RethinkDB et son langage d'interrogation

Projet du cours NFE204

Préparation du certificat de spécialisation Analyste de données massives

Rodolphe CHAZELLE

Année 2015-2016

Table des matières

Introduction

Dans le cadre du projet NFE204, j'ai décidé d'étudier le langage de requête du système No SQL RethinkDB. Développé à partir de 2009, RethinkDB est un système de gestion de base de données distribuées orienté documents sous licence libre. Ce système temps réel offre de nombreux avantages. En plus de la possibilité de visualiser en temps réel les modifications réalisées sur la base de données, RethinkDB dispose de son propre langage de requête : ReQL.

ReQL permet de manipuler avec aisance les données grâce à un système chainable d'instructions. Simple à utiliser, celui-ci il s'intègre facilement dans un environnement de programmation grâce à son package. Chose rare en système distribué, il permet de réaliser des jointures entre des tables.

Ce projet sera composé de trois parties. La première sera consacrée à la présentation du système RethinkDB. Nous verrons son histoire et décrirons ses principales caractéristiques (scalabilité, cohérence, partitionnement, reprise sur pannes etc.). La seconde partie introduira les données des événements en France, que nous utiliserons pour illustrer les principes énoncés. Nous présenterons les principales manipulations dans RethinkDB permettant d'importer les données et de configurer le système. Enfin, dans la dernière partie, nous aborderons les possibilités offertes par le langage de requête ReQL (filtre, agrégation, manipulations de bases, jointure etc.). Nous étudierons sa syntaxe en réalisant des requêtes sur la base de données constituée.

Logo de la dernière version de RethinkDB

I. Présentation et caractéristiques du systéme RethinkDB

a) Présentation

RethinkDB est un système de gestion de bases de données distribuées orienté documents qui permet de stocker des documents JSON. Il a été développé à partir de 2009 par une société du même nom. L'entreprise est domiciliée au 156 E.Dana St. Mountain View, CA 94041 aux Etats-Unis.

Distribué librement, le système et sa documentation sont disponibles à l'adresse suivante :

https://www.rethinkdb.com

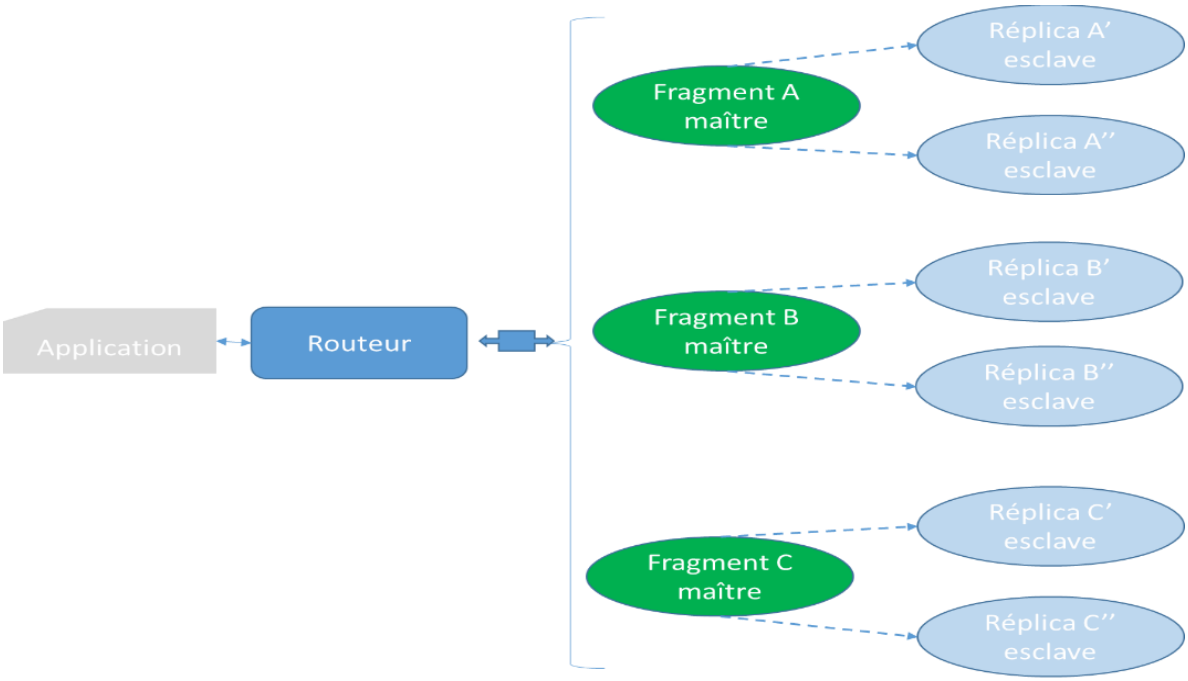
L'installation se réalise facilement et la documentation proposée est claire.

RethinkDB est un système dont la popularité ne cesse de croitre. Classé 67éme en septembre 2015 au classement DB-Engines, il est à la 46éme place à fin février 2016.

b) Mode de fonctionnement du système

RethinkDB fonctionne selon le schéma maître esclave. On choisit pour chaque table le nombre de fragments à mettre en place pour le stockage ainsi que le nombre de réplications de ces fragments.

Dans l'exemple d'une table partitionnée en trois fragments avec trois réplications par fragment. Une représentation simplifiée serait la suivante :



Tous les fragments de réplications sont associés à un réplica maître. Toutes les requêtes en lecture et écriture passent par ce nœud maître, où elles sont priorisées et évaluées.

c) Performance/Cohérence

Le rapport performance/cohérence du système est géré par les trois paramètres suivants :

En écriture

- Le paramètre write_ack permet de choisir entre une écriture synchrone ou asynchrone. Il peut prendre la valeur :

  «majority» : L'écriture est confirmée lorsque la majorité des réplicas a confirmé l'écriture. Les données sont majoritairement cohérentes.

