JNoSQL

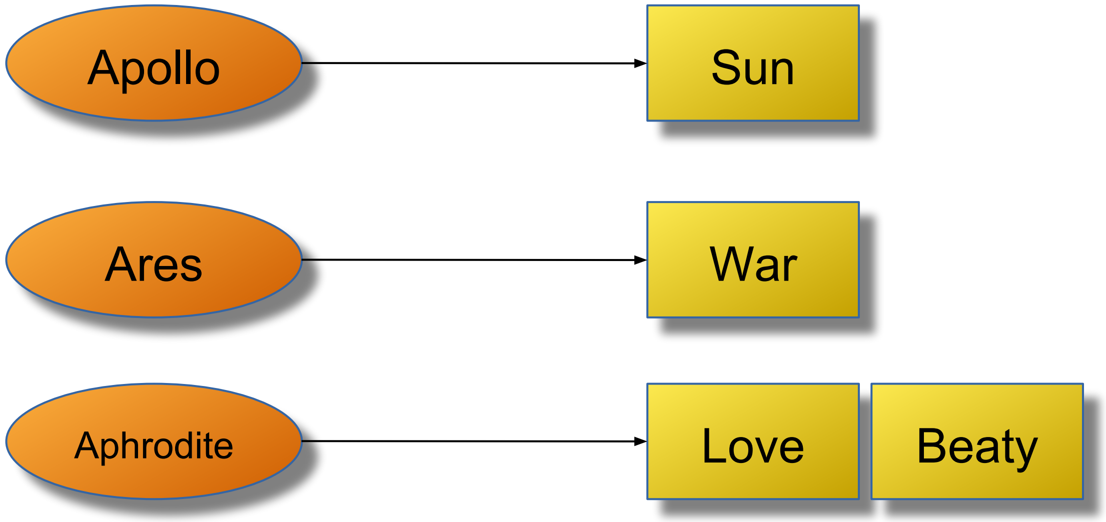
Otávio Santana
@otaviojava
otaviojava@apache.org

NoSQL

- Database
- Doesn't use structure
- Not Transaction
- BASE
- Five different types

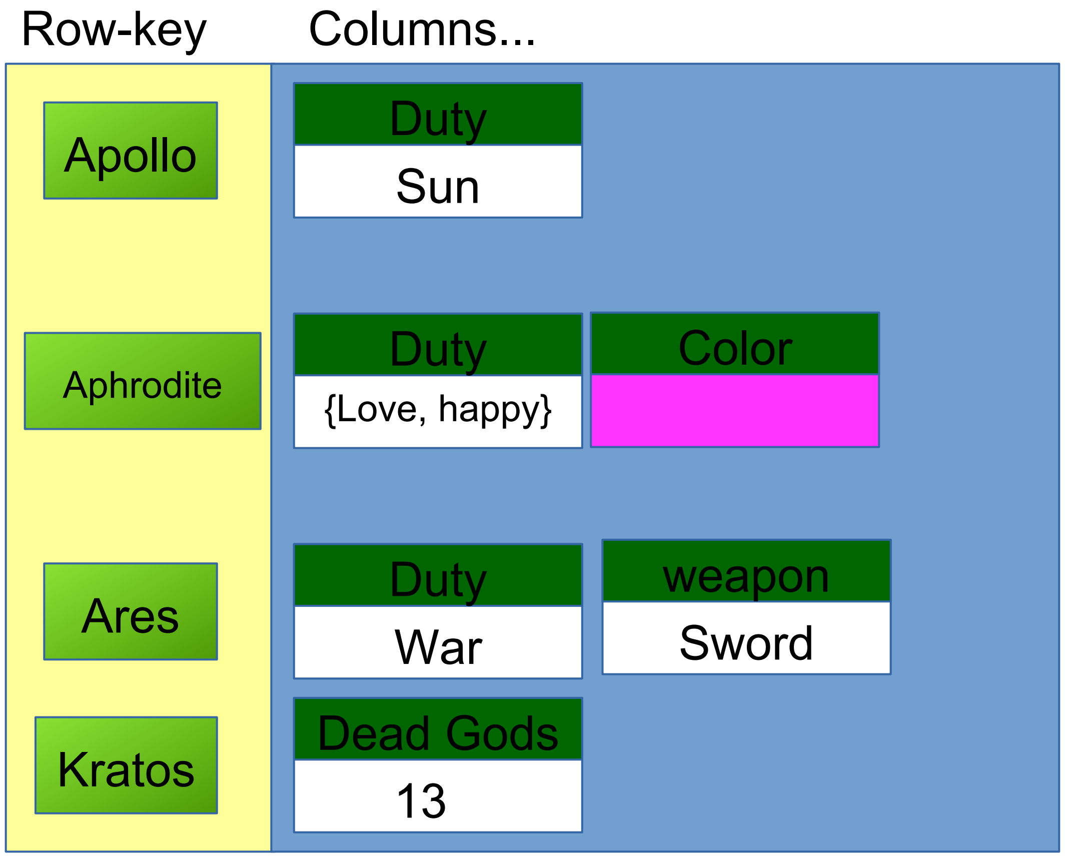
Key Value

- AmazonDynamo
- AmazonS3
- **Redis**
- Hazelcast



Column Family

- Hbase
- **Cassandra**
- Scylla
- Clouddata
- SimpleDb
- DynamoDB



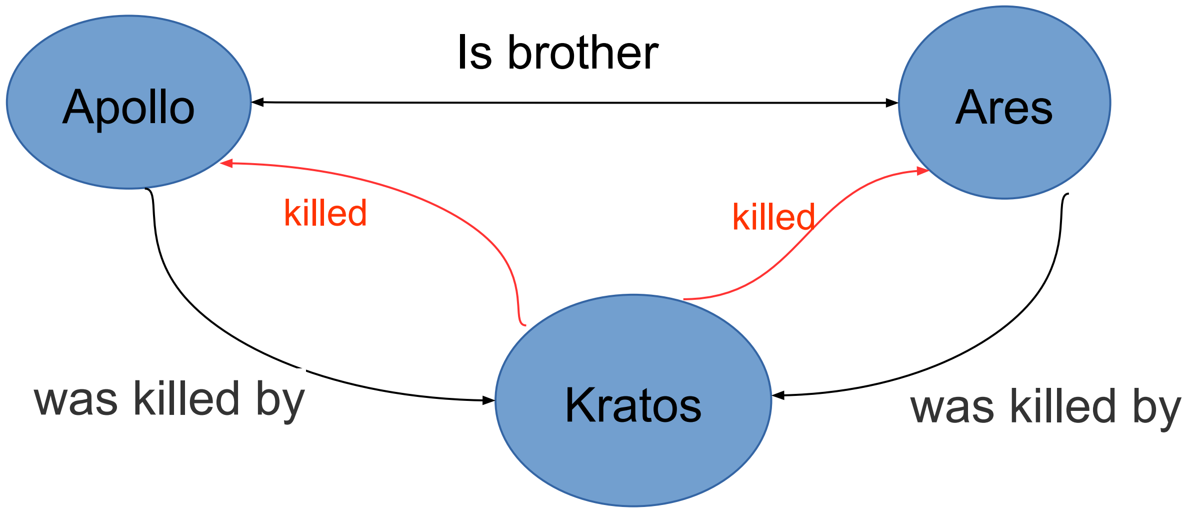
Document

- ApacheCouchDB
- **MongoDB**
- Riak
- Couchbase

```
{
  "name":"Diana",
  "duty":[
    "Hunt",
    "Moon",
    "Nature"
  ],
  "siblings":{
    "Apollo":"brother"
  }
}
```

Graph

- **Neo4j**
- InfoGrid
- Sones
- HyperGraphDB



Multi-Model

- OrientDB (graph, document)
- Couchbase (key value, document)
- Elasticsearch (document, graph)
- ArangoDB (column family, graph, key-value)

