Document stores and CouchDB

Hamza Nougba - Ziad Beyens

Université libre de Bruxelles

hnougba@ulb.ac.be - zbeyens@ulb.ac.be

December 18, 2017

Overview

Overview

What is CouchDB?

Definition of CouchDB

"CouchDB is a NoSQL database that completely embraces the web. Store your data with JSON documents. Access your documents with your web browser, via HTTP. Query, combine, and transform your documents with JavaScript."

- Document stores
- Distributed architecture
- Easy replication
- Document stored as JSON
- RESTful HTTP API
- MapReduce functions for views (in Javascript)

NoSQL

- **N**ot **o**nly **SQL**.
- Class of DBMS (Database Management Systems) and an alternative to RDBMS
- **Non-relational** databases and generally schema-less $\Rightarrow$ better performance and scalability.
- Types of NoSQL databases:
  - Document stores (i.e. **CouchDB**, MongoDB)
  - Key-Value stores (i.e. Redis)
  - Wide-column stores (i.e. Cassandra)
  - Graph stores (i.e. FlockDB)

What is JSON?

- **J**ava**S**cript **O**bject **N**otation
- Based on a subset of JavaScript syntax but different (JSON not necessary JavaScript and JavaScript not necessary JSON)
- **Lightweight** and easy syntax for structuring and formatting data
- Text based, human readable data exchange format
- Can be used by different programming languages (NodeJS, Java, Python, Ruby, etc)
- Filename extension **.json**

JSON format

- JSON format consists in 2 possible structures
  - an unordered collection of key/value pairs (an object).
    - The key is a string
    - The value can be a JSON object, a string, a number, an array, a boolean or null
  - an ordered collection of values (an array).

Example

```
{
"course_name" : "INFO-H415 advanced databases",
"Number of students": 40,
"students": [{"name": "Hamza Nougba",
              "project name": "CouchDB},
             {"name": "Ziad Beyens"
              "project name": "CouchDB},...]
}
```

- *A document* is an object.

CouchDB Documents

- A CouchDB document contains the data in a JSON format
- Every documents in CouchDB has:
  - an *_id* value.
  - a *_rev* (revision) value. Updating a document $\Rightarrow$ *_rev* changes.

Example

```
{
"_id": "myapp",
"_rev": "1-64db269b26ed399733beb259d1a31304"
}
```

Overview

The Main Principles of REST

- RESTful services are used to communicate with CouchDB
- **RE**presentational **S**tate **T**ransfer
  - Architectural style (client-server)
  - Distributed systems
  - Application (non-human users) - CouchDB
- Stateless
- Protocol: HTTP
- Representations: JSON, XML, HTML

Client-Server Architecture

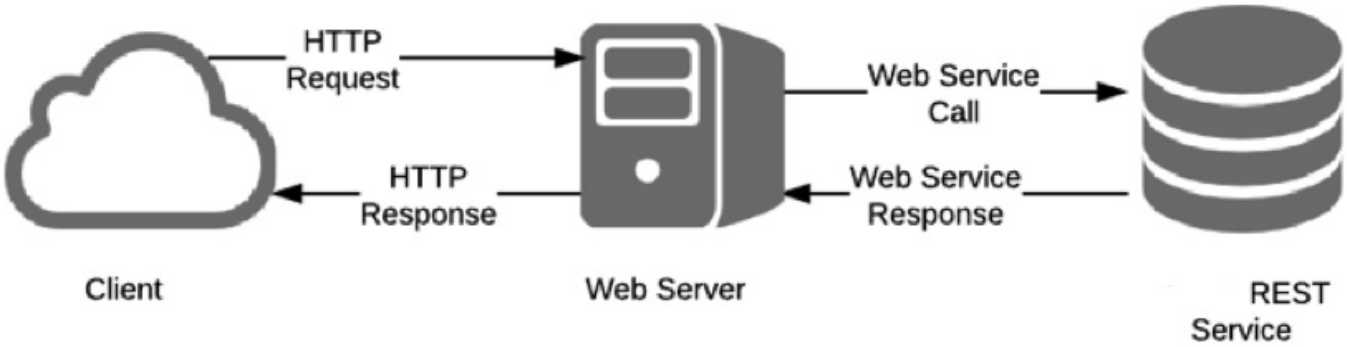


Figure: Client - Server - CouchDB representation at `https://smartdogservices.com/wp-content/uploads/2014/09/11.png`

Stateless

- The servers manages the resource states (no client state)
- The client manages the session states
- Each request needs all the data to process the request

