Spring Data Couchbase

Spring Data Couchbase

- **Integration with Spring Data**
  - Templates
  - Repositories
  - Exception Mapping
  - also @Cacheable

- **Maps POJO Entities to JSON (and back)**
- **2.0 M1 Released!**

Features

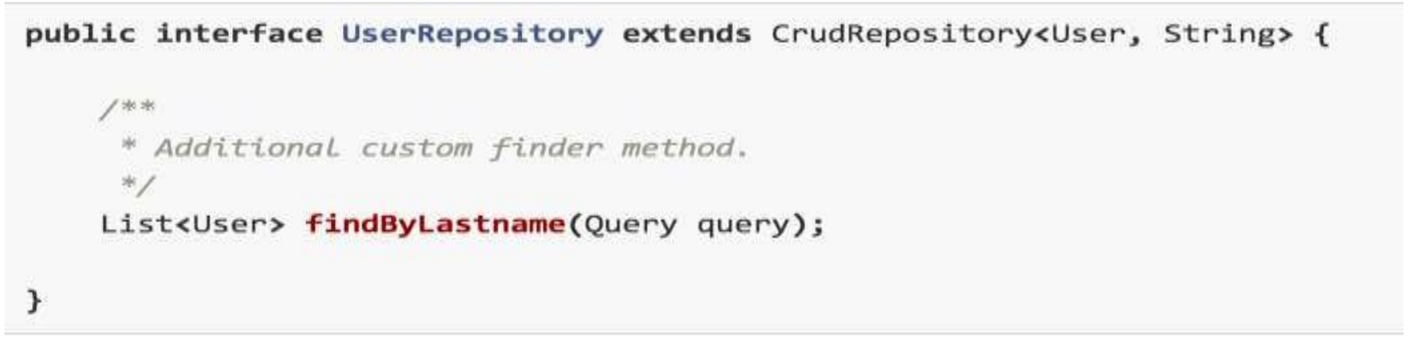
POJO centric model for interacting with Couchbase Buckets and easily writing a Repository style data access layer

- Support Java based @Configuration classes or an XML namespace
- Automatic implementation of Repository interfaces including support for custom finder methods
- Feature Rich Object Mapping integrated with Spring’s Conversion Service.

Dependency management

```
<dependencies>
    <dependency>
        <groupId>org.springframework.data</groupId>
        <artifactId>spring-data-couchbase</artifactId>
        <version>3.0.3.RELEASE</version>
    </dependency>
</dependencies><repositories>
    <repository>
        <id>spring-libs-release</id>
        <name>Spring Releases</name>
        <url>https://repo.spring.io/libs-release</url>
        <snapshots>
            <enabled>false</enabled>
        </snapshots>
    </repository>
</repositories>
```

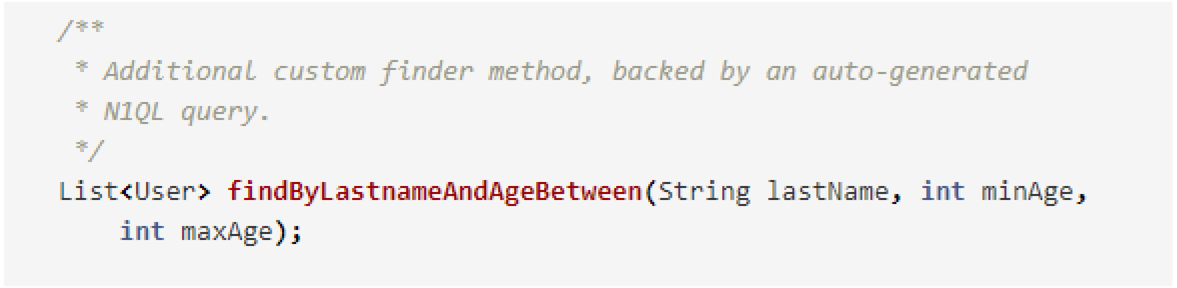
Repositories – Do this



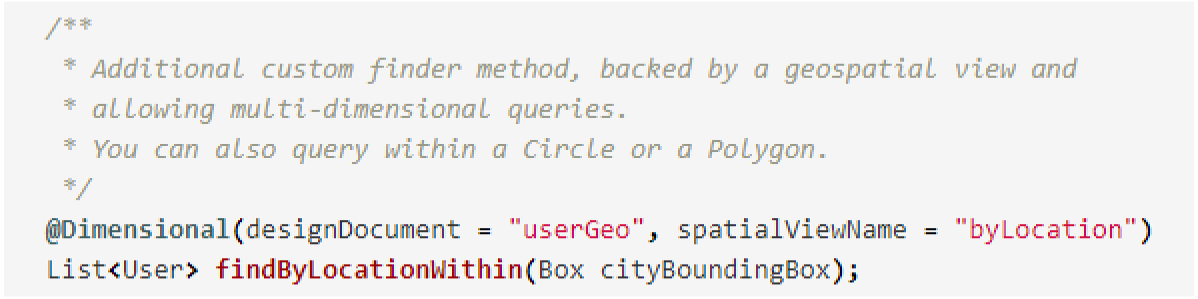
```
public interface UserRepository extends CrudRepository<User, String> {

    /**
     * Additional custom finder method.
     */
    List<User> findByLastname(Query query);

}
```

