DZone Refcardz

CONTENTS INCLUDE:



# Couchbase API:

## Ultra-Fast, Open Source NoSQL Data Access from Java, Ruby, and .NET

*By: Don Pinto*

Couchbase Server is an open-source NoSQL document database that provides a flexible data model, easy scalability, consistent high-performance and always-on characteristics. This Refcard covers the API's in Java, C#.Net and Ruby used to develop applications using Couchbase Server 2.0.

## JAVA CONNECTION MANAGEMENT

### Connecting to Couchbase Server

```
private static CouchbaseClient client;
new CouchbaseClient(urls, bucket, password)
```

**Parameters:**

| | |
|---|---|
| string urls | Linked list containing one or more URLs as strings |
| String username | Username for Couchbase bucket |
| String password | Password for Couchbase bucket |

### Disconnecting from Couchbase Server

```
client.shutdown();
client.shutdown(long TimeValue, TimeUnit);
```

**Parameters:**

| | |
|---|---|
| long TimeValue | Wait value until any outstanding queued work is completed |
| enum TimeUnit | TimeUnit.SECONDS, TImeUnit.MINUTES |

## STORE OPERATIONS

### Key

Keys can be any UTF-8 string up to 250 characters long. Metadata for keys is 125 bytes + length of key and always kept in RAM.

### Value

If Value is a Hash, it is converted to JSON for Documents and decoded back to a Hash on Retrieve. Value can also be Strings, Binary Strings, Integers (for Atomic Counters, must be positive Integer), and Decimal. If Value is a JSON String you will get back a JSON String (like any other string) that is not parsed by JSON decoder into a Hash.

### Add Operations

The add method adds a value to the database with the specified key, but will fail if the key already exists in the database.

```
client.add(key, expiry, value [, persistTo] ,[ replicateTo] )
client.add(key, expiry, value, transcoder)
```

### Replace Operations

The replace method will replace an existing key with a new value but will fail if the key does not exist in the database.

```
client.replace(key, expiry, value [, persistTo] ,[ replicateTo])
client.replace(key, expiry, value, transcoder)
```

### Set Operations (Durability requirements)

The set method stores a value to the database with a specified key.

```
client.set(key, expiry, value [, persistTo] ,[ replicateTo])
client.set(key, expiry, value, transcoder)
```

**Parameters:**

| | |
|---|---|
| string key | Key used to reference the value. |
| int expiry | Expiry time for the key in seconds. Values larger than 30*24*60*60 seconds (30 days) are interpreted as absolute times (from the epoch) |
| Object value | Value to be stored |
| enum persistTo | The amount of nodes the item should be persisted to before returning. - MASTER/ONE, TWO, THREE or FOUR nodes |
| enum replicateTo | The amount of nodes the item should be replicated to before returning ZERO, ONE, TWO or THREE nodes |
| Transcoder<T> transcoder | Transcoder class to be used to serialize values |

## RETRIEVE OPERATIONS

There are several different flavors of retrieve operations in the Couchbase Server 1.1 Java SDK

```
client.asyncGet(key)
client.asyncGet(key, transcoder)
client.asyncGetAndLock(key, expiry)
client.asyncGetAndLock(key, expiry, transcoder)
client.asyncGetAndTouch(key, expiry)
client.asyncGetAndTouch(key, expiry, transcoder)
client.asyncGetBulk(keycollection)
client.asyncGetBulk(keyn)
client.asyncGetBulk(transcoder, keyn)
client.asyncGetBulk(keycollection, transcoder)
client.asyncGet(key, transcoder)
client.get(key)
                                    ----Snippet cont'd on next page--->
```

```
client.getAndTouch(key, expiry)
client.getAndTouch(key, expiry, transcoder)
client.getBulk(keycollection)
client.getBulk(keyn)
client.getBulk(transcoder, keyn)
client.getBulk(keycollection, transcoder)
client.get(key, transcoder)
client.asyncGetLock(key [,getl-expiry])
client.asyncGetLock(key [,getl-expiry], transcoder)
client.getAndLock(key, getl-expiry)
client.getAndLock(key, getl-expiry, transcoder)
client.asyncGets(key)
client.asyncGets(key, transcoder)
client.gets(key)
client.gets(key, transcoder)
client.unlock(key, casunique)
```

