Chapitre 9
Bases de données

Plan

- NoSQL
- Quelques exemples
  - MongoDB
  - Firebase
  - CouchDB
  - RethinkDB
- SQL
  - NodeJS

Bases de données

Pourquoi ce chapitre à la fin du cours?

- La base de donnée demeure une composante essentielle de votre application… mais ne devrait pas en influencer **l'architecture**. Il s'agit d'un choix technologique comme les autres.

- Gardez en tête qu'il est très possible que votre application change de storage au fil de sa vie… d'où l'importance de ne pas **développer en fonction de votre BD**.

NoSQL

Les bases de données **NoSQL** existent maintenant depuis quelques années déjà et visent à s'éloigner du modèle relationnel traditionnel… le but étant d'obtenir une **vélocité** de développement.

- Nécessitent peu de connaissances spécifiques en BD (pas ou peu de DBA NoSQL)
- Coûtent traditionnellement moins cher
- Flexibilité du modèle de données

NoSQL

Le NoSQL vient avec certains **désavantages** qu'il est important de garder en tête -> savoir choisir le bon outil

- ***Eventual consistency*** - le concept de transaction n'existe pas, vous n'êtes pas garanti de relire les mêmes données que vous avez écrit à 100%

- **Duplication des données** - le concept de clés étrangères n'existe pas vraiment, le but est d'obtenir le plus de découplage possible... possible de répéter certaines informations

NoSQL

Relationnel : Divise l'information en diverses tables inter-reliées, se référençant par des clés étrangères.

Est basé sur des schémas rigides.

Langage : SQL

NoSQL : Entrepose des objets sous forme d'agrégats.
Une rangée peut être stockée dans 20 tables différentes.
N'utilise pas de schémas.

Langage : JSON

NoSQL

Relationnel vs NoSQL

Relational data model

Highly-structured table organization with rigidly-defined data formats and record structure.

Document data model

Collection of complex documents with arbitrary, nested data formats and varying "record" format.

Scaling vertical

Scaling horizontal

NoSQL

Quelques exemples de technologies populaires:

- MongoDB
- Firebase
- CouchDB
- Rethink DB

NoSQL

Pourquoi choisir une technologie plus qu'une autre?

- MongoDB est la technologie la plus **connue**, large écosystème construit autour
- Firebase est un service ***self-packaged***, facile à démarrer mais dispendieux à *scaler*
- CouchDB nécessite de définir des ***vues*** d'avance pour *querier*
- RethinkDB se présente comme une alternative nouveau genre pour Mongo.

MongoDB

MongoDB demeure la BD NoSQL la plus populaire.

- Open source, existe depuis 2009
- Utilisée par de gros clients
- Documents sauvegardés en **BSON** (Binary JSON), extension du JSON avec des types en plus.
- A parfois mauvaise réputation...