Couchbase

Repositories – Do this



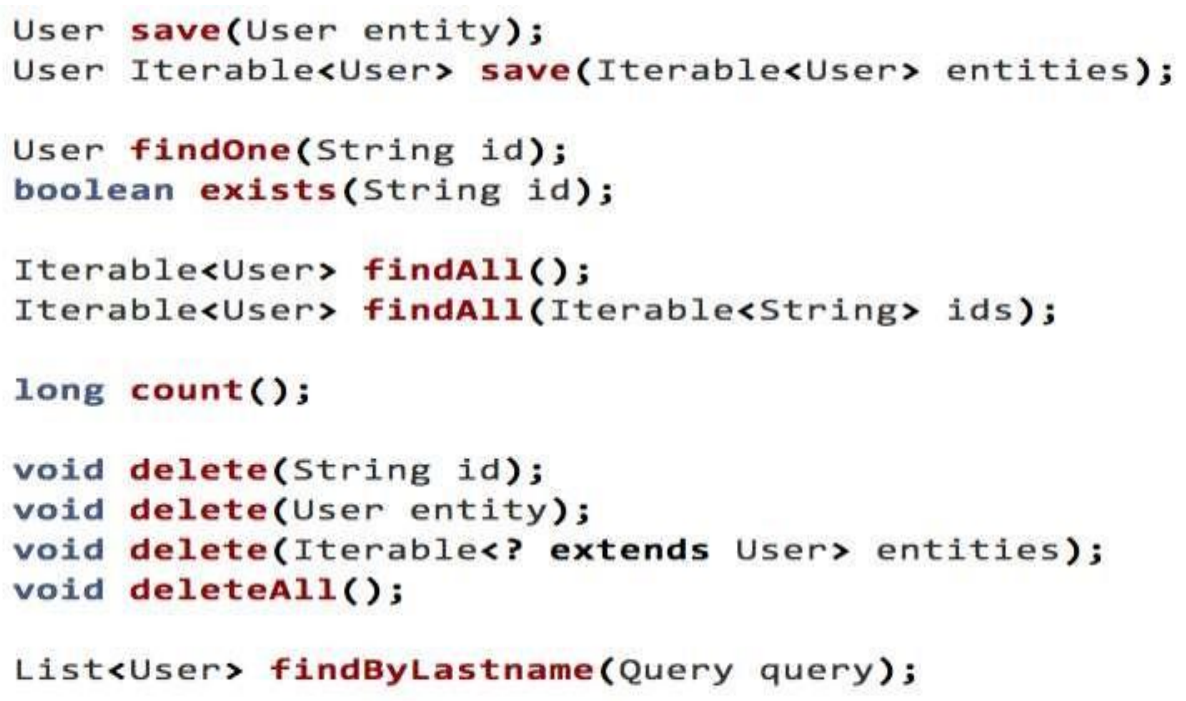
```java
/**
 * Additional custom finder method, backed by an auto-generated
 * N1QL query.
 */
List<User> findByLastnameAndAgeBetween(String lastName, int minAge,
    int maxAge);
```



```java
/**
 * Additional custom finder method, backed by a geospatial view and
 * allowing multi-dimensional queries.
 * You can also query within a Circle or a Polygon.
 */
@Dimensional(designDocument = "userGeo", spatialViewName = "byLocation")
List<User> findByLocationWithin(Box cityBoundingBox);
```

Couchbase

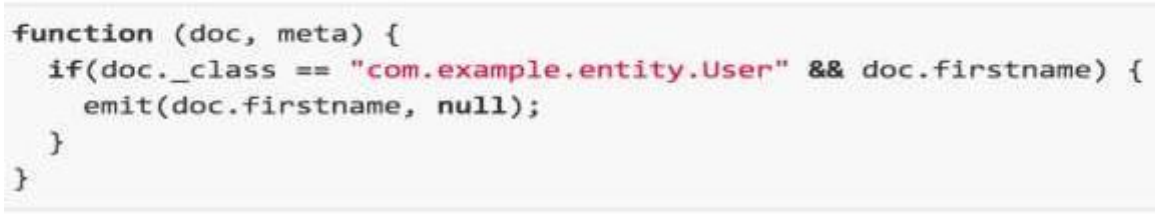
Repositories – Get this



```
User save(User entity);
User Iterable<User> save(Iterable<User> entities);

User findOne(String id);
boolean exists(String id);

Iterable<User> findAll();
Iterable<User> findAll(Iterable<String> ids);

long count();

void delete(String id);
void delete(User entity);
void delete(Iterable<? extends User> entities);
void deleteAll();

List<User> findByLastname(Query query);
```

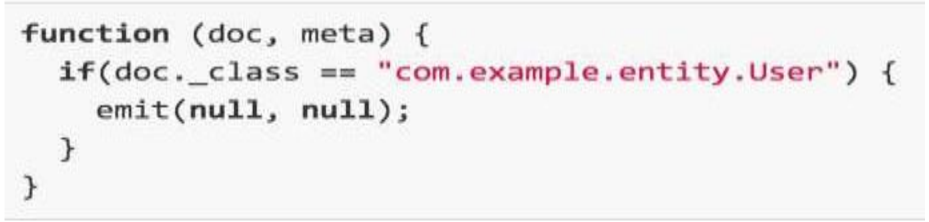
Repositories – Backed by Views

- findByFirstname()