**Parameters:**

| | |
|---|---|
| String key | Key used to reference the value. The key cannot contain control characters or whitespace |
| int expiry | Expiry time for the key in seconds. Values larger than 30*24*60*60 seconds (30 days) are interpreted as absolute times (from the epoch) |
| Collection<String> keycollection | One or more keys used to reference a value |
| String..keyn | One or more keys used to reference a value |
| long casunique | Unique value used to identify a key/value combination |
| int getl-expiry | Expiry time in seconds for lock : Default 15, Maximum 30 |
| Transcoder<T> transcoder | Transcoder class to be used to serialize values |

For more information on creating design documents using Java or for cluster management operations, take a look at http://www.couchbase.com/develop/java/current

## UPDATE OPERATIONS

The update methods support different methods of updating and changing existing information within Couchbase Server.

```
client.append(casunique, key, value)
client.append(casunique, key, value, transcoder)
client.asyncCAS(key, casunique, value)
client.asyncCAS(key, casunique, expiry, value, transcoder)
client.asyncCAS(key, casunique, value, transcoder)
client.cas(key, casunique, value [, persistTo] ,[ replicateTo])
client.cas(key, casunique, expiry, value, transcoder)
client.cas(key, casunique, value, transcoder)
client.asyncDecr(key, offset)
client.decr(key, offset)
client.decr(key, offset, default)
client.decr(key, offset, default, expiry)
client.delete(key [, persistTo] ,[ replicateTo])
client.asyncIncr(key, offset)
client.incr(key, offset)
client.incr(key, offset, default)
client.incr(key, offset, default, expiry)
client.prepend(casunique, key, value)
client.prepend(casunique, key, value, transcoder)
client.touch(key, expiry)
```

**Parameters:**

| | |
|---|---|
| long casunique | Unique value used to identify a key/value combination |
| String key | Key used to reference the value. The key cannot contain control characters or whitespace |
| int offset | Integer offset value to increment / decrement (default is 1) |
| int default | Default value to increment/decrement if key does not exist |
| int expiry | Expiry time for the key in seconds. Values larger than 30*24*60*60 seconds (30 days) are interpreted as absolute times (from the epoch) |
| Object value | Value to be stored |
| enum persistTo | The amount of nodes the item should be persisted to before returning. - MASTER/ONE, TWO, THREE or FOUR nodes |
| enum replicateTo | The amount of nodes the item should be replicated to before returning ZERO, ONE, TWO or THREE nodes |
| Transcoder<T> transcoder | Transcoder class to be used to serialize values |

## QUERY OPERATIONS

With Couchbase Server 2.0, you can add the power of views and querying those views to your applications.

Create a view object to be used when querying a view :

```
client.getView(ddocname, viewname)
```

**Parameters:**

| | |
|---|---|
| String ddocname | Design document name |
| String viewname | View name within a design document |

Then create a new query object to be used when querying the view:

```
Query.new()
Query query = new Query();
```

Once the view and query objects are available, the results of the server view can be accessed using:

```
client.query(view, query)
```

**Parameters:**

| | |
|---|---|
| View view | View object associated with a server view |
| Query query | Query object associated with a server view |

Before accessing the view, a list of options can be set with the query object:

**query.setKey(String key)** to set the key to query in the view
**query.setKey(ComplexKey key)** to set the key to query in the view
**query.setRangeStart(String startKey)** to set the starting key
**query.setRangeStart(ComplexKey startKey)** to set the starting key
**query.setRangeEnd(String endKey)** to set the ending key
**query.setRangeEnd(ComplexKey endKey)** to set the ending key
**query.setRange(String startKey, String endKey)** to set a range
**query.setRange(ComplexKey startKey, ComplexKey endKey)**
**query.setDescending(boolean descending)** to sort in descending order
**query.setIncludeDocs(boolean include)** to include the JSON doc
**query.setReduce(boolean reduce)** execute the reduce function
**query.setStale(Stale reduce)** set to OK will not refresh the view even if it is stale, set to UPDATE_AFTER will update the view after the stale result is returned, FALSE will update the view and return the latest results.

The format of the returned information of the query method is: ViewResponse or any of the other inherited objects such as ViewResponseWithDocs, ViewResponseNoDocs, ViewResponseReduced.

The ViewResponse method provides an iterator() method for iterating through the rows as a ViewRow interface. The ViewResponse method also provides a getMap() method where the result is available as a map.