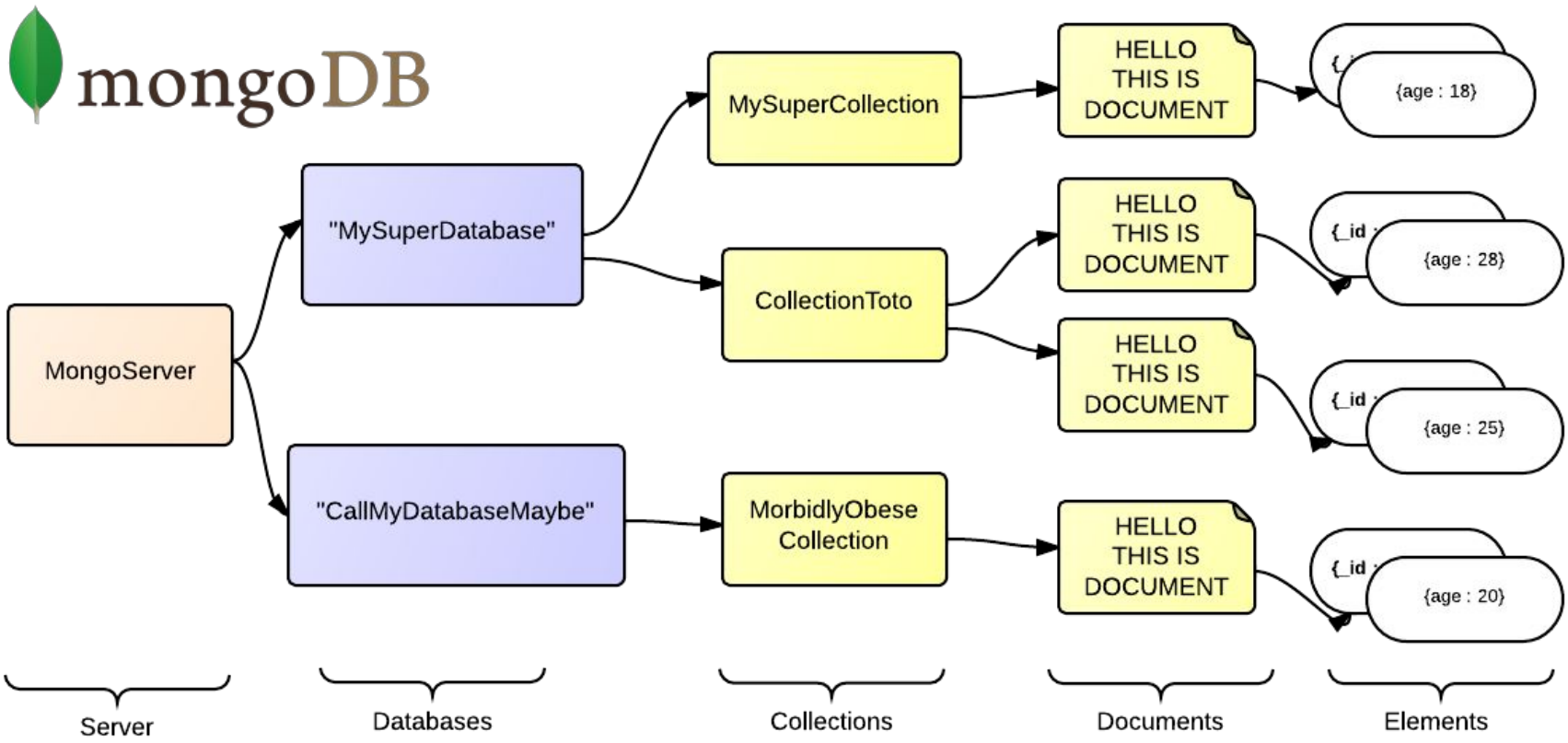
MongoDB

Quelques concepts importants :

- On ne parle plus de tables, mais bien de **collections**.
  Table vs collection? ➔ Pas de colonnes, chaque entrée repose sur un schéma dynamique, sous forme de clés-valeurs.

- Chaque entrée dans une collection est appelée un document.

MongoDB


mongoDB
MongoServer
"MySuperDatabase"
"CallMyDatabaseMaybe"
MySuperCollection
CollectionToto
MorbidlyObese Collection
HELLO THIS IS DOCUMENT
HELLO THIS IS DOCUMENT
HELLO THIS IS DOCUMENT
HELLO THIS IS DOCUMENT
{age : 18}
{age : 28}
{age : 25}
{age : 20}
Server
Databases
Collections
Documents
Elements

MongoDB

Document BSON :

```
{
  person: {
    first_name: "Peter",
    last_name: "Peterson",
    addresses: [
      {street: "123 Peter St"},
      {street: "504 Not Peter St"}
    ],
  }
}
```

À bien comprendre qu'une autre "person" dans la même collection pourrait avoir des membres complètement différents...

MongoDB

Quelques exemples :

(Note : les exemples seront en JS, afin d'être utilisés par NodeJS par exemple, mais ils pourraient très bien être en Java, C#, Ruby etc.)

Création de documents :

```
var age = {age: 25};
var languages = {name: 'Languages', languages: ['c', 'ruby', 'js']};
var student = {name: 'Jim', scores: [75, 99, 87.2]};
```

MongoDB

Opérations CRUD :

```
db.students.save(student);

db.students.remove( { company: "test" } )
```

Insère un document dans la collection students.
Supprime tous les éléments avec la compagnie ***test***.

Qu'arrive-t-il si la collection **students** n'existe pas?
⇒ Elle est créée automatiquement!

MongoDB

Requêtes : (voir : http://docs.mongodb.org/manual/tutorial/query-documents/)

Trouve tous les documents

```
db.inventory.find( {} )
```

Trouve le premier document

```
db.inventory.findOne()
```

Trouve tous les documents ayant la propriété ‘type’ égale à la valeur ‘snacks’

```
db.inventory.find( { type: "snacks" } )
```

```
db.inventory.find( { type: 'food', price: { $lt: 9.95 } } )
```

Condition AND
et opérateurs Mongo :
$lt = less than

MongoDB

Petite note…

Tous les objets insérés dans MongoDb ont un champ *_id* automatiquement populé par la base de données.

⇒ **Identifiant unique interne**

```
{
"_id": ObjectId("5210a64f846cb004b5000001"),
"permalink": "ca8W7mc0ZUx43bxTuSGN",
"data": "a lot of stuff",
"datetime": ISODate("2013-08-18T11:47:43.460+-100")
}
```

Voir :
http://mongodb.github.io/node-mongodb-native/api-bson-generated/objectid.html

ORM (Object-Relational Mapping)

Vous allez souvent voir le terme **ORM** lorsque vous choisirez votre base de données.