REST Triangle

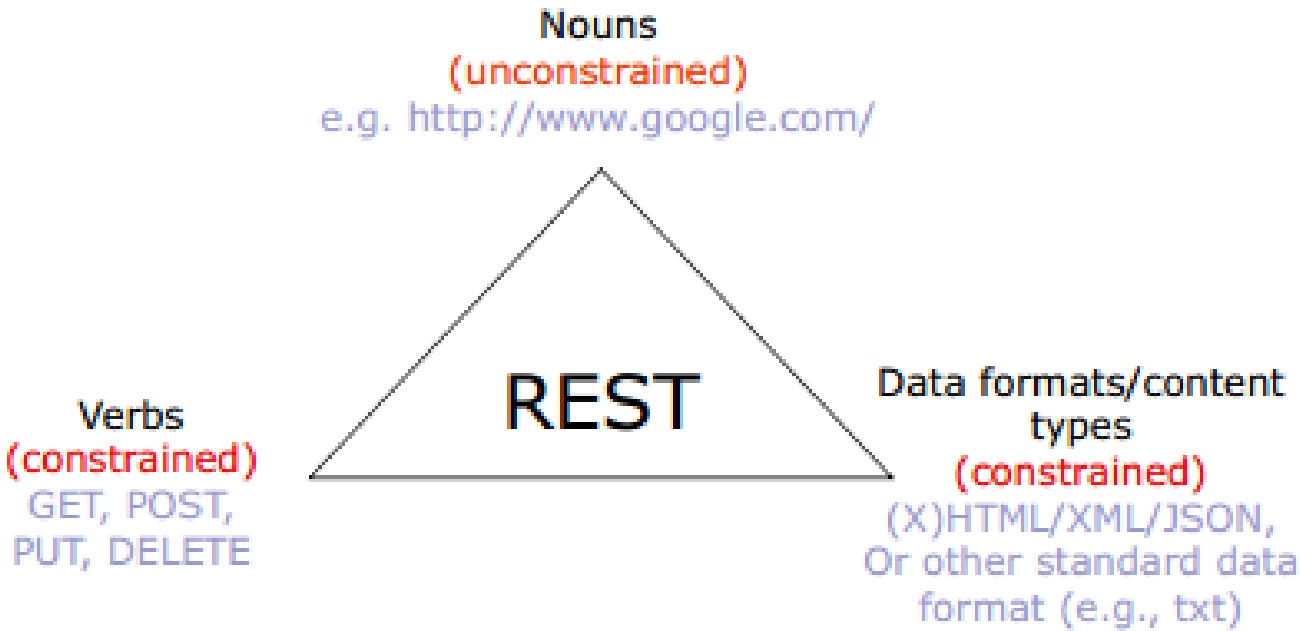


Figure: REST triangle at `https://wso2.com/files/2_triangle_0.png`

Verbs - HTTP Methods

Task	Method	Path
Create a new employee	POST	/employees
Delete an existing employee	DELETE	/employees/{id}
Get a specific employee	GET	/employees/{id}
Get a collection of employees	GET	/employees
Update an existing employee	PUT	/employees/{id}

Figure: CRUD verbs at `http://www.kennethlange.com/img/restful_2/rest_http_methods.png`

CRUD for Resources

- Create
  - PUT - create resource (client chooses URI)
  - POST - create resource (server chooses URI)
- Read
  - GET - fetch resource
- Update
  - PUT - update resource
- Delete
  - DELETE - remove resource