# RUBY

## CONNECTING TO COUCHBASE

```
client = Couchbase.connect(url, options = {});
client = Couchbase.connect(options = {});
```

**Hash Parameters:**

| | |
|---|---|
| :hostname | IP address for Couchbase Node (String) |
| :bucket | Bucket name (String) [optional, default is .default.]. |
| :password | SASL password for Bucket (String) [optional] |
| :nodelist | Array of IP.s.for Couchbase Nodes ([String]) like [.12.34.56.78:8091.,..12.34.56.79.]. |

```
client = Couchbase.bucket
```

There is a shared single instance of the Couchbase connection object in the Couchbase-model gem and can be used once connected, or when settings are specified in /config/couchbase.yml, which connects on Rails startup.

## WRITING AND UPDATING DATA

### Add Operations

The add method adds a value to the database with the specified key, but will fail (Couchbase::Error::KeyExists) if the key already exists in the database.

```
client.add(key, value, options = {})
client.add[key, options = {}] = value
```

### Replace Operations

The replace method will replace an existing key with a new value but will fail (Couchbase::Error::NotFound) if the key does not exist in the database.

```
client.replace(key, value, options = {})
client.replace[key, options = {}] = value
```

### Set Operations

The set method stores a value to the database with a specified key, overwriting existing content if it exists (see add/replace above).

```
client.set(key, value, options = {})
client.set[key, options = {}] = value
```

**options:**

| | |
|---|---|
| :ttl | [Fixnum, Integer] Time to Live in seconds [optional] |
| :flags | [Fixnum] Flags that you want to store/retrieve [optional] |
| :cas | [Fixnum] Optimistic locking for update control, must match the document cas in Couchbase to succeed |
| :format | [:document, :plain, :marshal] Explicit format setting, :plain for strings, :document for Hash(JSON), :marshal to use Marshal.load and Marshal.dump; format is :document by default |

### Asynchronous Store Example (Block)

```
client.run do
  client.add("foo" => "val1", "bar" => "val2") do |ret|
    ret.operation  #=> :add
    ret.success?   #=> true
    ret.keys       #=> "foo", "bar" in separate calls
    ret.cas
    ret.flags
  end
end
```

## READING DATA

The Get method is versatile, allowing for both optimistic and pessimistic locking, multiple gets, and hash-like syntax.

### Simple Get Operations

```
val = client.get(key, options = {})
val = client["key"]
```

### Extended Tuple Get Operations

```
val, flags, cas = client.get(key, :extended => true)
val, flags, cas = client["key", :extended => true]
```

### Multi-Get Operations

```
val_array = client.get(keys)
val_hash = client.get(keys, :assemble_hash => true)
```

**options:**

| | |
|---|---|
| :extended | [String, Symbol] Key used to reference the value |
| :quiet | [true, false] Return nil for missing keys default, or false to Return Couchbase::Error::KeyNotFound for missing key |
| :ttl | [Fixnum, Integer] Get and Touch, :ttl will also reset the TTL at the same time as the get (seconds) |
| :lock | Lock the document, true to use default lock timeout, or integer for number of seconds to lock |
| :format | [:document, :plain, :marshal] Explicit format setting, :plain for strings, :document for Hash(JSON), :marshal to use Marshal.load and Marshal.dump |
| :assemble_hash | [true, false] For Multi-Get, will return a Hash keyed on the document keys instead of default of array |

## GETTING STATS FROM COUCHBASE

Obtain stats from all servers for the connection (bucket)

```
client.stats
client.stats["stats"]
```

### To fetch memory stats

```
client.stats(:memory)
```

**Parameters:**

| | |
|---|---|
| stat | Individual stat string (i.e. "curr_items") |

## DELETING AND UPDATING DATA

### To delete keys/values

```
client.delete(key)
```

### Atomic Counter Operations

Atomic Counters are only atomic per cluster, but are useful for many different patterns and numeric data. They are unsigned positive integers. You cannot use atomic operations on floats or objects or arrays.

```
client.incr(key, delta, options = {})
client.decr(key, delta, options = {})
```

**options:**

| | |
|---|---|
| delta | [Fixnum] Amount to incr/decr, default is 1, can be up to 64 bits |
| :create | [true, false] If set to true, if key doesn.t exist, initializes to zero but doesn.t increment, can be combined with :initial |
| :initial | [Fixnum, Unsigned Integer] Sets initial value if key doesn.t exist, doesn.t increment the initial value! |
| :extended | [true, false] Returns [value, cas] tuple instead of just value |
| :ttl | [Fixnum, Integer] Sets TTL, doesn.t alter existing TTL, will only be applied to a new item created via :create and/or :initial |

### Touch Operations

Reset the TTL expiration on one or more keys explicitly with the touch command, can also be done with a get command.

```
client.touch(key, ttl)
client.touch({“key1”=> ttl1, “key2” => ttl2})
```