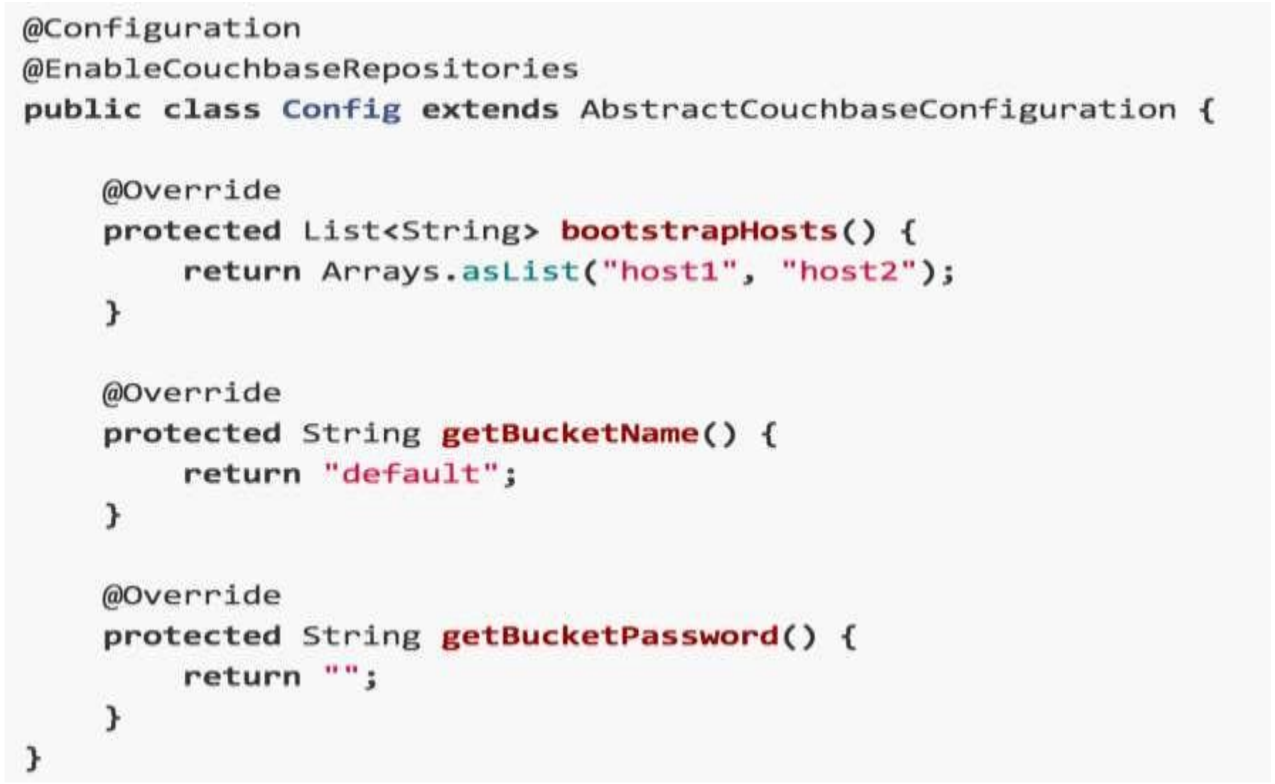
```
function (doc, meta) {
  if(doc._class == "com.example.entity.User" && doc.firstname) {
    emit(doc.firstname, null);
  }
}
```

- findAll(), count()



```
function (doc, meta) {
  if(doc._class == "com.example.entity.User") {
    emit(null, null);
  }
}
```

JavaConfig

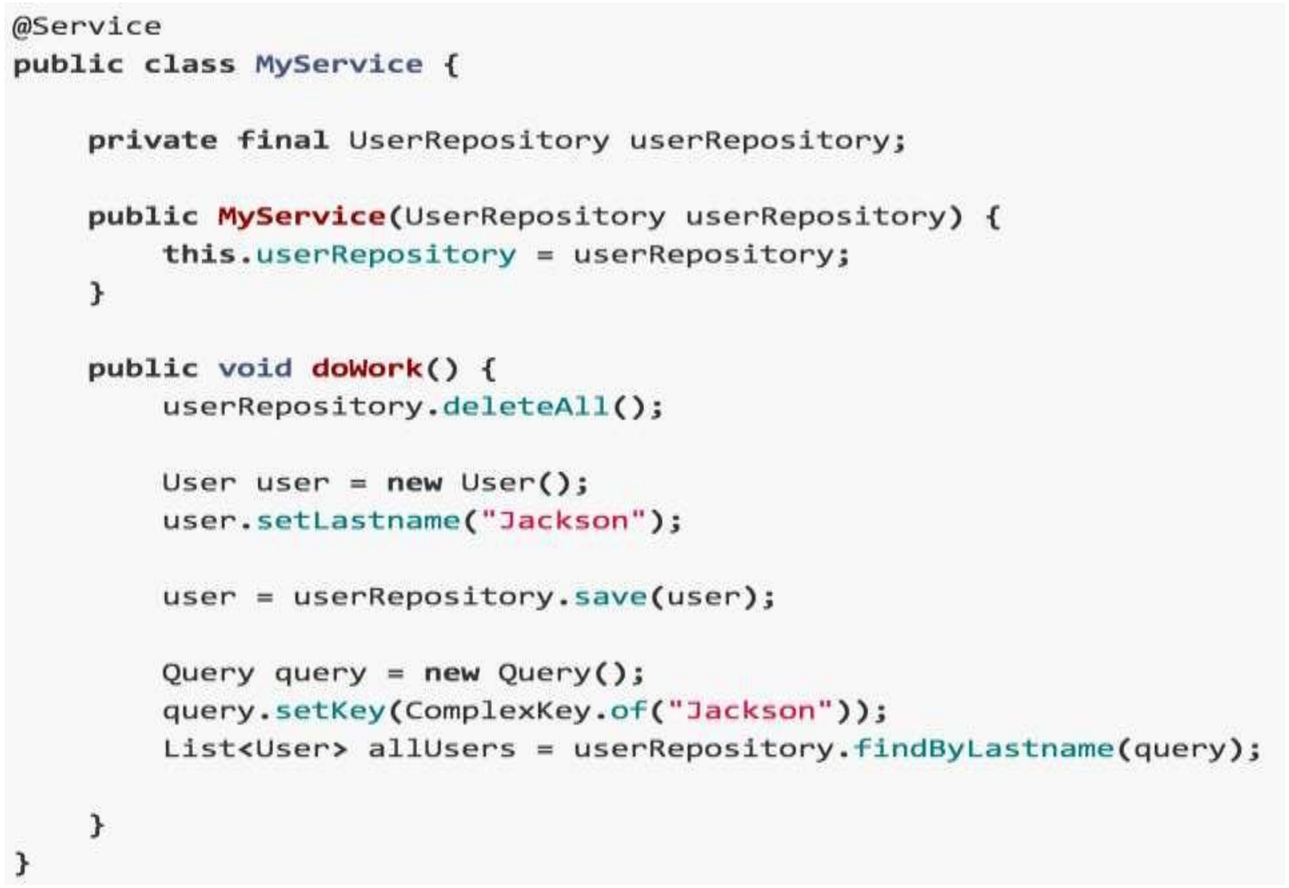


```
@Configuration
@EnableCouchbaseRepositories
public class Config extends AbstractCouchbaseConfiguration {

    @Override
    protected List<String> bootstrapHosts() {
        return Arrays.asList("host1", "host2");
    }

    @Override
    protected String getBucketName() {
        return "default";
    }

    @Override
    protected String getBucketPassword() {
        return "";
    }
}
```