Nouns - Identification of Resources

- Addressable through URIs
- Example: http://search/query?term=foo

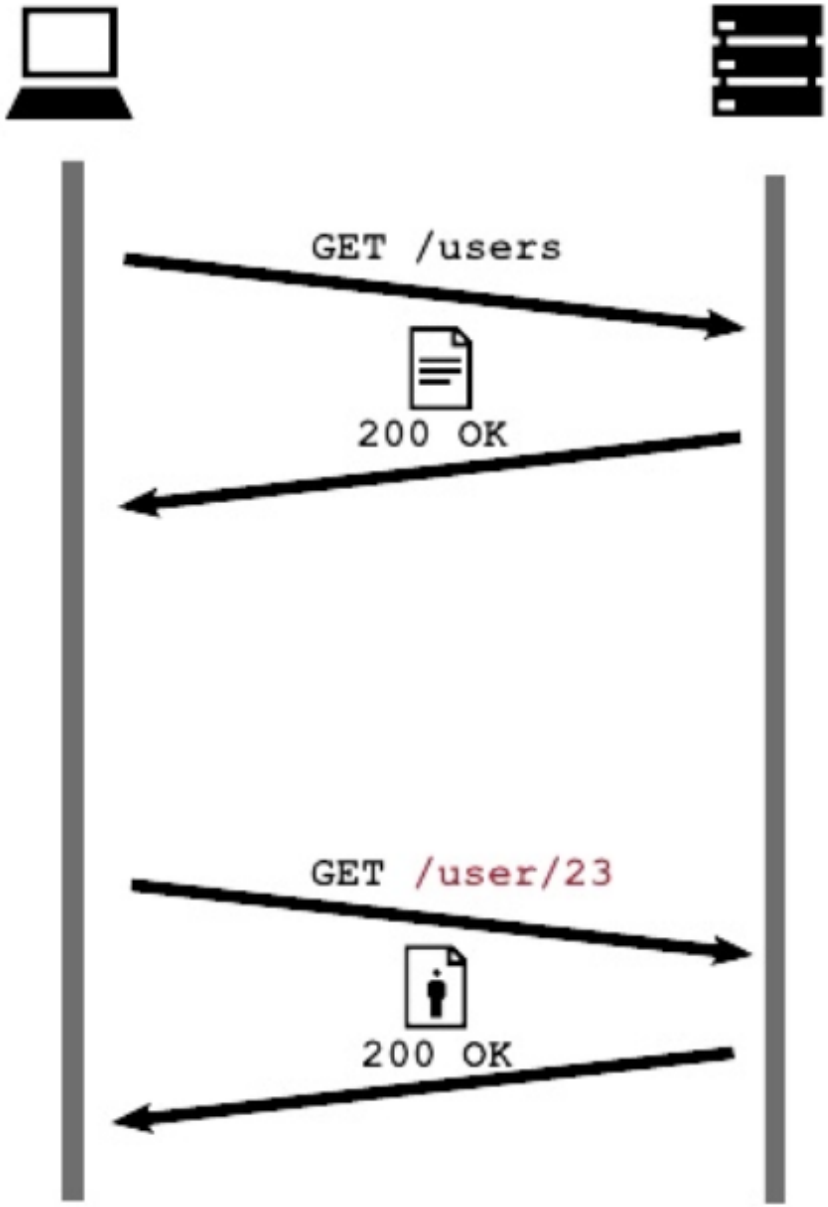
Hypermedia - Linking resources

/users

```
{"users" : [
  {
   "id": 23,
   "link" : "/user/23"
  },
  {
   "id": 42,
   "link" : "/user/42"
  }
]}
```

/user/23

```
{"user" : {
  "id" : 23,
  "name" : "foobar",
  "details" {
    ...
  }
}}
```