SQL vs NoSQL

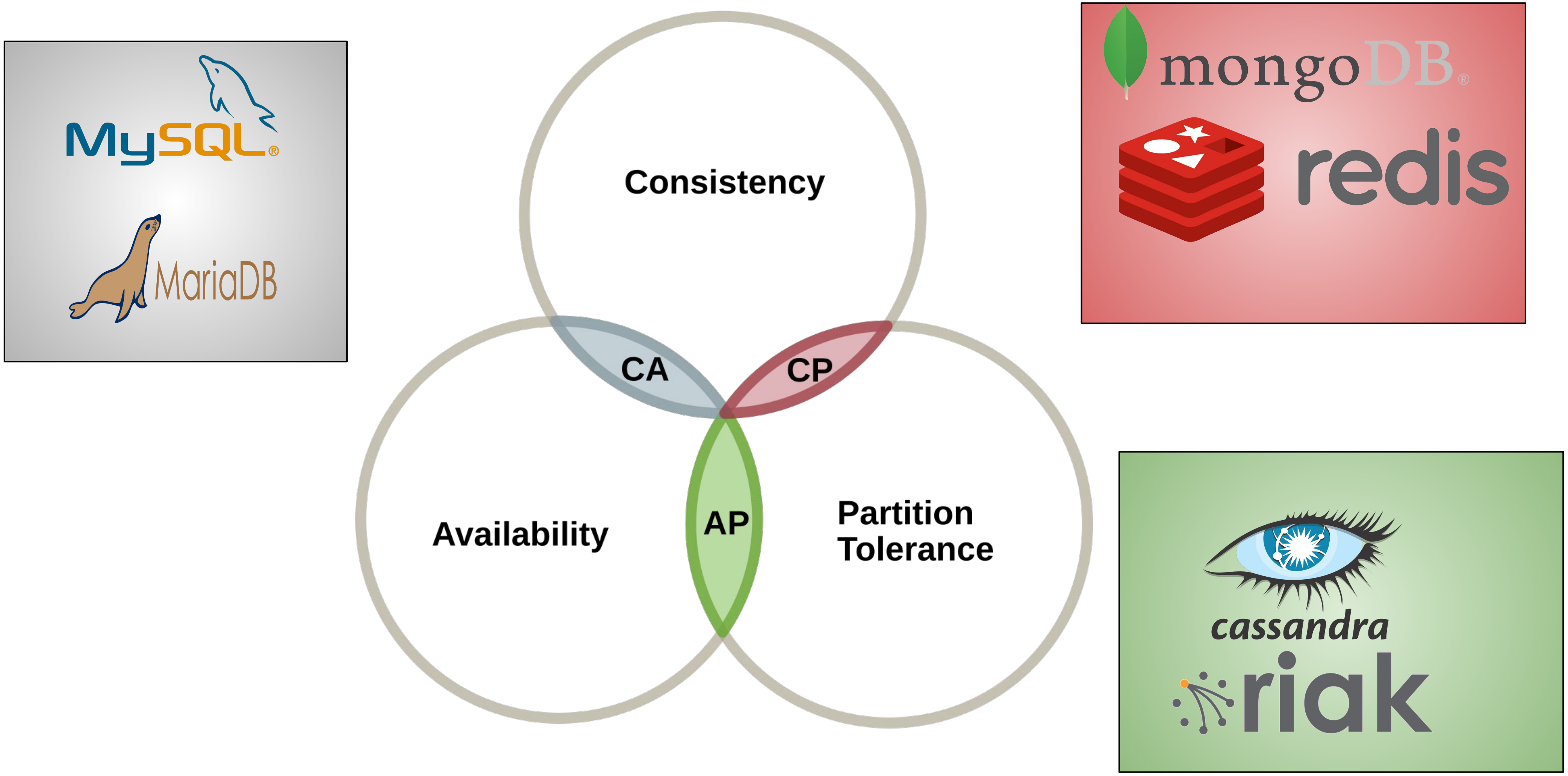
SQL	Key-value	Column	Document	Graph
Table	Bucket	Column Family	Collection	
Row	Key/value pair	Column	Document	Vertex
Column		Key/value pair	Key/value pair	Vertex and Edge property
Relationship			Link	Edge

BASE vs ACID

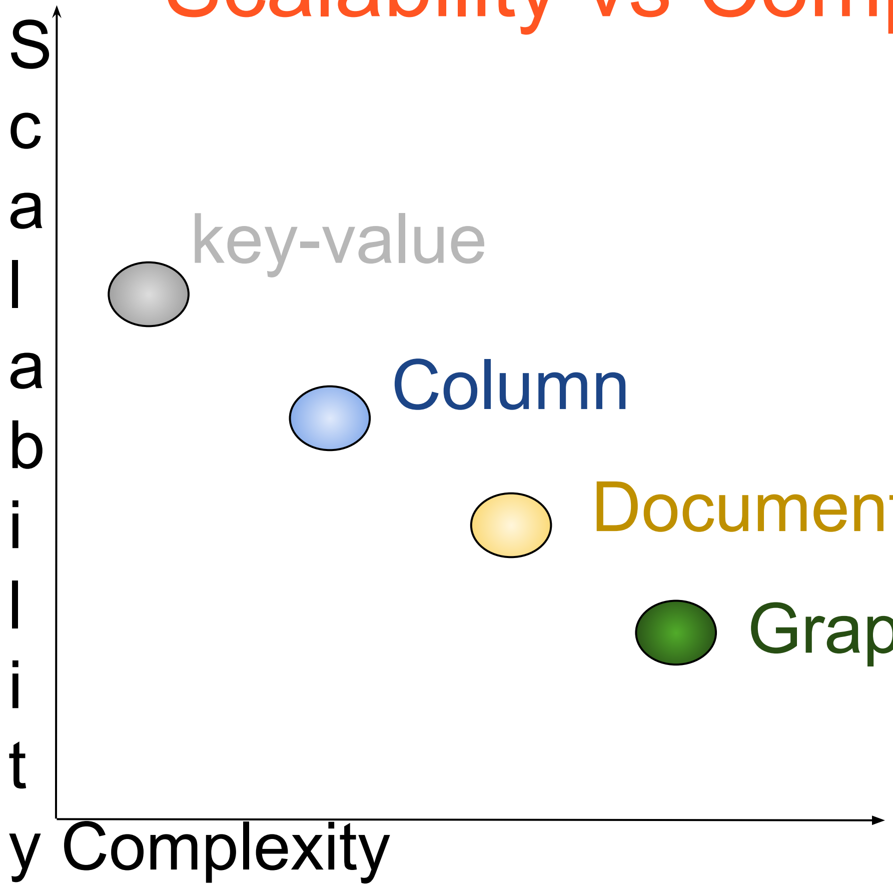
- **B**asically **A**vailable
- **S**oft state
- **E**ventual consistency

- **A**tomicity
- **C**onsistency
- **I**solation
- **D**urability

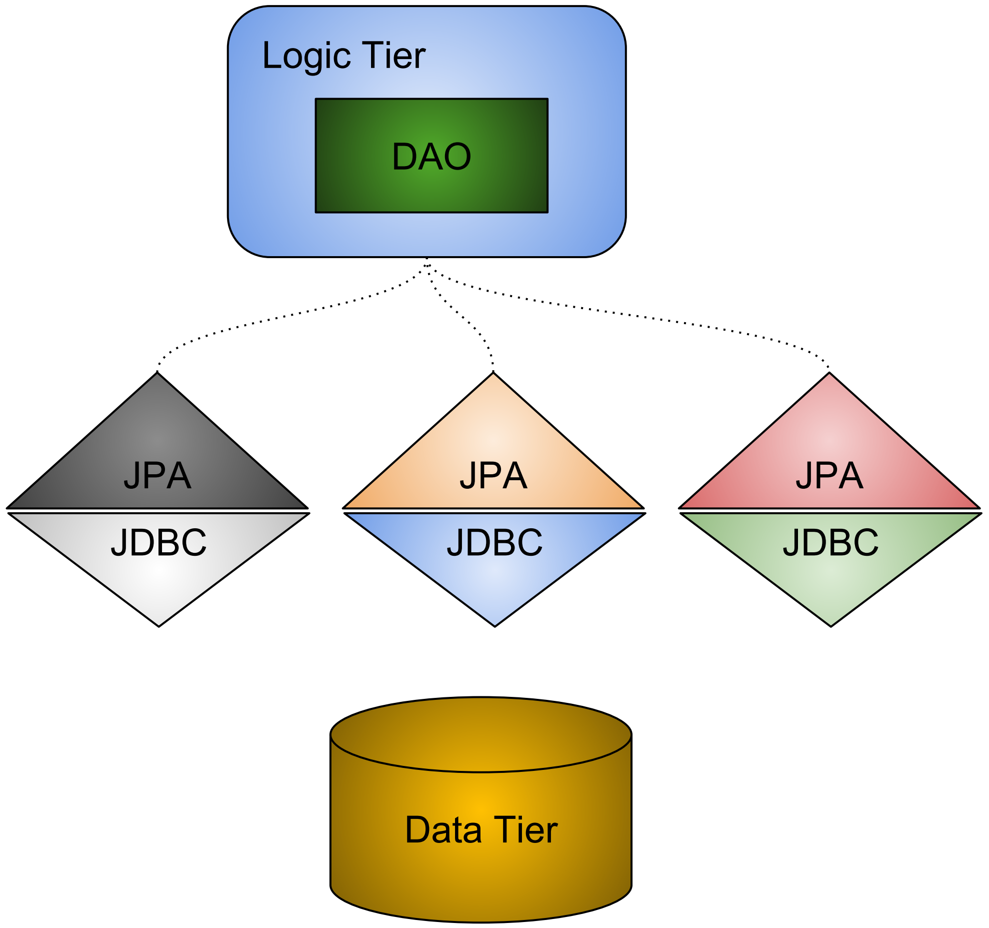
CAP


MySQL
MariaDB
Consistency
CA
CP
Availability
AP
Partition Tolerance
mongoDB
redis
cassandra
riak