XML configuration

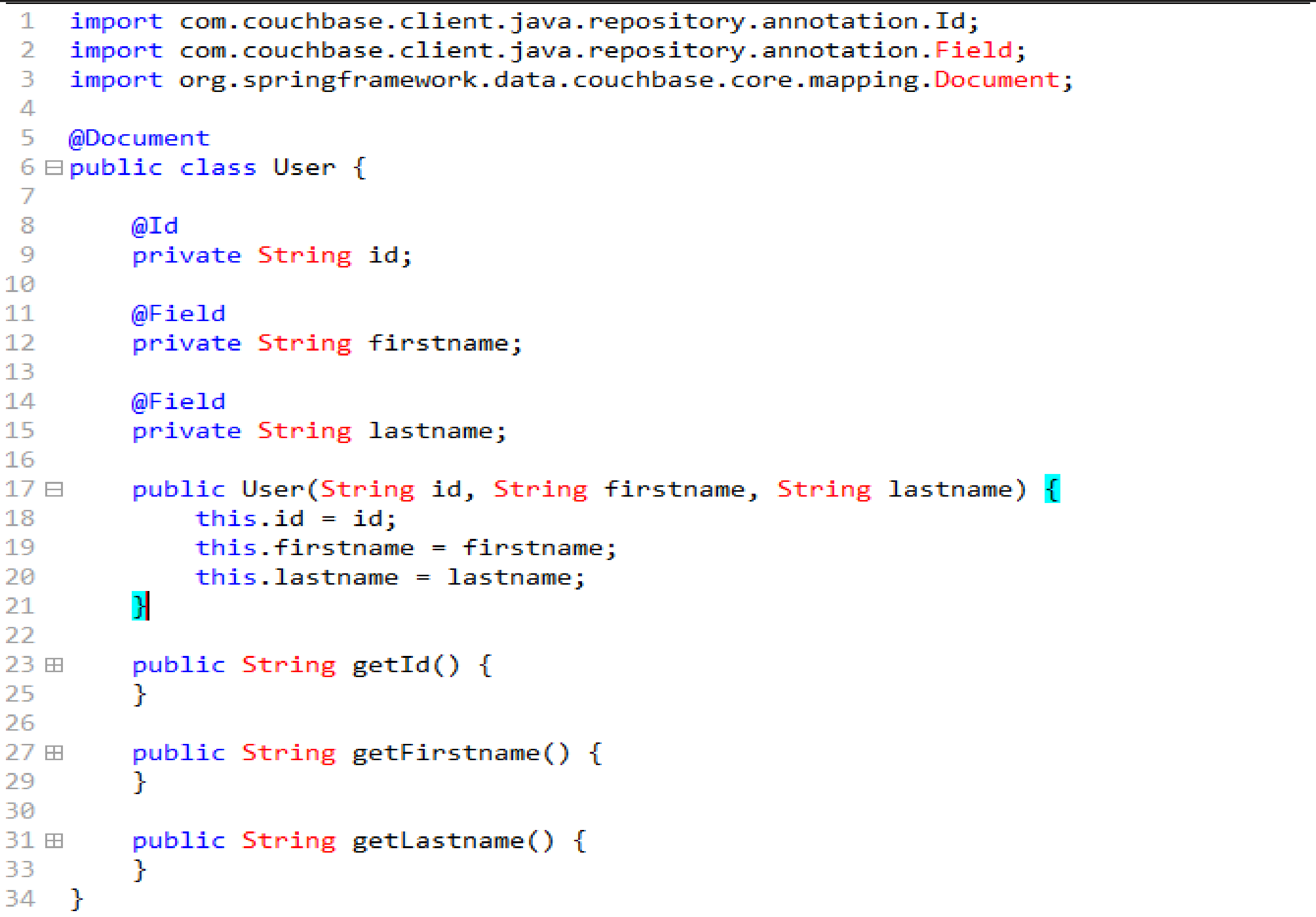
```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans:beans xmlns:beans="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://www.springframework.org/schema/data/couchbase
  xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans.xsd
    http://www.springframework.org/schema/data/couchbase
    http://www.springframework.org/schema/data/couchbase/spring-couchbase.xsd">

    <couchbase:cluster>
      <couchbase:node>127.0.0.1</couchbase:node>
    </couchbase:cluster>

    <!-- This is needed to probe the server for N1QL support -->
    <!-- Can be either cluster credentials or a bucket credentials -->
    <couchbase:clusterInfo login="beer-sample" password=""/>

    <couchbase:bucket bucketName="beer-sample" bucketPassword=""/>
</beans:beans>
```