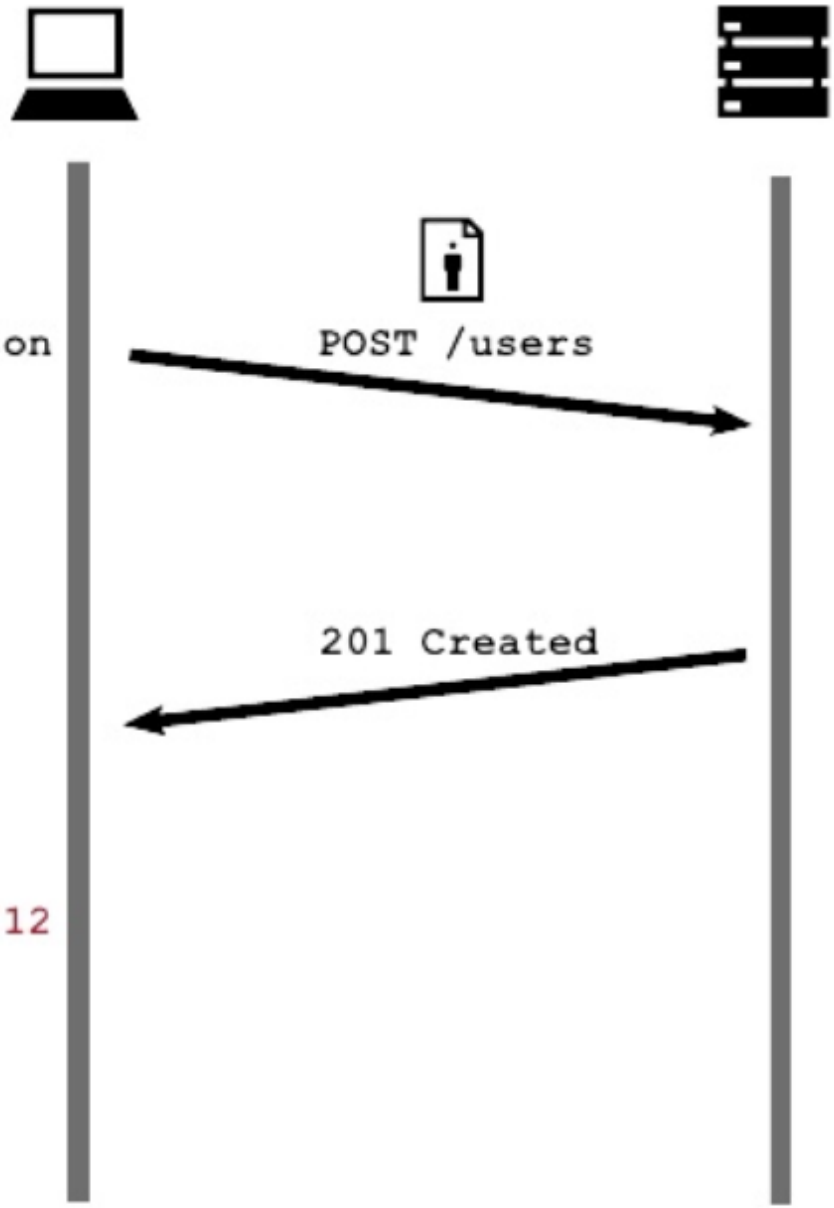


Hypermedia - Creating resources

```
POST /users HTTP/1.1
Host: foo.x
...
Content-Type: application/json

{"user" : {
  "name" : "qwerty",
  "details" {
    ...
  }
}}


HTTP/1.1 201 Created
Location: http://foo.x/user/12
...
```



Overview

Create, Delete and Update

- How to communicate with CouchDB?
  - cURL utility
  - Web interface Futon (or Fauxton)

Creating and deleting a database

```
curl -X PUT http://127.0.0.1:5984/db_name
curl -X DELETE http://127.0.0.1:5984/db_name
```

Creating and deleting a document

```
curl -X PUT http://127.0.0.1:5984/db_name/ -d
' { document} '
curl -X DELETE http://127.0.0.1:5984/db_name/db_id?_rev
```

Updating a document

```
curl -X PUT http://127.0.0.1:5984/db_name/doc_id/
-d  '{ "field" : "value", "_rev" : "revision id" }'
```