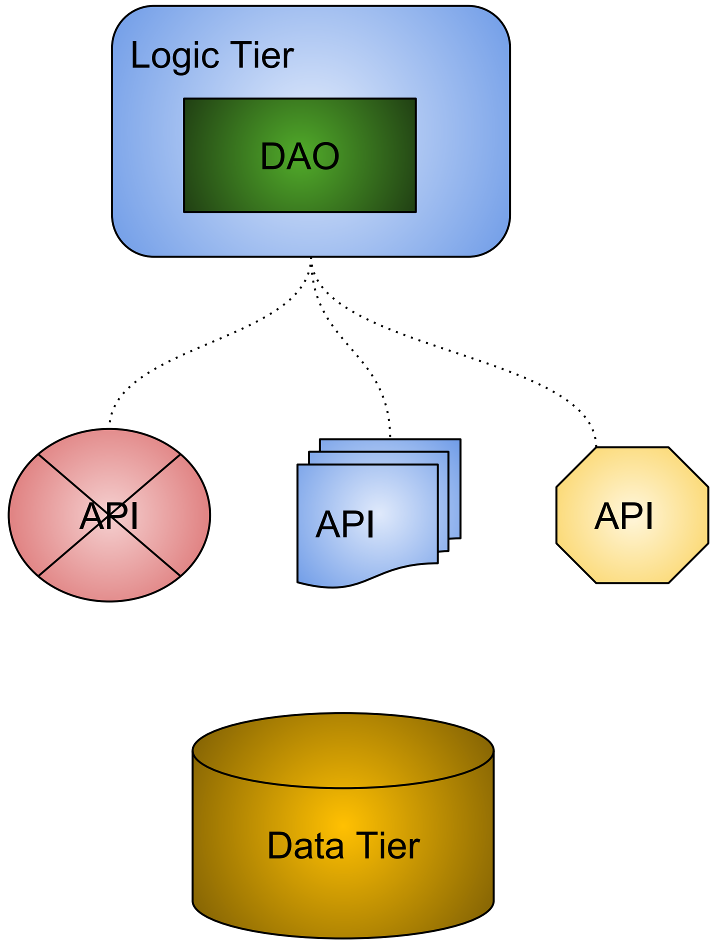
Scalability vs Complexity



Relational Application

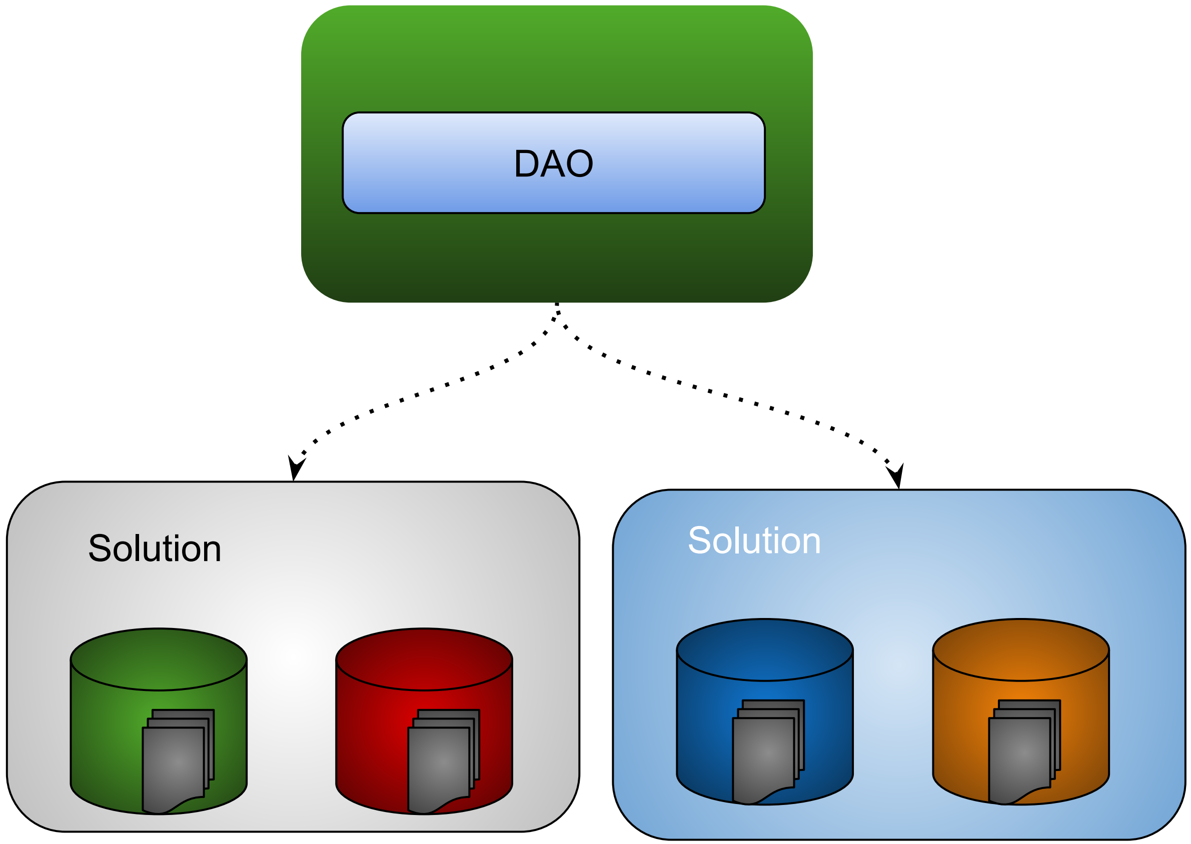


NoSQL Application



The Current Solution

- Hibernate OGM
- TopLink

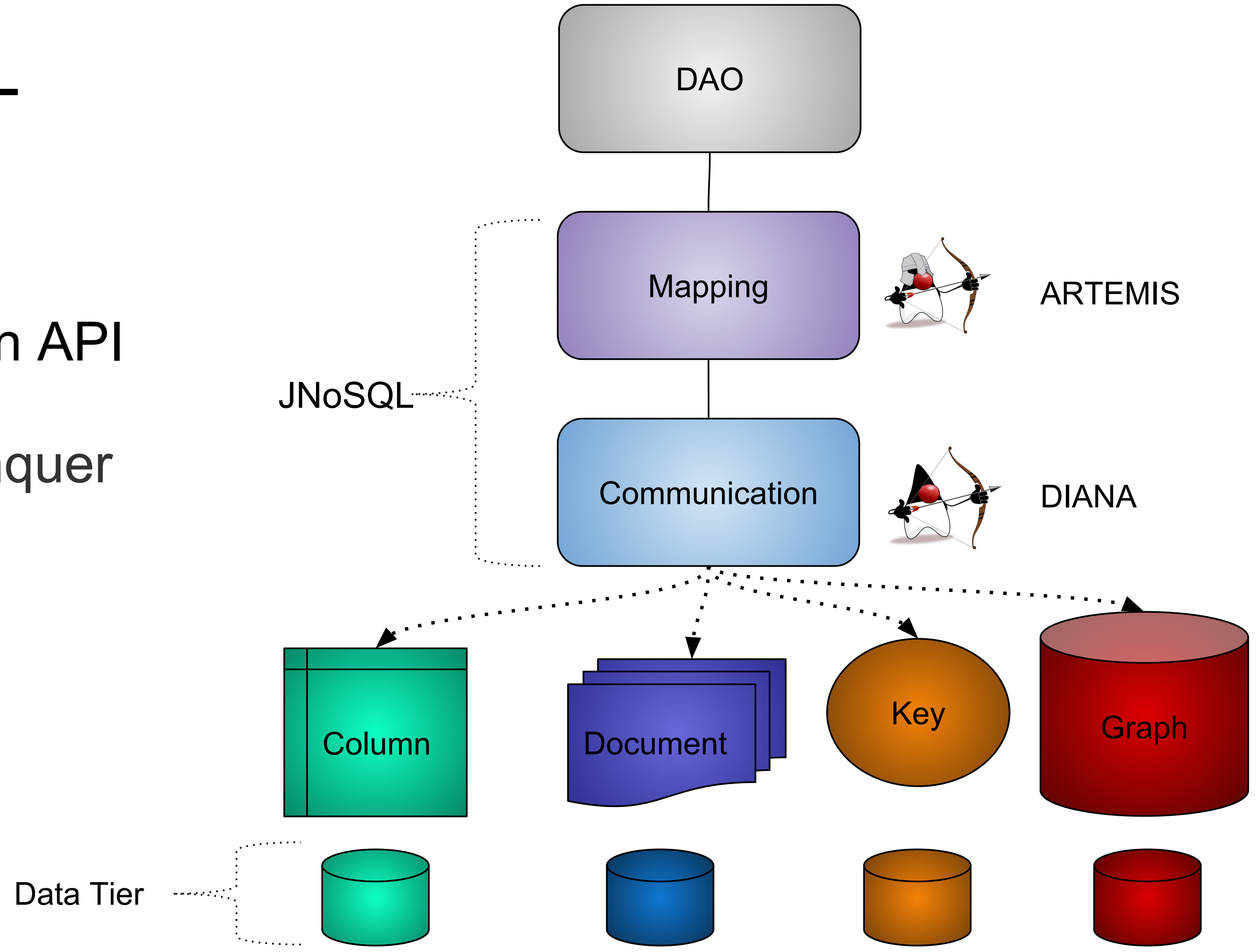


JPA problem for NoSQL

- Saves Async
- Async Callback
- Time to Live (TTL)
- Consistency Level
- SQL based
- Diversity in NoSQL

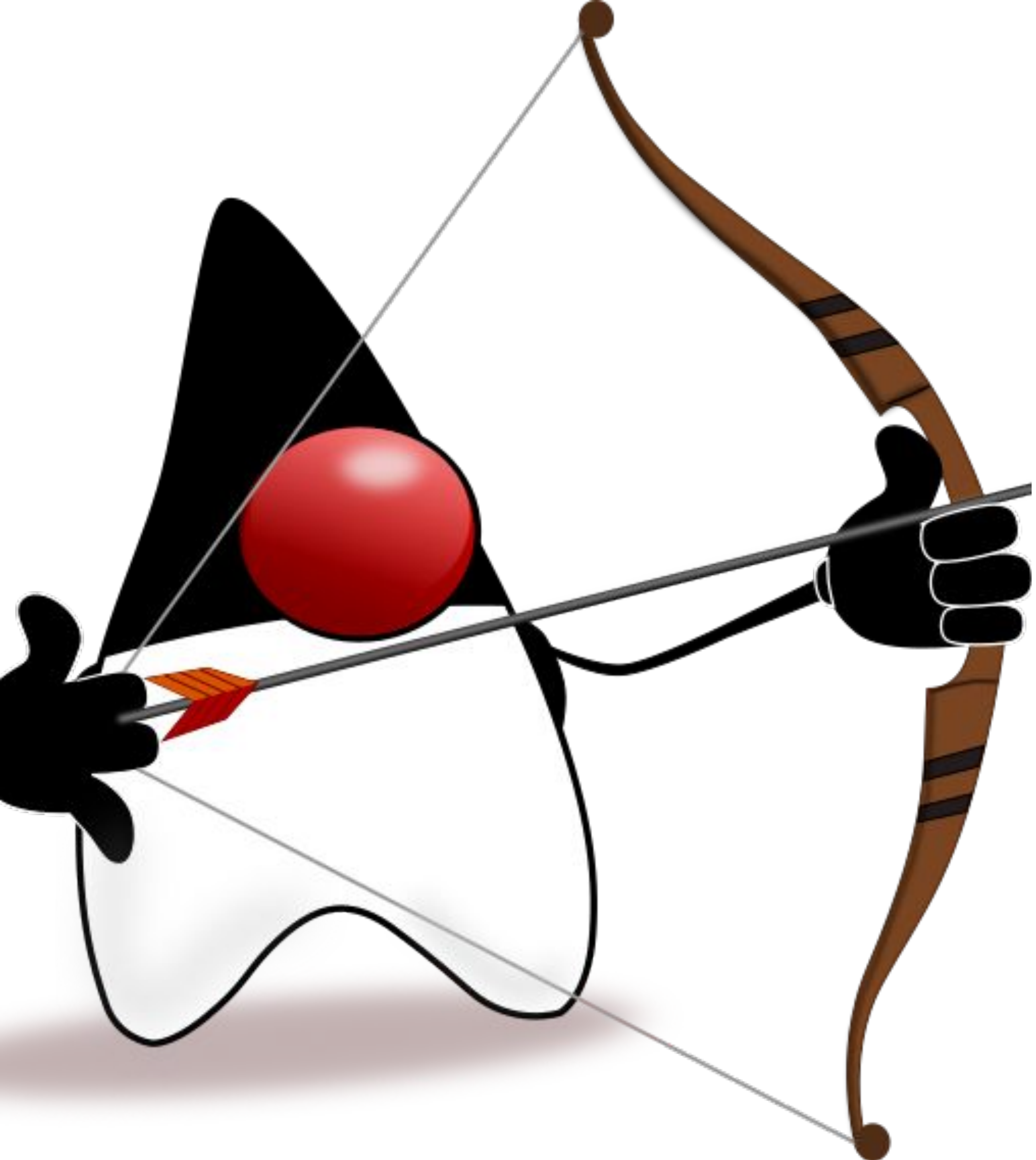
JNoSQL

- Mapping API
- Communication API
- No lock-in
- Divide and conquer



Why Diana?

- Goddess of the hunt, nature and moon
- Fought in Troy
- Brave warrior and hunter
- Diana Rome = Artemis Greek



Diana

- API Communication layer
- Document, key-value, Column, Graph (TinkerPop)

Communication Issue

ArangoDB

```
BaseDocument baseDocument = new
BaseDocument();