Putting it together



```
@Service
public class MyService {

    private final UserRepository userRepository;

    public MyService(UserRepository userRepository) {
        this.userRepository = userRepository;
    }

    public void doWork() {
        userRepository.deleteAll();

        User user = new User();
        user.setLastname("Jackson");

        user = userRepository.save(user);

        Query query = new Query();
        query.setKey(ComplexKey.of("Jackson"));
        List<User> allUsers = userRepository.findByLastname(query);

    }
}
```

Document with Fields

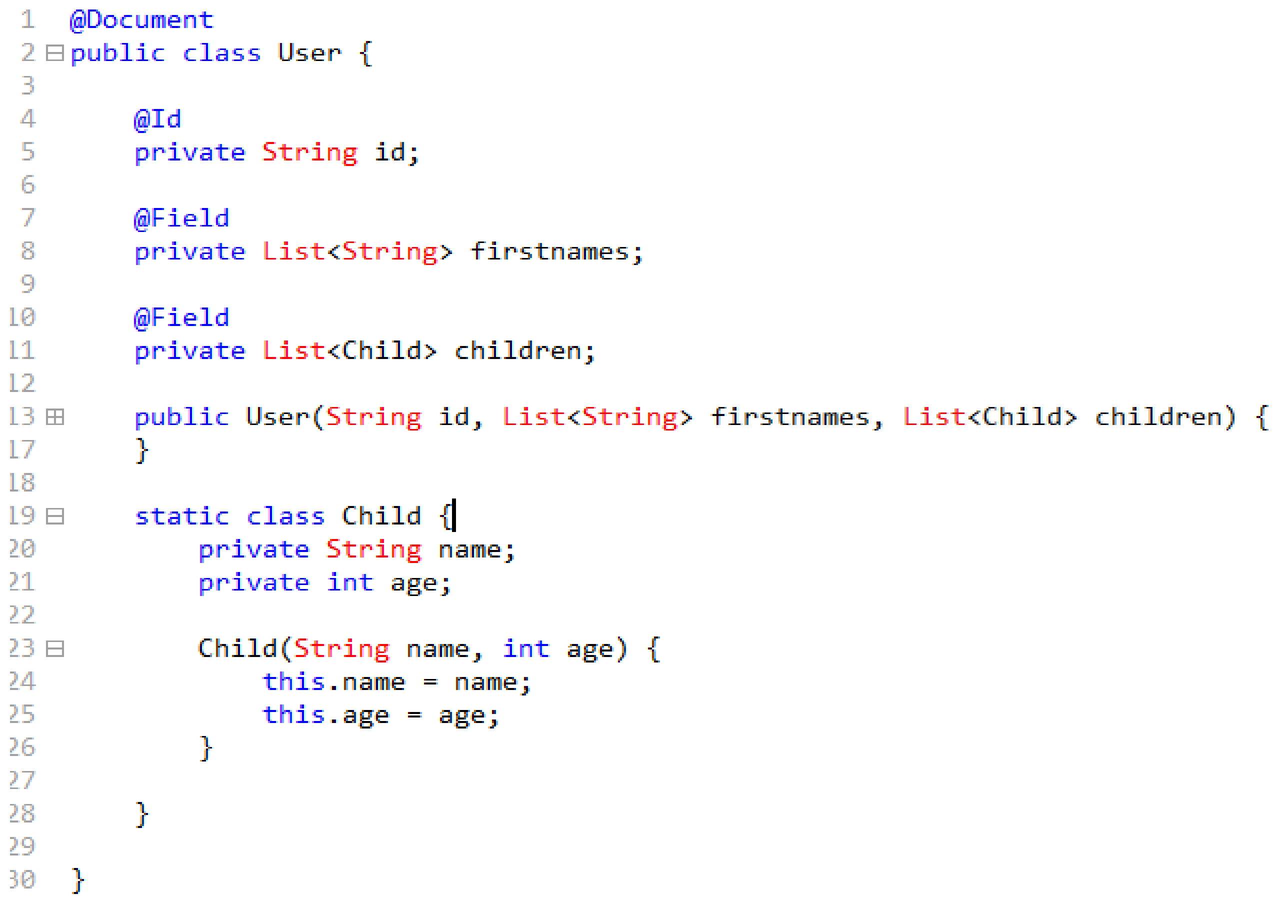


```java
import com.couchbase.client.java.repository.annotation.Id;
import com.couchbase.client.java.repository.annotation.Field;
import org.springframework.data.couchbase.core.mapping.Document;

@Document
public class User {

    @Id
    private String id;

    @Field
    private String firstname;

    @Field
    private String lastname;

    public User(String id, String firstname, String lastname) {
        this.id = id;
        this.firstname = firstname;
        this.lastname = lastname;
    }

    public String getId() {
    }

    public String getFirstname() {
    }

    public String getLastname() {
    }
}
```