→ Un ORM permet d'abstraire typiquement les queries natives vers le langage de programmation choisi.

Arme à double tranchant

- Permet de se protéger facilement contre les injections SQL…
- … Mais très important de comprendre ce qui est fait en dessous. Possibilité de générer des queries monstrueuses.

MongoDB

MongoDB avec NodeJS : MONGOOSE
http://mongoosejs.com/

Mongoose est un package NodeJS permettant d'offrir une vue plus *traditionnelle* sur Mongo.

- Solution basée sur des modèles… rend le tout beaucoup plus orienté objet.
- Permet de structurer les données et de les utiliser comme **modules**.
- Souvent ORM de choix pour Mongo en NodeJS.

MongoDB

Un petit exemple bien simple…

```
var mongoose = require('mongoose');
mongoose.connect('mongodb://localhost/test');

// Définition d'un modèle Cat.
var Cat = mongoose.model('Cat', { name: String });

// Instanciation du modèle en question.
var kitty = new Cat({ name: });

kitty.save(function (err) {
    if (err) // ...
        console.log('meow');
});
```

MongoDB

Exemple un peu plus complexe…

Définissions un fichier **user.js** qui représente notre modèle d'usager (typiquement dans un dossier models)

```
var mongoose = require('mongoose');
var Schema = mongoose.Schema;

// Définition d'un schéma d'usager… ils devront respecter ce canvas.
var userSchema = new Schema({
    email: String,
    name: String
});

// Définition du modèle en utilisant le schéma précédemment défini.
var User = mongoose.model('User', userSchema);

exports.schema = userSchema;
exports.model = User;
```

MongoDB

Exemple un peu plus complexe…

Comment se servir du modèle **User** ensuite?

```
var User = require('../models/user.js').model;

// La méthode findById est disponible sur tous les modèles
exports.findById = function (req, res) {
    User.findById(req.params.id, function (err, user) {
        if (!err) {
            if (user) {
                res.status(200).send(user);
            }
              …
        });
};
```

Firebase

https://firebase.google.com/docs/database/

Particularités de Firebase:

- Entièrement hébergée dans le Cloud (**GCP**)
- Bases de données en *temps réel* - tous les clients connectés peuvent s'enregistrer sur des mises à jour du data
- Offre la même flexibilité JSON que MongoDB

Firebase

Firebase

Voir https://firebase.google.com/docs/database/web/structure-data ...

Ici, la façon dont vous structurez vos données influence directement la **performance** de votre application.

⇒ Dénormalization de votre data

Firebase

Firebase

Firebase se base beaucoup sur l'architecture **REST** que nous avons vu précédemment… permet d'offrir une certaine interface commune pour tout type de client (iOS, Android, Web etc.)

```
function writeUserData(userId, name, email, imageUrl) {
  firebase.database().ref('users/' + userId).set({
    username: name,
    email: email,
    profile_picture : imageUrl
  });
}
```

CouchDB

Designée par **Apache**, CouchDB est un peu différente…
http://docs.couchdb.org/en/2.1.0/

- Encore une fois, tout est stocké en JSON
- Communique avec la BD exclusivement en HTTP
- Pas de *query language*… fonctionne exclusivement avec des vues prédéfinies et du MapReduce.