baseDocument.addAttribute(name, value);
```

mongoDB

```
Document document = new Document();
document.append(name, value);
```

Couchbase

```
JsonObject jsonObject = JsonObject.create();
jsonObject.put(name, value);
```

OrientDB

```
ODocument document = new ODocument(“collection”);
document.field(name, value);
```

Communication Issue

ArangoDB

```
BaseDocument baseDocument = new
BaseDocument();
baseDocument.addAttribute(name, value);
```

mongoDB

```
Document document = new Document();
document.append(name, value);
```

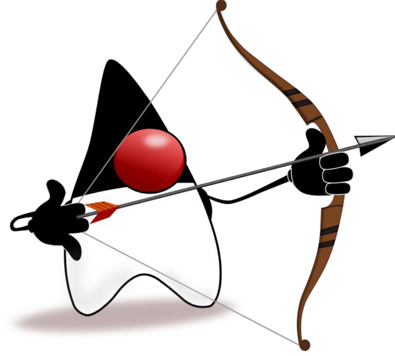
Couchbase

```
JsonObject jsonObject = JsonObject.create();
jsonObject.put(name, value);
```

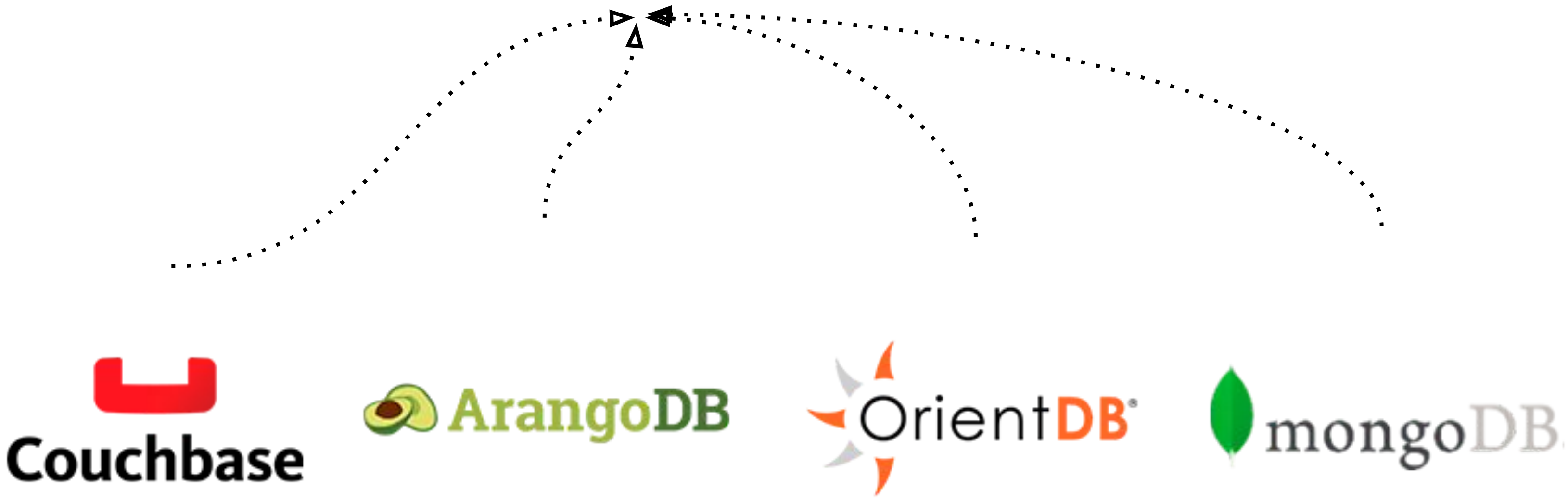
OrientDB

```
ODocument document = new ODocument(“collection”);
document.field(name, value);
```