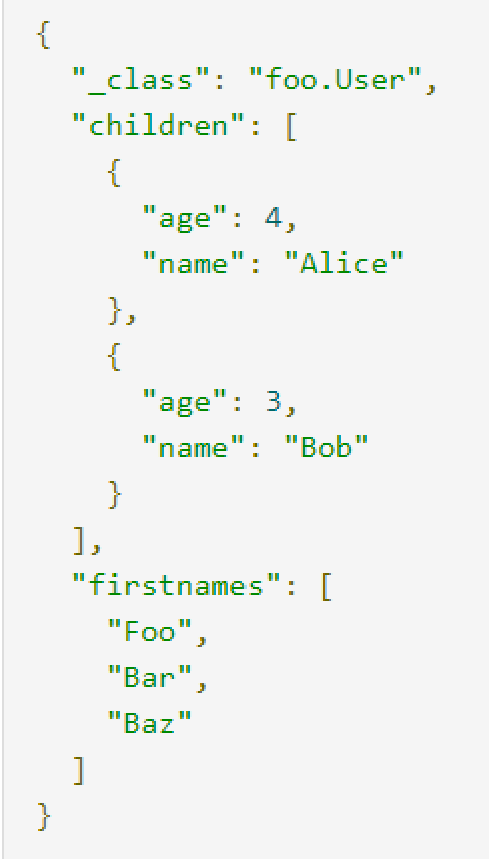
Document with Fields…

```
@Document(expiryExpression = "${valid.document.expiry}")
@Document(expiry = 10)
@Field("fname")
```

A Document with composed objects



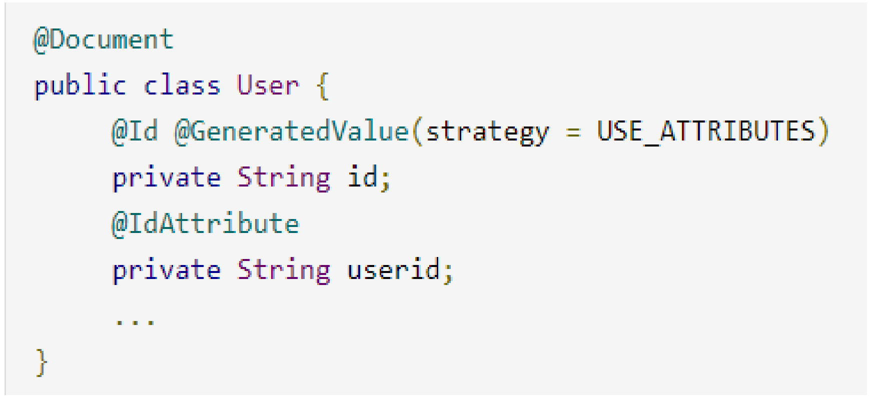
```java
@Document
public class User {

    @Id
    private String id;

    @Field
    private List<String> firstnames;

    @Field
    private List<Child> children;

    public User(String id, List<String> firstnames, List<Child> children) {
    }

    static class Child {
        private String name;
        private int age;

        Child(String name, int age) {
            this.name = name;
            this.age = age;
        }

    }

}
```



```json
{
  "_class": "foo.User",
  "children": [
    {
      "age": 4,
      "name": "Alice"
    },
    {
      "age": 3,
      "name": "Bob"
    }
  ],
  "firstnames": [
    "Foo",
    "Bar",
    "Baz"
  ]
}
```

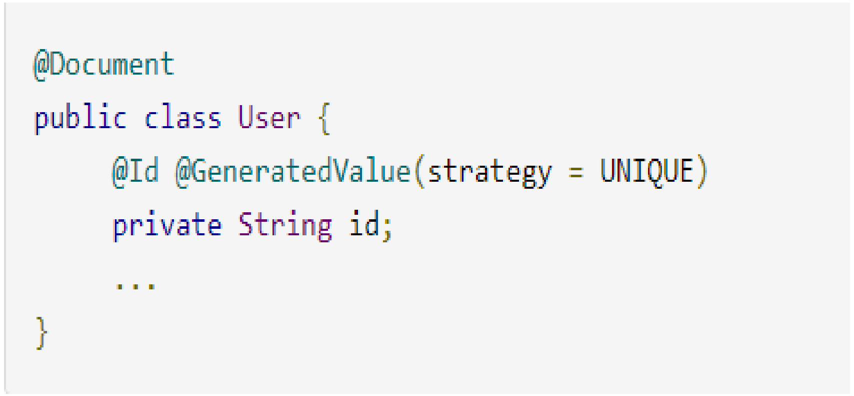
Auto generating keys

using attributes



```
@Document
public class User {
     @Id @GeneratedValue(strategy = USE_ATTRIBUTES)
     private String id;
     @IdAttribute
     private String userid;
     ...
}
```