CouchDB

Voir https://wiki.apache.org/couchdb/HTTP_view_API en détail…

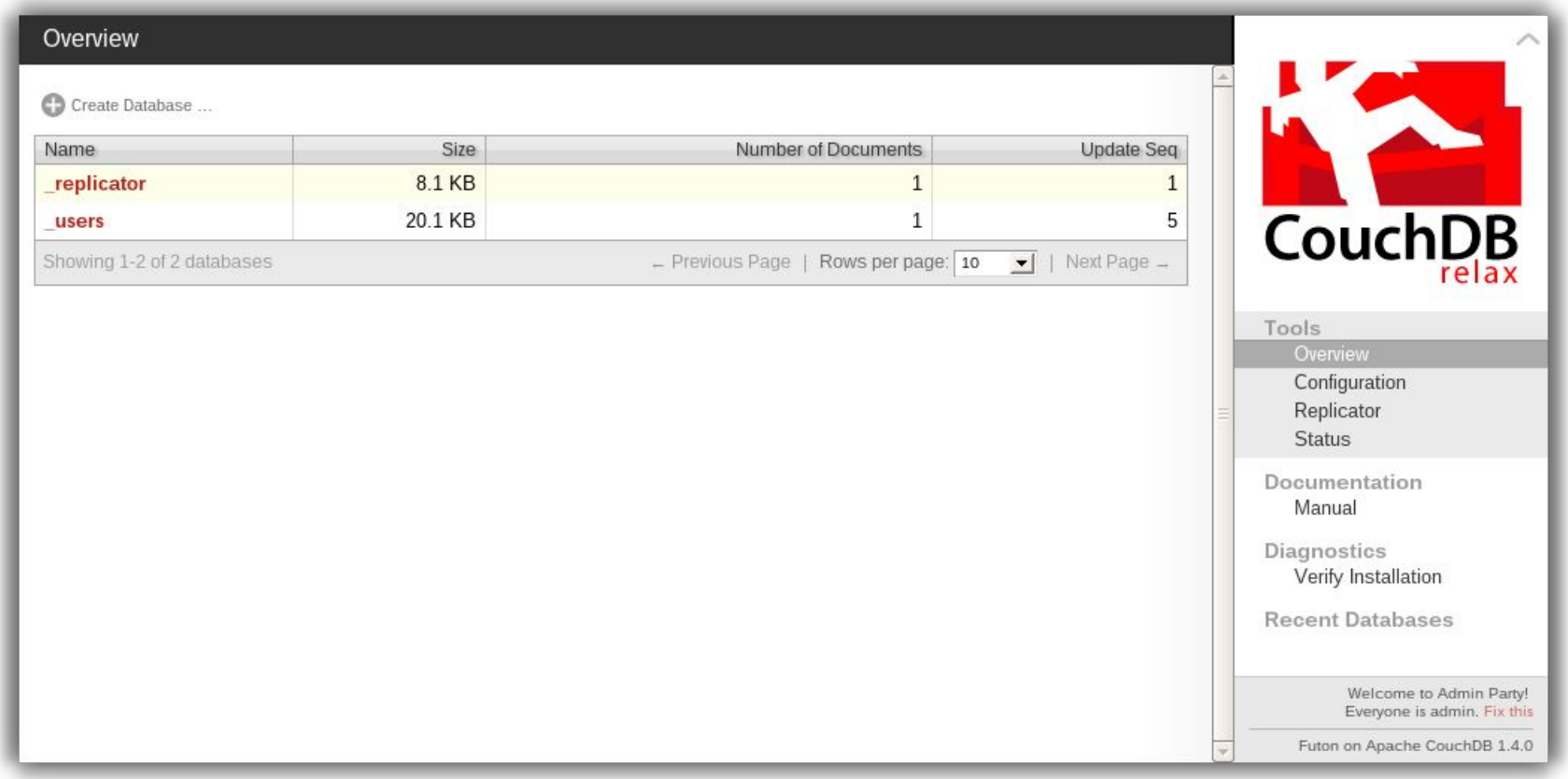
- Les *views* sont sauvegardées dans ce qu'on appelle des *design documents* - document spéciaux utilisés exclusivement pour la définition de ces vues.

- Facile à définir, mais ne supporte donc pas les queries dynamiques… un peu comme un schéma de requête?

- Tout peut se faire en HTTP

CouchDB

- Gros avantage - l'interface *built-in (Futon)*



RethinkDB

RethinkDB ressemble un peu à un mélange entre Firebase et MongoDB.

Excellente explication ici: https://www.rethinkdb.com/faq/

- Base de données temps réel telle que Firebase
- Données stockées en JSON
- Utilise ReQL comme langage de *querying*

RethinkDB

- Supporte de multitude de types de données nativement, notamment tout ce qui est **géospatial**

  https://www.rethinkdb.com/docs/data-types/

- Possède une interface native telle que Couch ou Firebase

  https://www.rethinkdb.com/docs/administration-tools/

- Une question de préférence personnelle à ce point...

SQL

Et le **SQL** dans tout ça?

Le SQL reste une option viable, surtout lorsque les données sont **prévisibles** et dans un format **prédéterminé**. L'efficacité des bases de données SQL est prouvée depuis belle lurette.

Certaines bases de données traditionnelles ont moins leur place dans un monde web/cloud (Oracle, SQL Server notamment…).

SQL

Alternatives :

- PostgreSQL : http://www.postgresql.org/
  ⇒ Open-source et activement supportée. Offre également les champs de type JSON...

- MySQL : http://www.mysql.fr/
  ⇒ Un classique du web. Offre aussi le JSON maintenant.

- Amazon Aurora : https://aws.amazon.com/rds/aurora
  ⇒ Base de données extrêmement performante.

SQL + NodeJS

De part sa nature, NodeJS est plus naturellement adapté aux bases de données NoSQL (JSON natif).

→ Par contre, certaines excellentes librairies existent et permettent de rendre le SQL tout aussi viable.

http://bookshelfjs.org/

- ORM SQL pour NodeJS, ressemble à Mongoose.
- Basé sur http://knexjs.org/