Communication Issue



```
DocumentEntity entity =
DocumentEntity.of("documentCollection");
Document document = Document.of(name, value);
entity.add(document);
```



Names & definitions

- Configuration
- Factory
- Manager
- Entity

Names & definitions

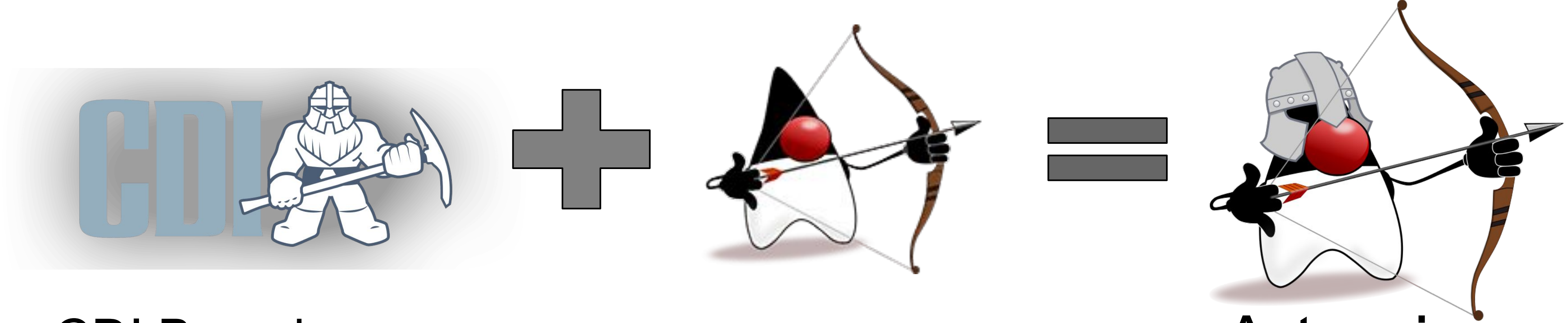
```
ColumnConfiguration<?> configuration = new DriverConfiguration();
try(ColumnFamilyManagerFactory managerFactory = configuration.get()) {
    ColumnFamilyManager entityManager = managerFactory.get(KEY_SPACE);
    entityManager.insert(entity);

    ColumnQuery select = select().from(COLUMN_FAMILY)
                                      .where("id").eq("Ada").build();
    ColumnDeleteQuery delete = delete().from(COLUMN_FAMILY)
                                      .where("id").eq("Ada").build();

   Optional<ColumnEntity> result = entityManager.singleResult(query);
   entityManager.delete(delete);
}
```

Diversity

```
ColumnEntity entity = ColumnEntity.of(COLUMN_FAMILY);
Column id = Column.of("id", 10L);
entity.add(id);
entity.add(Column.of("version", 0.001));
entity.add(Column.of("name", "Diana"));
entity.add(Column.of("options", Arrays.asList(1, 2, 3)));
```

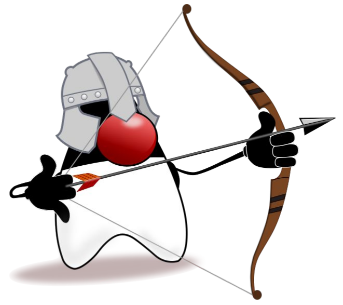
```
//mutiple implementation
entityManager.insert(entity);
ColumnQuery query =
select().from(COLUMN_FAMILY).where("id").eq(10L).build();
Optional<ColumnEntity> result = entityManager.singleResult(query);
```

```
//cassandra only
List<ColumnEntity> entities = entityManagerCassandra
.cql("select * from newKeySpace.newColumnFamily where id=10;");
entityManagerCassandra.insert(entity, ConsistencyLevel.ALL);
```



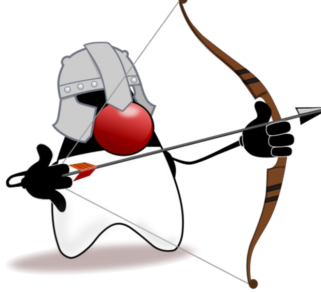
- CDI Based
- Diana Based
- Annotation Based
- Events to insert, delete, update
- Supports to Bean Validation
- Configurable and Extensible
- Query Method

Names & definitions



- Annotated Entities
- Template
- Repository
- Configuration

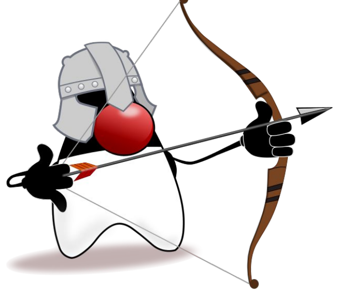
Annotated Entities



- MappedSuperclass
- Entity
- Column

```
@Entity("god")
public class God {

   @Column
   private String name;

   @Column
   private long age;

   @Column
   private Set<String> powers;
```

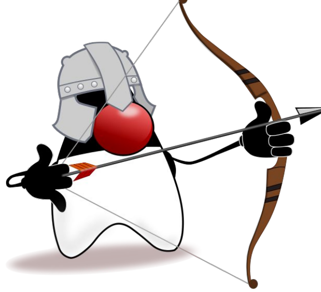
Template



God artemis = ...;
DocumentTemplate template = …
template.**insert**(artemis);
template.**update**(artemis);

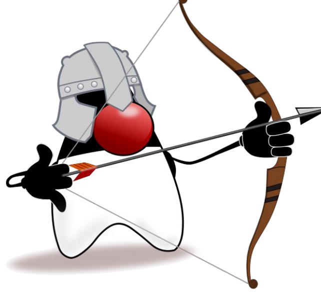
DocumentQuery query = *...*
List<God> gods = template.**select**(query);

Repository



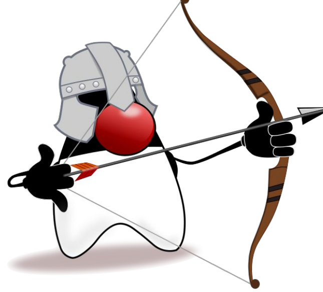
```
interface GodRepository extends Repository<God, String> {

    Optional<God> findByName(String name);

    Stream<God> findByNameAndAgeOrderByName(String name, Integer age);

}
```

Repository



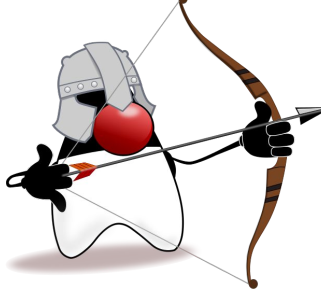
```
@Inject
@Database(DatabaseType.COLUMN)
private GodRepository godRepository;

@Inject
@Database(DatabaseType.KEY_VALUE)
private GodRepository godRepository;
```

Configuration



```
[
 {
  "description": "The couchbase document configuration",
  "name": "document",
  "provider": "org.jnosql.diana.couchbase.document.CouchbaseDocumentConfiguration",
  "settings": {
    "couchbase-host-1": "localhost",
    "couchbase-user": "root",
    "couchbase-password": "123456"
  }
 }
]
```

Configuration



```
@Inject
@ConfigurationUnit
private DocumentCollectionManagerFactory<?> entityManager;
```

Diversity

```
@Entity("god")
public class God {

@Column
private String name;

@UDT("weapon")
@Column
private Weapon weapon;

}
```

```
interface GodRepository extends
CassandraRepository<God, String> {

  @CQL("select * from God where name = ?")
  List<God> findByName(String name);

}
```

Demo

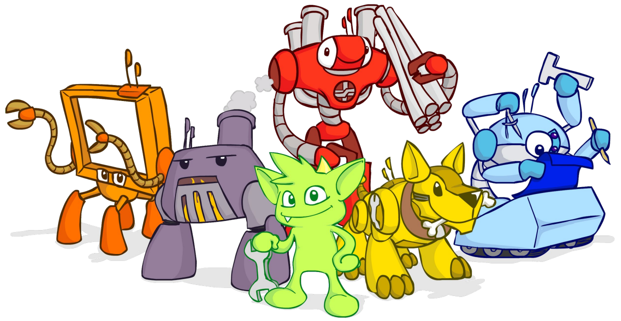
JNoSQL

Configuration

CDI 2.0 with Java SE

Couchbase + Docker

NoSQL Providers



cassandra

Couchbase

mongoDB

ORACLE
NOSQL DATABASE

ArangoDB

riak

SCYLLA.

elastic

hazelcast

APACHE HBASE

neo4j

OrientDB

redis

Many more...