### Durability Requirements

Observe the state of the keys on all the nodes or using :replicated and :persisted it allows to set up the waiting rule.

```
client.observe(*keys, options = {})
client.observe_and_wait(*keys, options = {})
```

### Non-JSON Operations

For non-json values, prepend and append can concatenate to existing values accordingly. Defaults to:plain format for encoder/decoder.

```
client.prepend(key, value)
client.append(key, value)
```

## QUERYING DATA

With Couchbase Server 2.0, you can create Indexes through Map/Reduce functions. http://www.couchbase.com/docs/couchbase-manual-2.0/couchbase-views.html

### Create Design Doc Object

Refer to the design_doc by name, the Views are an array of view names, as well as functions defined by name.

```
ddoc = client.design_docs[‘design_doc_name’]
ddoc.views   #=> [‘view_name’, ‘view_name_2’, …]
```

Use dot notation for accessing the view by your defined view name and pass in options for query parameters:

```
ddoc.view_name(params = {}).each do |doc|
  # do stuff with docs
end
```

**options:**

| | |
|---|---|
| :include_docs | [true, false] Perform get with key on rows returned with key (non-reduce) |
| :descending | [true, false], reverse order of returned rows |
| :key | [String, Fixnum, Hash, Array] Return only rows with index key that match :key, simple and compound keys can be used (JSON encoded) |
| :keys | [Array] Return only rows that match array of keys (see :key), both simple and compound keys (JSON encoded) |
| :startkey | [String, Fixnum, Hash, Array] Range query from :start_key to :end_key (JSON encoded) |
| :endkey | [String, Fixnum, Hash, Array] Range query from :start_key to :end_key (JSON encoded) |
| :startkey_docid | [String] Document id to start with (to allow pagination for duplicate startkeys |
| :endkey_docid | [String] Last document id to include in the output (to allow pagination for duplicate startkeys) |
| :inclusive_end | [true, false] If true, specificed end key is included in result |
| :limit | [Fixnum] Limit the results returned |
| :skip | [Fixnum] Skip a number of results before returning (can be used with :limit to page through results) |
| :reduce | [true, false] Perform the reduce function defined in the view. If there is no reduce defined default is false and setting to true will return error. |
| :group | [true, false] Groups results using the view defined reduce function |
| :group_level | [Fixnum] Sets the grouping level for compound keys |
| :stale | [:false, :update_after, :ok] Consistency setting, :false initiates Design Document to update indexes before returning results, :update_after updates indexes after returning results, :ok doesn.t trigger indexers and returns existing indexed results |
| :body | [Hash] Takes same parameters as options but does it as a POST if the query has many parameters or is complex |
| :on_error | [:continue, :stop] Behavior setting if an error occurs during view query |

## .NET CONNECTING TO COUCHBASE

```
var config = new CouchbaseClientConfiguration();
config.Urls.Add(new Uri(“http://<hostname>:8091/pools/”));
config.Bucket = “<bucketname>”;
var client = new CouchbaseClient(config);
```

## WRITING DATA

Based on the storemode, Store and ExecuteStore may be an Add, Replace or Set operation. Execute API's return the operation result.

### Store Operations

```
object.Store(storemode, key, value)
object.Store(storemode, key, value, validfor)
object.Store(storemode, key, value, expiresat)
```

### ExecuteStore Operations

```
object.ExecuteStore(storemode, key, value)
object.ExecuteStore(storemode, key, value, expiresat)
object.ExecuteStore(storemode, key, value, validfor)
object.ExecuteStore(storemode, key, value, ReplicateTo)
object.ExecuteStore(storemode, key, value, PersistTo)
object.ExecuteStore(storemode, key, value, PersistTo,
ReplicateTo)
```

**Parameters:**

| | |
|---|---|
| String key | Key used to reference the value. |
| Object value | Value to be stored |
| StoreMode storemode | Storage mode for a given key/value pair can be Add, Replace or Set. |
| Timespan validfor | Expiry timespan (in seconds) for key |
| DateTime expiresat | Explicit expiry time for key |
| Enum PersistsTo | Persist to one or more replicas. Master plus one, two, three replicas |
| Enum ReplicateTo | Replicate to zero or more replicas |

## READING DATA

### Get Operations

```
object.Get(key, expiry)
object.ExecuteGet(key, expiry)
object.ExecuteGet(key)
object.ExecuteGet(keyarray)
object.Get(key)
object.Get(keyarray)
object.GetWithCas(key)
```