Fauxton - Web interface

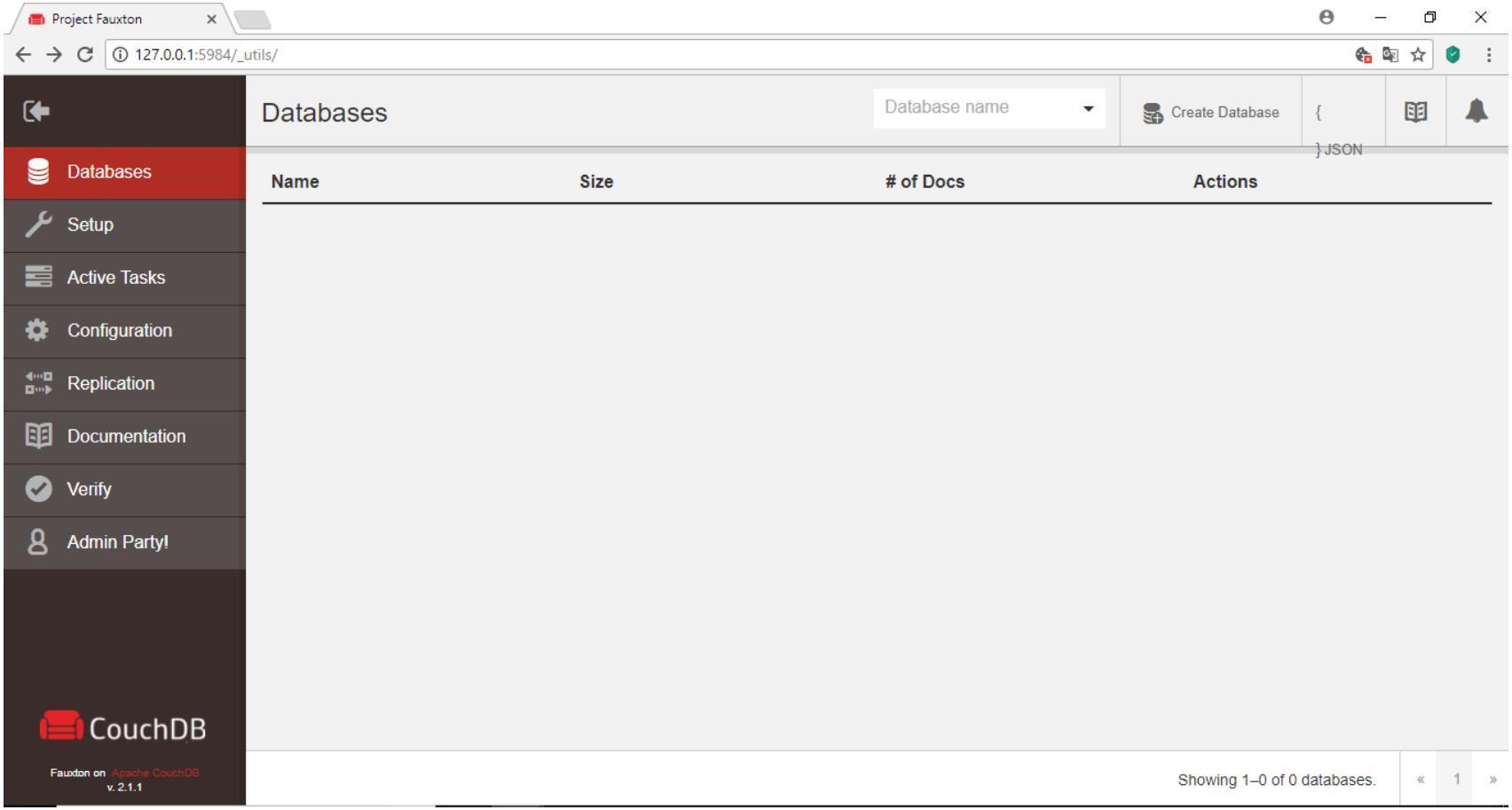


Figure: Fauxton Dashboard

Views

- Dynamic representation of document contents
- Purposes:
  - Querying the database for a specific data
  - Filtering documents
  - Building and using indexes (B-trees)
  - Make some calculations on the data
- MapReduce principle

Map function

- A single argument: document object.
- Results: *emit(key, value)* function.
- Each call of that function $\Rightarrow$ a row is added to the view.
- A view result stored in B+ tree

Example

```
function(doc) {
    emit(doc.course_name, doc.number_of_student);
}
```

Result of a map function

Example

```
{
"total_rows" : 2,
"offset": 0,
"rows": [
          {
          "id": "id1"
          "key": "INFO-H415 Advanced databases",
          "value": 40
          },
          {
          "id": "id2"
          "key": "INFO-H502 Virtual reality"
          "value": 35
          }
        ]
}
```

Reduce function

- Inputs: results of the map function or **results returned by the reduce function itself (rereduce)**