JUGs/Communities

SOUJava
sociedade de usuários java

JavaBahia
Grupo de Usuários Java da Bahia

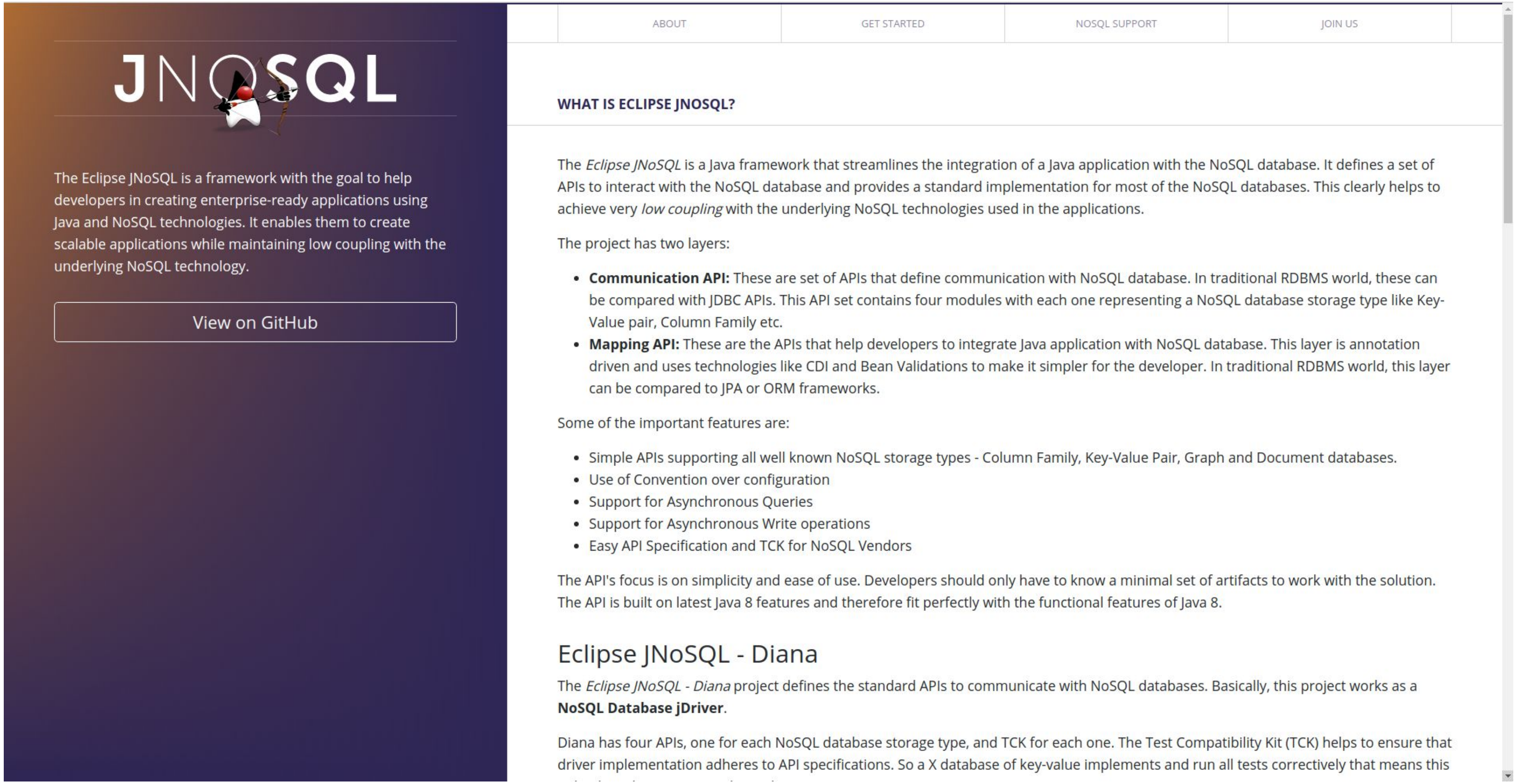
LJC
London Java Community

GUJavaSC
Grupo de Usuários Java de Santa Catarina

Road Map

- ✓ Draft and code proposal
- ✓ Community Feedback
- ✓ Involve NoSQL Vendors
- ✓ Involve Solution Vendors
- ✓ Eclipse Project
- ✓ Development

Site



The Eclipse JNoSQL is a framework with the goal to help developers in creating enterprise-ready applications using Java and NoSQL technologies. It enables them to create scalable applications while maintaining low coupling with the underlying NoSQL technology.

WHAT IS ECLIPSE JNOSQL?

The *Eclipse JNoSQL* is a Java framework that streamlines the integration of a Java application with the NoSQL database. It defines a set of APIs to interact with the NoSQL database and provides a standard implementation for most of the NoSQL databases. This clearly helps to achieve very *low coupling* with the underlying NoSQL technologies used in the applications.

The project has two layers:

- **Communication API:** These are set of APIs that define communication with NoSQL database. In traditional RDBMS world, these can be compared with JDBC APIs. This API set contains four modules with each one representing a NoSQL database storage type like Key-Value pair, Column Family etc.
- **Mapping API:** These are the APIs that help developers to integrate Java application with NoSQL database. This layer is annotation driven and uses technologies like CDI and Bean Validations to make it simpler for the developer. In traditional RDBMS world, this layer can be compared to JPA or ORM frameworks.

Some of the important features are:

- Simple APIs supporting all well known NoSQL storage types - Column Family, Key-Value Pair, Graph and Document databases.
- Use of Convention over configuration
- Support for Asynchronous Queries
- Support for Asynchronous Write operations
- Easy API Specification and TCK for NoSQL Vendors

The API's focus is on simplicity and ease of use. Developers should only have to know a minimal set of artifacts to work with the solution. The API is built on latest Java 8 features and therefore fit perfectly with the functional features of Java 8.

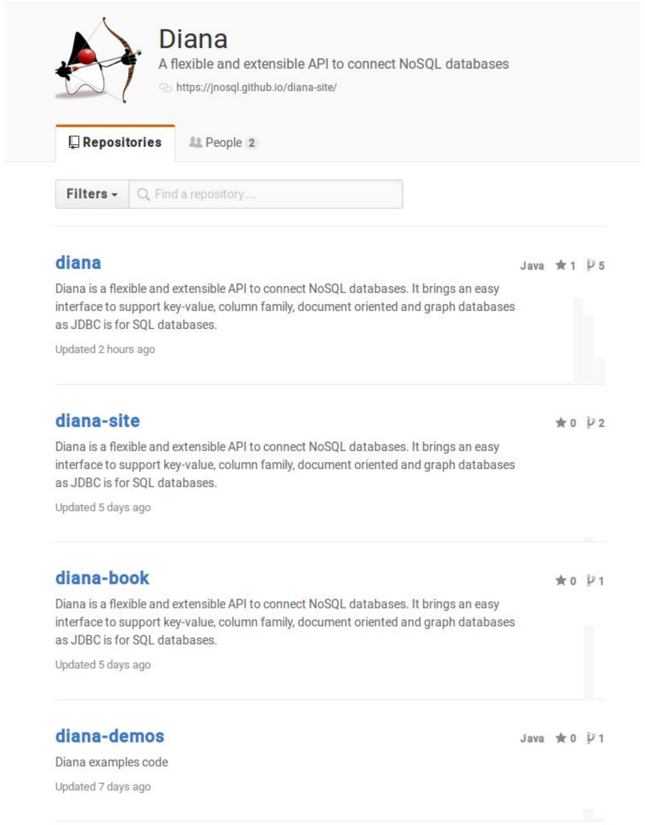
Eclipse JNoSQL - Diana

The *Eclipse JNoSQL - Diana* project defines the standard APIs to communicate with NoSQL databases. Basically, this project works as a **NoSQL Database jDriver**.