**Parameters:**

| | |
|---|---|
| String key | Key used to reference the value. The key cannot contain control characters or whitespace |
| object expiry | Expiry time for the key in seconds. Values larger than 30*24*60*60 seconds (30 days) are interpreted as absolute times (from the epoch) |
| List <string> keyarray | Array of keys used to reference one or more values |

## UPDATING DATA

Update existing information on the server:

```
object.Append(key, value)
object.Append(key, casvalue, value)
object.ExecuteAppend(key, value)
object.ExecuteAppend(key, casvalue, value)
object.Cas(storemode, key, value)
object.Cas(storemode, key, value, casunique)
object.Cas(storemode, key, value, validfor, casunique)
object.ExecuteCas(storemode, key, value)
object.ExecuteCas(storemode, key, value, casunique)
object.ExecuteCas(storemode, key, value, expiresat, casunique)
object.ExecuteCas(storemode, key, value, validfor, casunique)
object.Decrement(key, defaultvalue, offset)
object.Decrement(key, defaultvalue, offset, casunique)
object.Decrement(key, defaultvalue, offset, expiresat, casunique)
object.Decrement(key, defaultvalue, offset, validfor, casunique)
object.Decrement(key, defaultvalue, offset, expiresat)
object.Decrement(key, defaultvalue, offset, validfor)
object.Decrement(key, defaultvalue, offset)
object.Decrement(key, defaultvalue, offset, casunique)
object.Decrement(key, defaultvalue, offset, validfor, casunique)
object.Decrement(key, defaultvalue, offset, expiresat)
object.Decrement(key, defaultvalue, offset, validfor)
object.ExecuteDecrement(key, defaultvalue, offset)
object.ExecuteDecrement(key, defaultvalue, offset, casunique)
object.ExecuteDecrement(key, defaultvalue, offset, expiresat,
casunique)
object.ExecuteDecrement(key, defaultvalue, offset, validfor,
casunique)
object.ExecuteDecrement(key, defaultvalue, offset, expiresat)
object.ExecuteDecrement(key, defaultvalue, offset, validfor)
object.ExecuteRemove(key)
object.Remove(key)
object.ExecuteIncrement(key, defaultvalue, offset)
object.ExecuteIncrement(key, defaultvalue, offset, casunique)
object.ExecuteIncrement(key, defaultvalue, offset, expiresat,
casunique)
object.ExecuteIncrement(key, defaultvalue, offset, validfor,
casunique)
object.ExecuteIncrement(key, defaultvalue, offset, expiresat)
object.ExecuteIncrement(key, defaultvalue, offset, validfor)
object.Increment(key, defaultvalue, offset)
object.Increment(key, defaultvalue, offset, casunique)
object.Increment(key, defaultvalue, offset, expiresat, casunique)
object.Increment(key, defaultvalue, offset, validfor, casunique)
object.Increment(key, defaultvalue, offset, expiresat)
object.Increment(key, defaultvalue, offset, validfor)
object.ExecutePrepend(key, value)
object.ExecutePrepend(key, casunique, value)
object.Prepend(key, value)
object.Prepend(key, casunique, value)
object.Touch(key, expiry)
```

**Parameters:**

| | |
|---|---|
| String key | Key used to reference the value. |
| Object value | Value to be stored |
| StoreMode storemode | Storage mode for a given key/value pair |
| Timespan validfor | Expiry timespan (in seconds) for key |
| DateTime expiresat | Explicit expiry time for key |
| Object defaultvalue | Value to be stored if the key does not exists |
| offset | Integer offset value to increment or decrement (default 1) |
| casunique | Unique value used to verify a key/value combination |

## QUERYING DATA

With Couchbase Server 2.0, you can add views and query those views using your applications.

Create a view object to be used when querying a view :

```
GetView(designName, viewName)
```

**Parameters:**

| | |
|---|---|
| String designName | Design document name |
| String viewName | View name within a design document |

There is also a generic version of GetView, which has a third boolean parameter that tells the client to lookup the original document by its ID.

```
GetView<T>(designName, viewName, bLookup)
```

You can iterate over the returned collection as follows:

```
var beersByNameAndABV = client.GetView<Beer>("beers", "by_name_
and_abv");
foreach(var beer in beersByNameAndABV) { … }
```

If you iterate over a strongly typed view, each item is of the type you specified. If you use the non-generic version, each item you enumerate over will be of type IViewRow. IViewRow provides methods for accessing details of the row that are not present when using strongly typed views.