Example

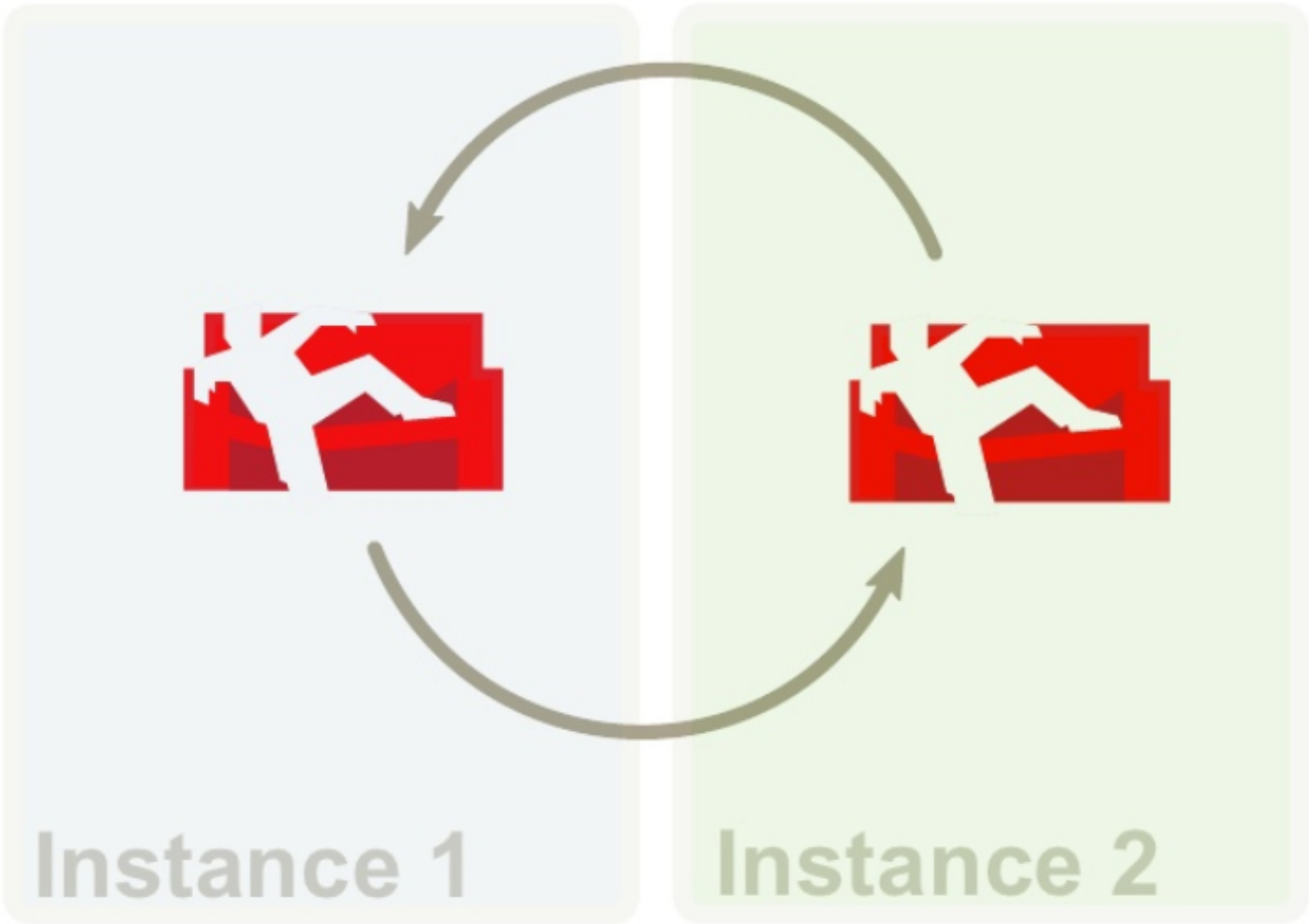
```
function(keys, values, rereduce){
    return sum(values);
}
```

- It will return the total number of students
- rereduce parameter is a boolean value. If rereduce equals to true → keys equal to null and values equal to the intermediate values returned by the reduce function.
  - useful while dealing with a large number of rows.