Diana has four APIs, one for each NoSQL database storage type, and TCK for each one. The Test Compatibility Kit (TCK) helps to ensure that driver implementation adheres to API specifications. So a X database of key-value implements and run all tests correctively that means this

http://jnosql.org/

Code



Diana

A flexible and extensible API to connect NoSQL databases

https://jnosql.github.io/diana-site/

Repositories | People 2

Filters | Find a repository...

diana — Java ★ 1 ⑂ 5

Diana is a flexible and extensible API to connect NoSQL databases. It brings an easy interface to support key-value, column family, document oriented and graph databases as JDBC is for SQL databases.

Updated 2 hours ago

diana-site — ★ 0 ⑂ 2

Diana is a flexible and extensible API to connect NoSQL databases. It brings an easy interface to support key-value, column family, document oriented and graph databases as JDBC is for SQL databases.

Updated 5 days ago

diana-book — ★ 0 ⑂ 1

Diana is a flexible and extensible API to connect NoSQL databases. It brings an easy interface to support key-value, column family, document oriented and graph databases as JDBC is for SQL databases.

Updated 5 days ago

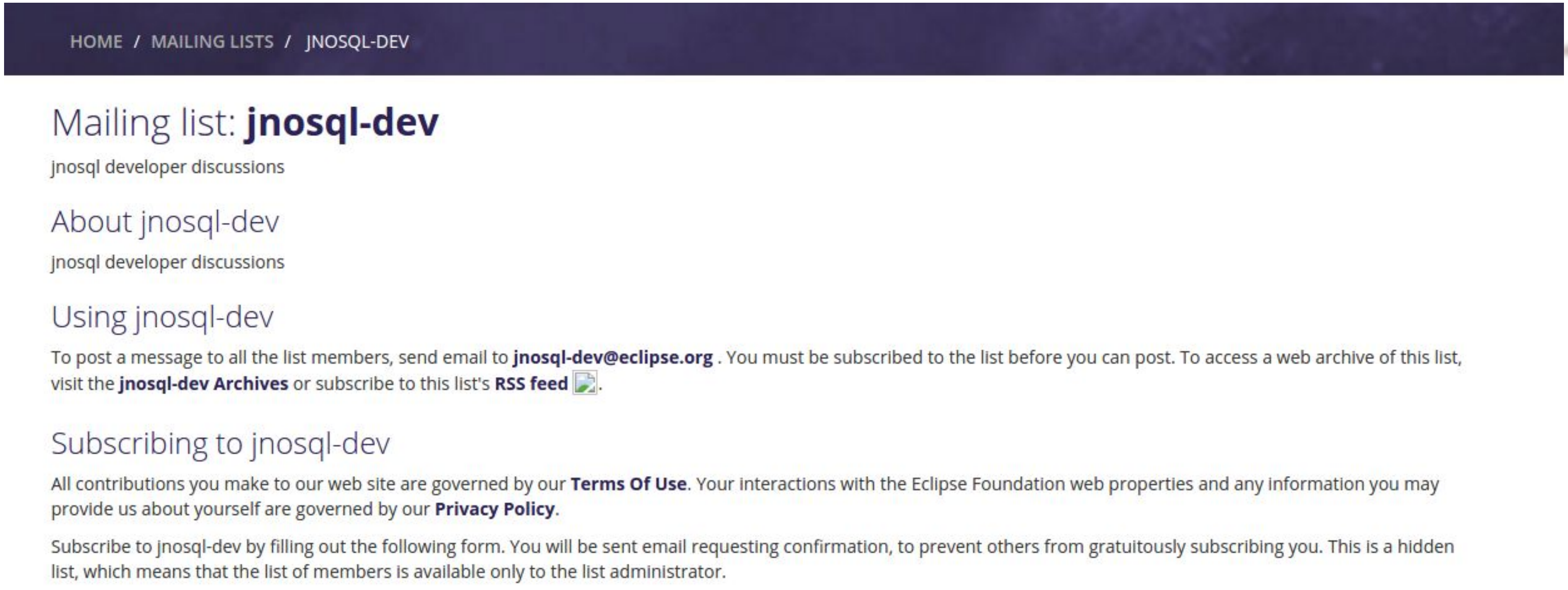
diana-demos — Java ★ 0 ⑂ 1

Diana examples code

Updated 7 days ago

https://github.com/JNOSQL

Mailing List



HOME / MAILING LISTS / JNOSQL-DEV

Mailing list: **jnosql-dev**

jnosql developer discussions

About jnosql-dev

jnosql developer discussions

Using jnosql-dev

To post a message to all the list members, send email to **jnosql-dev@eclipse.org** . You must be subscribed to the list before you can post. To access a web archive of this list, visit the **jnosql-dev Archives** or subscribe to this list's **RSS feed** .

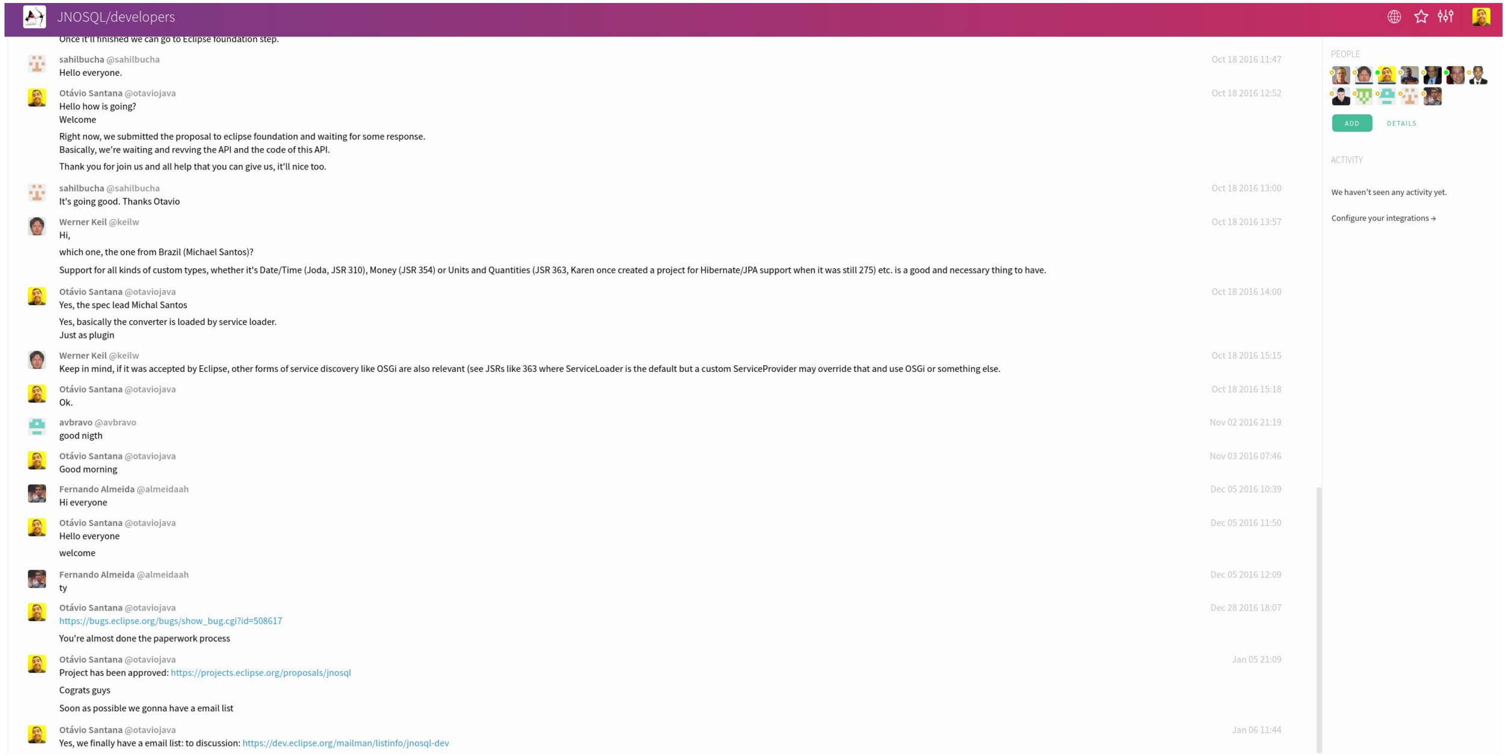
Subscribing to jnosql-dev

All contributions you make to our web site are governed by our **Terms Of Use**. Your interactions with the Eclipse Foundation web properties and any information you may provide us about yourself are governed by our **Privacy Policy**.