To get the original document from Couchbase:

```
row.GetItem()
```

To get a Dictionary representation of the view:

```
row.Info
```

To get the original document's ID:

```
row.ItemId
```

To get the key emitted by the map function:

```
row.ViewKey
```

To limit the number of documents returned by the query to 10:

```
GetView(designName, viewName).Limit(10)
```

To group the results when using _count for example:

```
GetView(designName, viewName).Group(true)
```

To disallow stale results in the view:

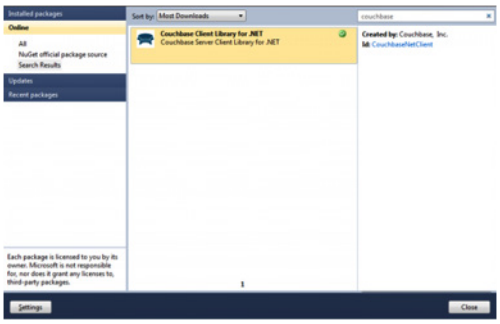
```
GetView(designName, viewName).Stale(StaleMode.False)
```

To limit the number of results to and order the results in descending order:

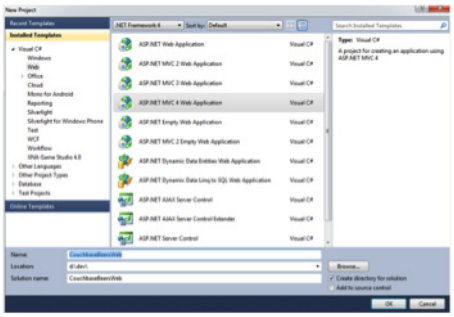
```
GetView(designName, viewName).Limit(5).Descending(true)
```

## GETTING STARTED WITH COUCHBASE C# LIBRARIES

Using the NuGet package manager you can get the Couchbase .Net client using '**Install-Package CouchbaseNetClient**'



To create a new MVC project using Couchbase Server, select **File -> New Project** and then select **Web -> ASP.Net MVC4 application** under the Visual C# project templates. Give your project a name and click "OK" to create the solution.



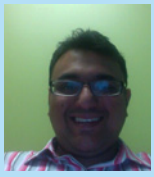
Finally, add a reference to the Couchbase .Net Client Library in your project.

Enjoy building your application using Couchbase Server!

### Useful Links

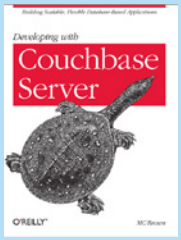
Couchbase Website: http://www.couchbase.com
Couchbase Blog: http://blog.couchbase.com
Developer SDKs: http://www.couchbase.com/develop
Download: http://www.couchbase.com/download
Autodocs: http://www.couchbase.com/autodocs
Java : http://www.couchbase.com/develop/java/current
.NET : http://www.couchbase.com/develop/net/current
Ruby : http://www.couchbase.com/develop/ruby/current

### ABOUT THE AUTHORS



**Don Pinto** is a Product Marketing Manager at Couchbase responsible for product marketing development, positioning, messaging and collateral. For this Refcard, Don worked with Jasdeep Jaitla, John Zablocki, Michael Nitschinger, Sergey Avseyev and Tugdual Grall, all employees of Couchbase, Inc. Check out their website[http://www.couchbase.com/] and blog[http://blog.couchbase.com/] for Couchbase news, tips, and tricks.

### RECOMMENDED BOOK




Today's highly interactive websites pose a challenge for traditional SQL databases—the ability to scale rapidly and serve loads of concurrent users. With this concise guide, you'll learn how to build web applications on top of Couchbase Server 2.0, a NoSQL database that can handle websites and social media where hundreds of thousands of users read and write large volumes of information.

**Buy Here**



DZone

DZone communities deliver over 6 million pages each month to more than 3.3 million software developers, architects and decision makers. DZone offers something for everyone, including news, tutorials, cheat sheets, blogs, feature articles, source code and more. "DZone is a developer's dream", says PC Magazine.



DZone, Inc.
150 Preston Executive Dr.
Suite 201
Cary, NC 27513

888.678.0399
919.678.0300

**Refcardz Feedback Welcome**
refcardz@dzone.com

**Sponsorship Opportunities**
sales@dzone.com

ISBN-13: 978-1-936502-74-5
ISBN-10: 1-936502-74-7
50795
9 781936 502745
$7.95

Version 1.0