 'contributor.id’ : 1 } )

db.recipes.ensureIndex ( { 'contributor’ : 1 } )
```

How Do I Manage Indexes?

```
// List a collection's indexes
db.recipes.getIndexes()
db.recipes.getIndexKeys()

// Drop a specific index
db.recipes.dropIndex ( { ingredients : 1 } )

// Drop all indexes and recreate them
db.recipes.reIndex( )

// Default (unique) index on _id
```

Background Index Builds

```
// Index creation is a blocking operation that can take a long time
// Background creation yields to other operations
db.recipes.ensureIndex (
    { ingredients: 1 },
    { background: true }
)
```

Options

- Uniqueness constraints (unique, dropDups)
- Sparse Indexes

Uniqueness Constraints

```
// Only one recipe can have a given value for name

db.recipes.ensureIndex( { name: 1 }, { unique: true } )


// Force index on collection with duplicate recipe names – drop the duplicates

db.recipes.ensureIndex(
    { name: 1 },
    { unique: true, dropDups: true }
)
```

* dropDups should be used with caution

Sparse Indexes

```
// Only documents with field calories will be indexed
db.recipes.ensureIndex(
    { calories : -1 },
    { sparse : true }
)
// Allow multiple documents to not have calories field
db.recipes.ensureIndex (
    { name: 1 , calories : -1 },
    { unique : true, sparse : true }
)
```

* Missing fields are stored as null(s) in the index

Other Index Types

- Geospatial Indexes (2d Sphere)
- Text Indexes
- TTL Collections (expireAfterSeconds)
- Hashed Indexes for sharding

Geospatial Indexes

```
// Add GeoJSON
{
   name: ’MongoDB Palo Alto’,
   loc: { type : “Point”,
          coordinates: [ 37.449157 , -122.158574 ] }
}
// Index the coordinates
db.locations.ensureIndex( { loc : '2dsphere' } )
```

Geospatial Indexes

```
// Query for locations 'near' a particular coordinate
db.locations.find ( {
   loc: { $near:
          { $geometry:
               { type : “Point”,
               coordinates: [ 37.4, -122.3 ] },
          $maxDistance: 40 }
          }
} )
```

Text Indexes

```
db.recipes.insert( { _id : 4 , y : “add flour and mix" } );
db.recipes.ensureIndex( { y : "text" } );
```

Limitations

- Collections can not have > 64 indexes.
- Index keys can not be > 1024 bytes (1K).
- The name of an index, including the namespace, must be < 125 characters.
- Queries can only use 1 index*
- Indexes have storage requirements, and impact the performance of writes.
- In memory sort (no-index) limited to 32mb of return data.

Optimize Your Queries

The “Explain” Plan (Pre-Index)

```
db.recipes.find( { calories:
   { $lt : 40 } }
).explain( )
{
  "cursor" : "BasicCursor" ,
  "n" : 42,
  "nscannedObjects” : 12345,
  "nscanned" : 12345,
   ...
  "millis" : 356,
   ...
}
```

* Doesn’t use cached plans, re-evals and resets cache

The “Explain” Plan (Post-Index)

```
db.recipes.find( { calories:
   { $lt : 40 } }
).explain( )
{
  "cursor" : ”BtreeCursor calories_1" ,
  "n" : 42,
  "nscannedObjects” : 42,
  "nscanned" : 42,
   ...
  "millis" : 3,
   ...
}
```

* Doesn’t use cached plans, re-evals and resets cache

Profiling Slow Operations

db.setProfilingLevel(*n* , *slowms=100ms*)

n=0 profiler off

n=1 record operations longer than *slowms*

n=2 record all queries

db.system.profile.find()

* The profile collection is a capped collection, and fixed in size

The Query Optimizer

- For each "type" of query, MongoDB periodically tries *all* useful indexes
- Aborts the rest as soon as one plan wins
- The winning plan is temporarily cached for each “type” of query

Manually Select Index to Use

```
// Tell the database what index to use
db.recipes.find( {
    calories: { $lt : 1000 } }
).hint ( { _id : 1 } )

// Tell the database to NOT use an index
db.recipes.find(
    { calories: { $lt : 1000 } }
).hint( { $natural : 1 } )
```

Use Indexes to Sort Query Results

```
// Given the following index
db.collection.ensureIndex( { a : 1, b : 1 , c : 1, d : 1 } )

// The following query and sort operations can use the index
db.collection.find( ).sort( { a : 1 } )
db.collection.find( ).sort( { a : 1, b : 1 } )

db.collection.find( { a : 4 } ).sort( { a : 1, b : 1 } )

//This query will not use the index, but will use the index to sort
db.collection.find( { b : 5 } ).sort( { a : 1, b : 1 } )
```

Indexes that won’t work for sorting query results

```
// Given the following index
db.collection.ensureIndex( { a : 1, b : 1, c : 1, d : 1 } )


// These can not sort using the index
db.collection.find( ).sort( { b : 1 } )
db.collection.find( { b : 5 } ).sort( { b : 1 } )
```

Indexes that won’t work for sorting query results

```
// MongoDB can return data from just the index
db.recipes.ensureIndex({ main_ingredient : 1, name : 1 })

// Return only the ingredients field
db.recipes.find(
   { main_ingredient : 'chicken’ },
   { _id: 0, name : 1 }
)

// indexOnly will be true in the explain plan
db.recipes.find(
   { main_ingredient : 'chicken' },
   { _id: 0, name: 1 }
).explain()
{
   "indexOnly” : true,
}
```

Absent or suboptimal indexes are the most common avoidable MongoDB performance problem.

Avoiding Common Mistakes

Trying to Use Multiple Indexes

```
// MongoDB can only use one index for a query
db.collection.ensureIndex( { a : 1 } )
db.collection.ensureIndex( { b : 1 } )

// Only one of the above indexes is used
db.collection.find( { a : 3, b : 4 } )
```

Compound Key Mistakes

```
// MongoDB can only use one index for a query
db.collection.ensureIndex({ a: 1 })
db.collection.ensureIndex({ b: 1 })

// Only one of the above indexes is used
db.collection.find({ a: 3, b: 4 })

// Use a compound index
db.collection.ensureIndex( { a : 1, b : 1 } )
```

Low Selectivity Indexes

```
db.collection.distinct('status’)
[ 'new', 'processed' ]

db.collection.ensureIndex( { status : 1 } )

// Low selectivity indexes provide little benefit
db.collection.find( { status : 'new' } )

// Better
db.collection.ensureIndex ({ status : 1, created_at : -1 } )
db.collection.find(
    { status : 'new' }
).sort( { created_at : -1 } )
```

Regular Expressions

```
db.users.ensureIndex( { username : 1 } )

// Left anchored regex queries can use the index
db.users.find( { username : /^joe smith/ } )

// But not generic regexes
db.users.find( {username : /smith/ } )

// Or case insensitive queries
db.users.find( { username : /Joe/i } )
```

Negation

```
// Indexes aren't helpful with negations
db.things.ensureIndex( { x : 1 } )

// e.g. "not equal" queries
db.things.find( { x : { $ne : 3 } } )

// ...or "not in" queries
db.things.find( { x : { $nin : [2, 3, 4 ] } } )

// ...or the $not operator
db.people.find( { name: { $not : 'John Doe' } } )
```

Choosing the right indexes is one of the most important things you can do as a MongoDB developer. **Take indexes into consideration when designing your application.**

Thank You