Subscribe to jnosql-dev by filling out the following form. You will be sent email requesting confirmation, to prevent others from gratuitously subscribing you. This is a hidden list, which means that the list of members is available only to the list administrator.

https://dev.eclipse.org/mailman/listinfo/jnosql-dev

Gitter



JNOSQL/developers

Once it'll finished we can go to Eclipse foundation step.

sahilbucha @sahilbucha — Oct 18 2016 11:47
Hello everyone.

Otávio Santana @otaviojava — Oct 18 2016 12:52
Hello how is going?
Welcome
Right now, we submitted the proposal to eclipse foundation and waiting for some response.
Basically, we're waiting and revving the API and the code of this API.
Thank you for join us and all help that you can give us, it'll nice too.

sahilbucha @sahilbucha — Oct 18 2016 13:00
It's going good. Thanks Otavio

Werner Keil @keilw — Oct 18 2016 13:57
Hi,
which one, the one from Brazil (Michael Santos)?
Support for all kinds of custom types, whether it's Date/Time (Joda, JSR 310), Money (JSR 354) or Units and Quantities (JSR 363, Karen once created a project for Hibernate/JPA support when it was still 275) etc. is a good and necessary thing to have.

Otávio Santana @otaviojava — Oct 18 2016 14:00
Yes, the spec lead Michal Santos
Yes, basically the converter is loaded by service loader.
Just as plugin

Werner Keil @keilw — Oct 18 2016 15:15
Keep in mind, if it was accepted by Eclipse, other forms of service discovery like OSGi are also relevant (see JSRs like 363 where ServiceLoader is the default but a custom ServiceProvider may override that and use OSGi or something else.

Otávio Santana @otaviojava — Oct 18 2016 15:18
Ok.

avbravo @avbravo — Nov 02 2016 21:19
good nigth

Otávio Santana @otaviojava — Nov 03 2016 07:46
Good morning

Fernando Almeida @almeidaah — Dec 05 2016 10:39
Hi everyone

Otávio Santana @otaviojava — Dec 05 2016 11:50
Hello everyone
welcome

Fernando Almeida @almeidaah — Dec 05 2016 12:09
ty

Otávio Santana @otaviojava — Dec 28 2016 18:07
https://bugs.eclipse.org/bugs/show_bug.cgi?id=508617
You're almost done the paperwork process

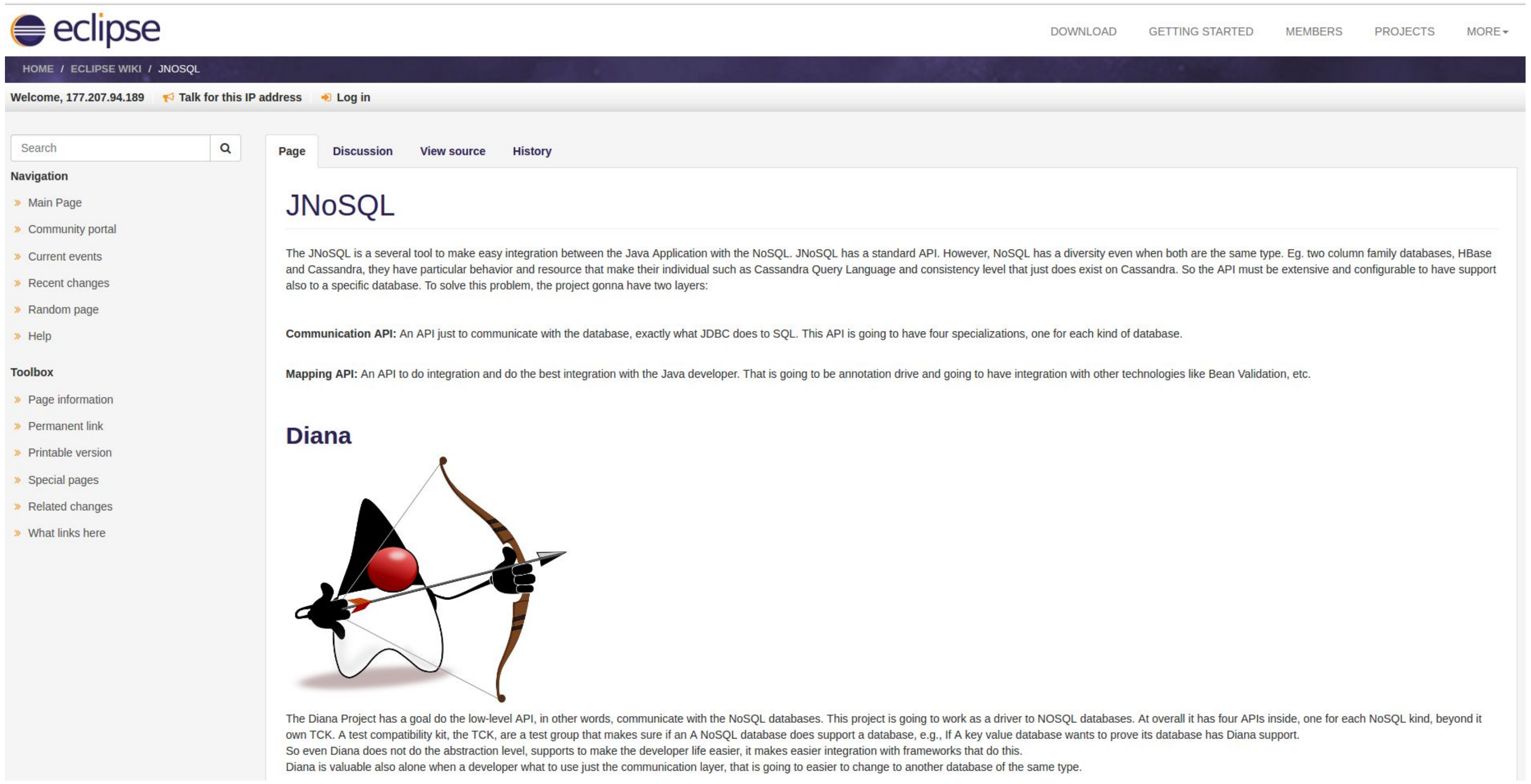
Otávio Santana @otaviojava — Jan 05 21:09
Project has been approved: https://projects.eclipse.org/proposals/jnosql
Cograts guys
Soon as possible we gonna have a email list

Otávio Santana @otaviojava — Jan 06 11:44
Yes, we finally have a email list: to discussion: https://dev.eclipse.org/mailman/listinfo/jnosql-dev

PEOPLE
ADD
DETAILS
ACTIVITY
We haven't seen any activity yet.
Configure your integrations →

https://gitter.im/JNOSQL/developers

Wiki



JNoSQL

The JNoSQL is a several tool to make easy integration between the Java Application with the NoSQL. JNoSQL has a standard API. However, NoSQL has a diversity even when both are the same type. Eg. two column family databases, HBase and Cassandra, they have particular behavior and resource that make their individual such as Cassandra Query Language and consistency level that just does exist on Cassandra. So the API must be extensive and configurable to have support also to a specific database. To solve this problem, the project gonna have two layers:

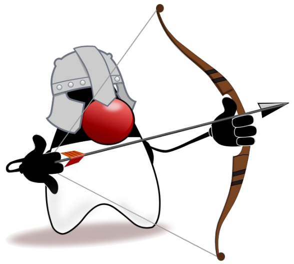
Communication API: An API just to communicate with the database, exactly what JDBC does to SQL. This API is going to have four specializations, one for each kind of database.

Mapping API: An API to do integration and do the best integration with the Java developer. That is going to be annotation drive and going to have integration with other technologies like Bean Validation, etc.

Diana

The Diana Project has a goal do the low-level API, in other words, communicate with the NoSQL databases. This project is going to work as a driver to NOSQL databases. At overall it has four APIs inside, one for each NoSQL kind, beyond it own TCK. A test compatibility kit, the TCK, are a test group that makes sure if an A NoSQL database does support a database, e.g., If A key value database wants to prove its database has Diana support.
So even Diana does not do the abstraction level, supports to make the developer life easier, it makes easier integration with frameworks that do this.
Diana is valuable also alone when a developer what to use just the communication layer, that is going to easier to change to another database of the same type.

https://wiki.eclipse.org/JNoSQL



Thank you

Otávio Santana
@otaviojava
osantana@tomitribe.com
otaviojava@apache.org