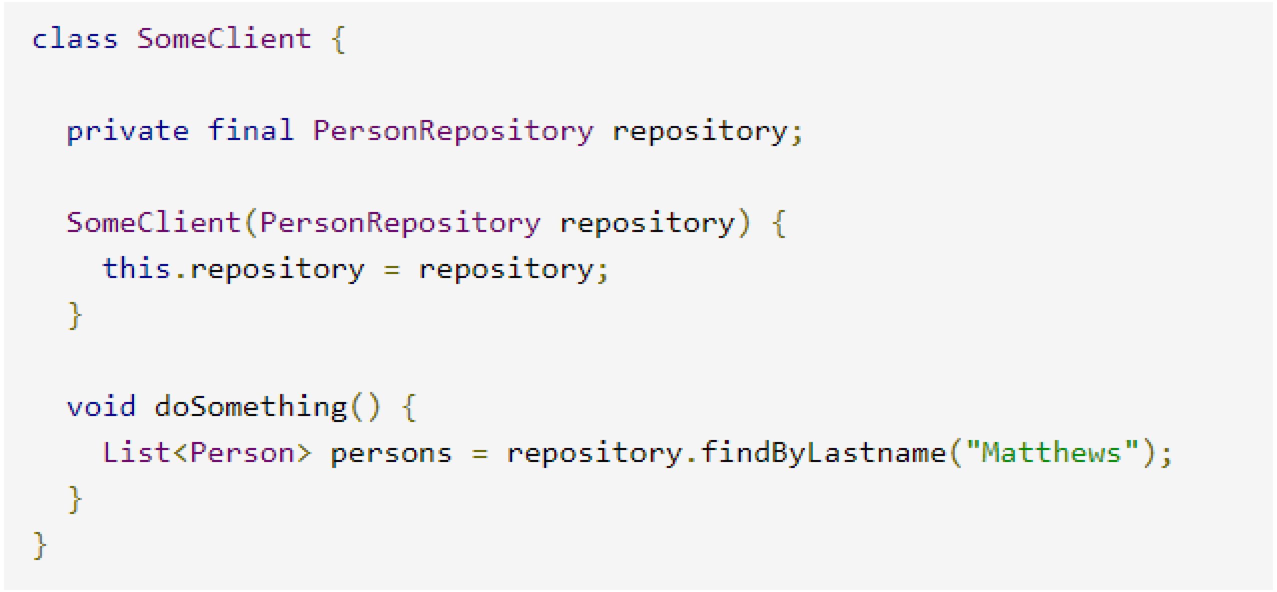
using uuid



```
@Document
public class User {
     @Id @GeneratedValue(strategy = UNIQUE)
     private String id;
     ...
}
```

Repositories

significantly reduce the amount of boilerplate code required to implement data access layers for various persistence stores.



```java
class SomeClient {

  private final PersonRepository repository;

  SomeClient(PersonRepository repository) {
    this.repository = repository;
  }

  void doSomething() {
    List<Person> persons = repository.findByLastname("Matthews");
  }
}
```

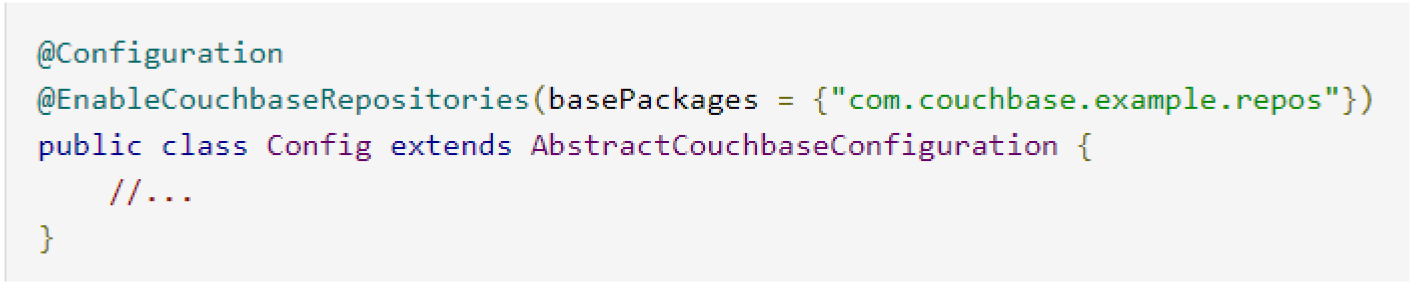
Couchbase repositories

three backing mechanisms in Couchbase for repositories

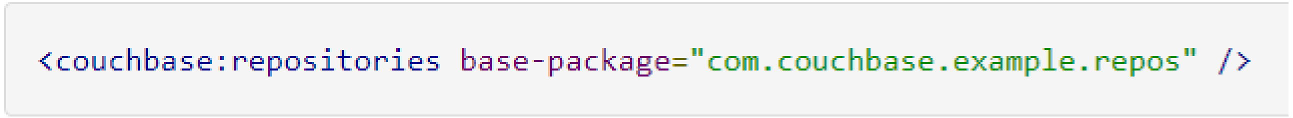
- N1QL based querying
- View based querying
- Spatial View based querying

- CRUD operations are still mostly backed by Couchbase views
- Such views (and, for N1QL, equivalent indexes) can be automatically built, but note this is **discouraged in production** and can be an **expensive operation**