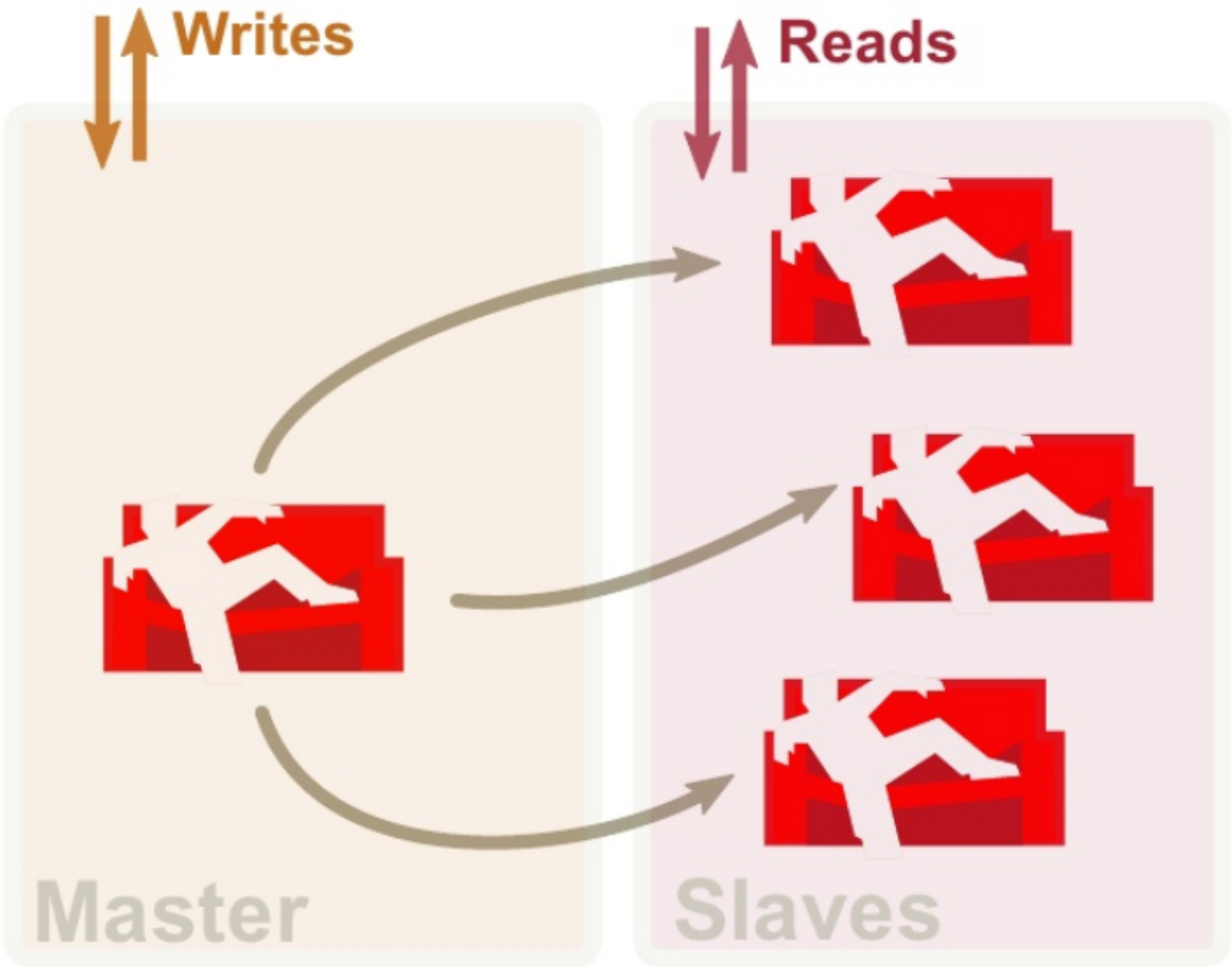
Replication

- Process of copying data such that databases share the same and redundant content. (source $\rightarrow$ target)
- For scalability: increasing the number of instances of a database $\Rightarrow$ less users read the data from one database $\Rightarrow$ latency decreases.
- Better availability and less data movements in a network
- Unidirectional or bidirectional.
- But... a good concurrency control is necessary.
  - CouchDB uses a MVCC (Multiversion Concurrency Control)