Couchbase repositories

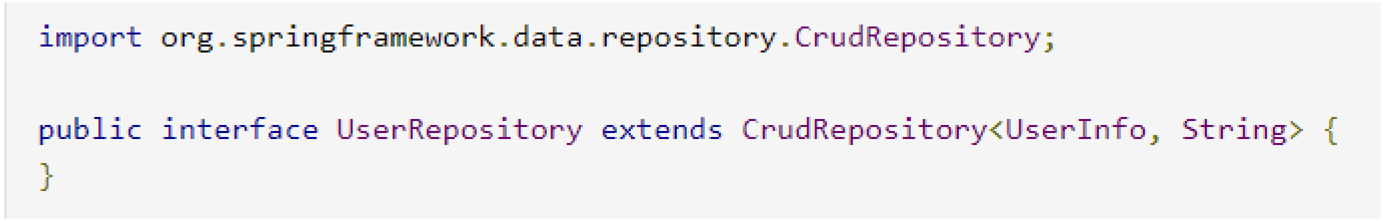


```
@Configuration
@EnableCouchbaseRepositories(basePackages = {"com.couchbase.example.repos"})
public class Config extends AbstractCouchbaseConfiguration {
    //...
}
```



```
<couchbase:repositories base-package="com.couchbase.example.repos" />
```

UserInfo repository - Example



```java
import org.springframework.data.repository.CrudRepository;

public interface UserRepository extends CrudRepository<UserInfo, String> {
}
```

- Just an interface and not an actual class.
- when context gets initialized, actual implementations for the repository descriptions get created and can access them through regular beans.

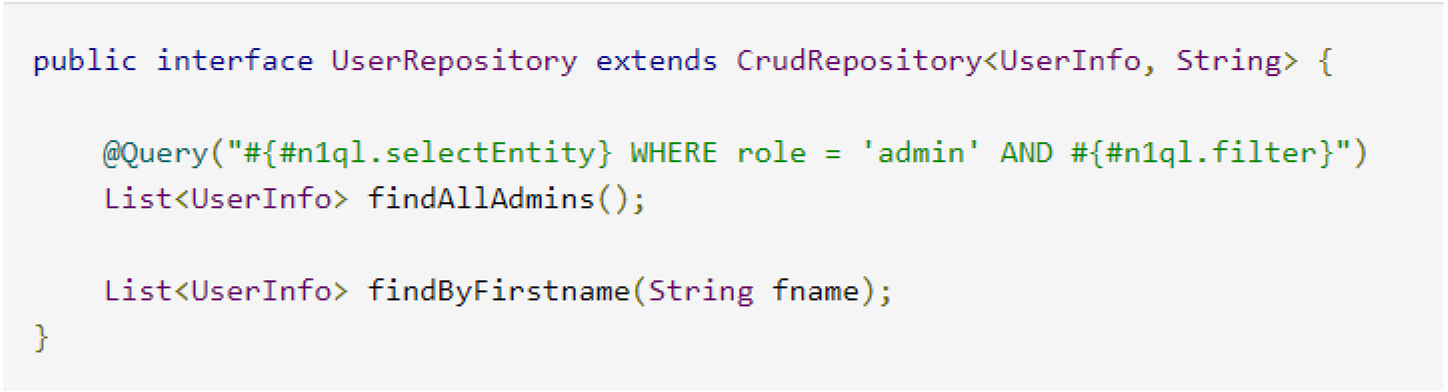
N1QL based querying

Spring-Data-Couchbase 2.0 - N1QL is the default way of doing queries and allow to fully derive queries from a method name

Prerequisite →

- have a N1QL-compatible cluster
- created a PRIMARY INDEX on the bucket

N1QL queries

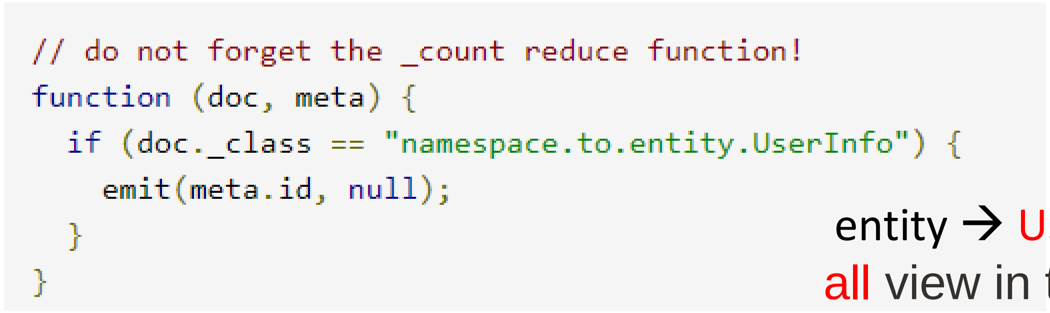


```
public interface UserRepository extends CrudRepository<UserInfo, String> {

    @Query("#{#n1ql.selectEntity} WHERE role = 'admin' AND #{#n1ql.filter}")
    List<UserInfo> findAllAdmins();

    List<UserInfo> findByFirstname(String fname);
}
```

Backing Views

All repository CRUD access methods which are not "by a specific key" still require a single backing view, by default all, to find the one or more matching entities

To cover the basic CRUD methods from the CrudRepository, one view needs to be implemented in Couchbase Server



```
// do not forget the _count reduce function!
function (doc, meta) {
  if (doc._class == "namespace.to.entity.UserInfo") {
    emit(meta.id, null);
  }
}
```

The all view map function

entity → UserInfo.
all view in the userInfo design document.

Lab : Spring – Data Couchbase