Replication - Example: Bidirectional

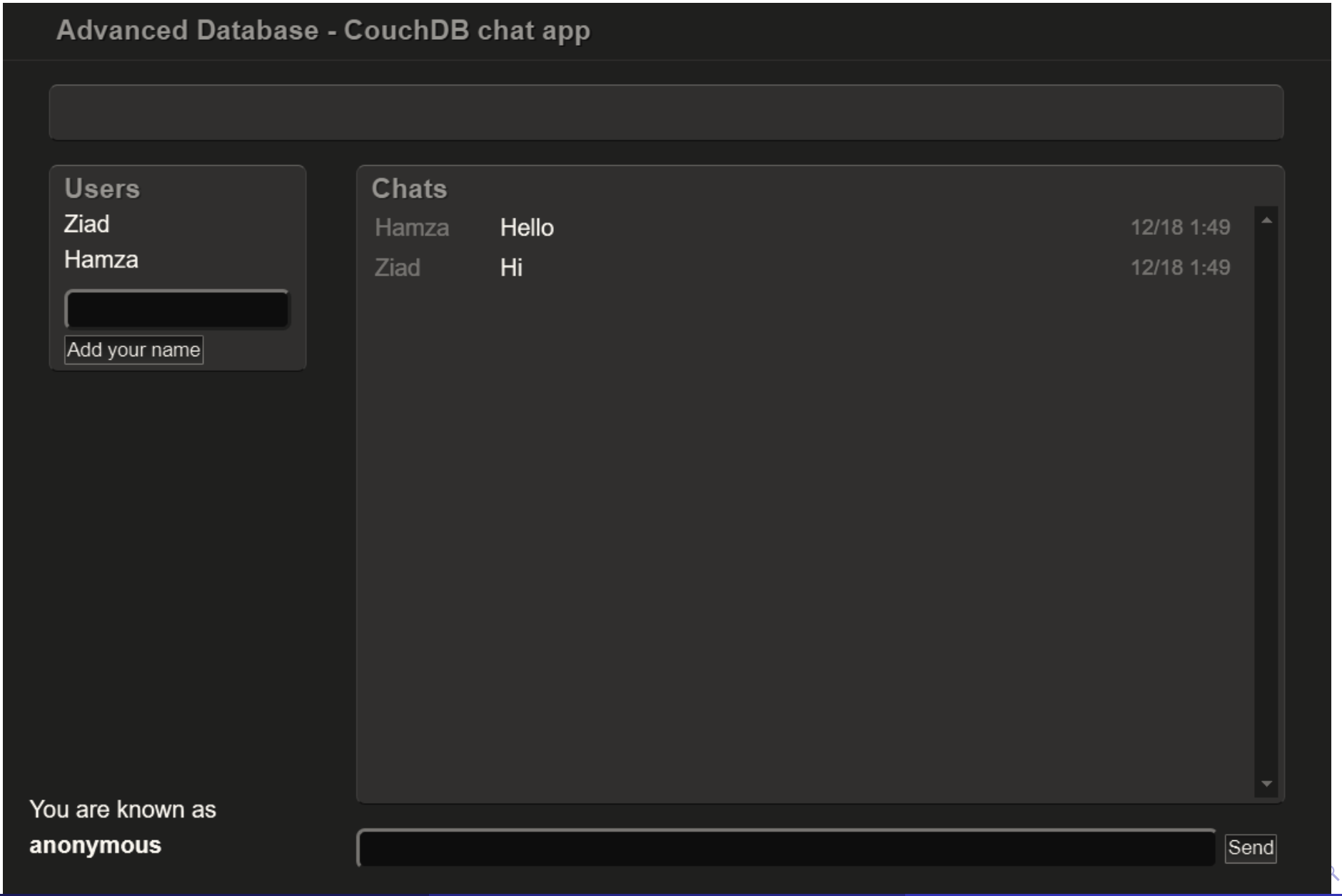


Replication - Example: Master-Slave



Overview

CouchDB Chat



CouchDB Chat

```
// Views
var add_view = function(name, doc) {
    db.get('_design/' + name, function(err, rs) {
        if (err && err.error == 'not_found') {
            db.save('_design/' + name, doc);
        }
    });
};

// Users view
add_view('users', {
    all: {
        map: function(doc) {
            if (doc.type == 'user') {
                emit(null);
            }
        }
    },
});
```

```
// Socket.IO
io.on('connection', (socket) => {
    // Retrieve the user list from CouchDB and send it to the socket
    db.view('users/all', {
        include_docs: true
    }, (err, rows) => {
        var list = [];
        if (rows) {
            rows.forEach((row) => {
                list.push(row);
            });
        }

        socket.emit('chat message', {users: list});

        // Send a few latest chats to the socket
        db.view('chats/all', {
            include_docs: true
        }, (err, rows) => {
            if (rows) {
                socket.emit('chat message', {history: rows});
            }
        });
```

CouchDB Chat

```
var chat = {
    type: 'chat',
    message: data.chat,
    from: data.user,
    timestamp: Date.now()
};
db.post(chat, (err, res) => {
    // socket.send(chat);
    io.emit('chat message', chat);
});
```

Figure: Posting a message

CouchDB Chat

```
var user = {
    type: 'user',
    name: data.addUser,
    status: 'online'
};
db.post(user, (err, res) => {
    db.view('users/all', {
        include_docs: true
    }, (err, rows) => {
        var list = [];
        if (rows) {
            rows.forEach((row) => {
                list.push(row);
            });
        }
        io.emit('chat message', {users: list})
    });
});
```

Figure: Posting a user

References I

- *Apache couchdb 2.1 documentation*, http://docs.couchdb.org/en/2.1.1/, Accessed: 2017-12-17.
- *Introducing json*, https://www.json.org/index.html, Accessed: 2017-12-17.
- *Nosql databases explained*, https://www.mongodb.com/nosql-explained, Accessed: 2017-12-17.
- *Nosql (not only sql database)*, http://searchdatamanagement.techtarget.com/definition/NoSQL-Not-Only-SQL, Accessed: 2017-12-15.
- *Introduction to rest and couchdb*, https://www.slideshare.net/partlycloudy/introduction-to-rest-couchdb, Accessed: 2017-12-17.

References II

- *Couchdb tutorial*, https://www.tutorialspoint.com/couchdb/, Accessed: 2017-12-17.
- *Http view api*, https://wiki.apache.org/couchdb/HTTP_view_API, Accessed: 2017-12-17.

Any questions?