  «single» : L'écriture est confirmée quand un seul réplica a confirmé l'écriture. Plus rapide mais risque pour la cohérence des données en cas de plantage.

En méthode de stockage

- Le paramètre durability permet de choisir le type de sauvegarde. Il possède les attributs suivants :

«hard» : les écritures sont réalisées sur le disque dur avant que la confirmation ne soit renvoyée.

«soft» : les écritures sont réalisées en RAM et par la suite sur le disque dur. Plus rapide mais en cas de panne les données n'ont pas été écrites dans la base.

En lecture

- Le paramètre read_mode peut prendre les valeurs suivantes :

  «single» : retourne la valeur dans la mémoire (pas nécessairement le disque dur) du réplica maître.

  «majority» : retourne la valeur majoritaire stockée sur les disques durs des réplicas. C'est la solution la plus longue mais qui garantit la meilleure cohérence.

  «outdated» : renvoie les valeurs qui sont en mémoire d'une sélection arbitraire de réplicas. C'est la solution la plus rapide mais qui garantit le moins la cohérence des données.

Configuration de base : Majority, hard, single. RethinkDB préfère la prudence à la performance, excepté pour l'écriture (single à la place de majority), car le mode «majority» nécessite d'envoyer des requêtes à tous les réplicas et d'attendre une majorité de retours, ce qui a pour conséquence de dégrader significativement la performance.

d) Partitionnement

RethinkDB effectue un partitionnement par intervalles. Il utilise comme clé de partitionnement la clé primaire indiquée lors de l'import des données. Le nombre de fragments et de réplications est paramétrable pour chaque table.

e) Disponibilité du système : reprise sur panne

Tant qu'une table possède plus de la moitié des réplicas de ses fragments, alors sa disponibilité est assurée. Si un des fragments ne possède plus la moitié de ses réplicas, alors les opérations de lecture et d'écriture échoueront.

Si un réplica maître n'est plus disponible mais qu'une majorité des réplicas est disponible, alors ils élisent un nouveau réplica maître.

II. Présentation des données et import dans RethinkDB

a) Source des données

Les données sont issues du site Infolocale (http://datainfolocale.opendatasoft.com/explore/?sort=modified)

Infolocale propose divers services :

- Annonce et diffusion d'événements dans les médias (en une seule saisie, l'événement créé est publié dans les journaux et sur les sites partenaires).
- Faire connaitre son organisme (diffuser sa carte de visite dans l'annuaire local des associations de la commune).

C'est à ce premier service que nous allons nous intéresser dans le cadre du projet. Infolocale accepte de mettre à disposition ses données des événements au format JSON. Pour cela, il suffit de s'inscrire (gratuitement) et de présenter son projet par écrit. L'accés à la base de données m'a été permis suite à la présentation de ce projet.

b) Description du format des données

Le fichier est au format JSON et fait 1.2go. Les principaux champs de description d'un événement sont les suivants :

- Datasetid : constante « infolocale_evenements »
- Record_id : Clé primaire du document
- geometry
    - Coordinates : [longitude, latitude]
- Fields :
    - Categorie : catégorie de l'évènement
    - Code_insee : code INSEE de l'événement
    - Code postal : code postal de la ville
    - Commune : nom de la commune
    - Coordonnées GPS : latitude et longitude
    - Date modification : date de modification de l'annonce
    - Departement : numéro du département
    - Etat : «V»
    - Genre : genre de l'événement
    - Genre_ident : genre de l'événement en numérique
    - Jour_1 : date de début de l'événement
    - Jour_max : dernier jour de l'événement
    - Jour_min : premier jour de l'événement
    - Organisme_nom : Nom de l'organisme organisateur
    - Tarif_mention_payant : Oui ou non
    - Tarif mention_gratuit : oui ou non

- Texte_début : Haut du texte
- Texte_milieu : Milieu du texte
- Theme : thème de l'événement
- Tite : titre de l'événement

Un exemple d'événement est présenté en **Annexe 1**

c) Import des données dans RethinkDB

RethinkDB possède une commande d'import des données. Sa syntaxe est la suivante :

```
rethinkdb import -f FILE --table DB.TABLE [-c HOST:PORT] [-a AUTH_KEY]
  [--force] [--clients NUM] [--format (csv | json)] [--pkey PRIMARY_KEY]
  [--delimiter CHARACTER] [--custom-header FIELD,FIELD... [--no-header]]
```

Dans un premier temps il faut lancer rethindb à partir d'une console

```
rethinkdb
```

En parallèle on lance la commande d'import du fichier des événements à partir d'une seconde console. On demande à rethinkDB de les stocker dans une table « evenements » de la base de données « projet ». On lui demande également de prendre comme clé primaire le champ recordid.

```
rethinkdb import -f infolocale_enevements.json --table projet.evenements --
pkey recordid
```

On observe une barre d'avancement qui progresse jusqu'à 100%

```
chazelle@Rodolphe:~$ rethinkdb import -f /home/chazelle/Bureau/infolocale_evenements.json --table projet.evenements --pkey recordid
[==========================================] 100%
```

On peut accéder à la console d'administration du système en se connectant à l'aide du navigateur à localhost :8080 . 5 onglets sont à notre disposition pour piloter le système distribue.

- **Dashboard** : présente les informations générales du système (nombre de serveurs, de tables, d'index, les performances du système)

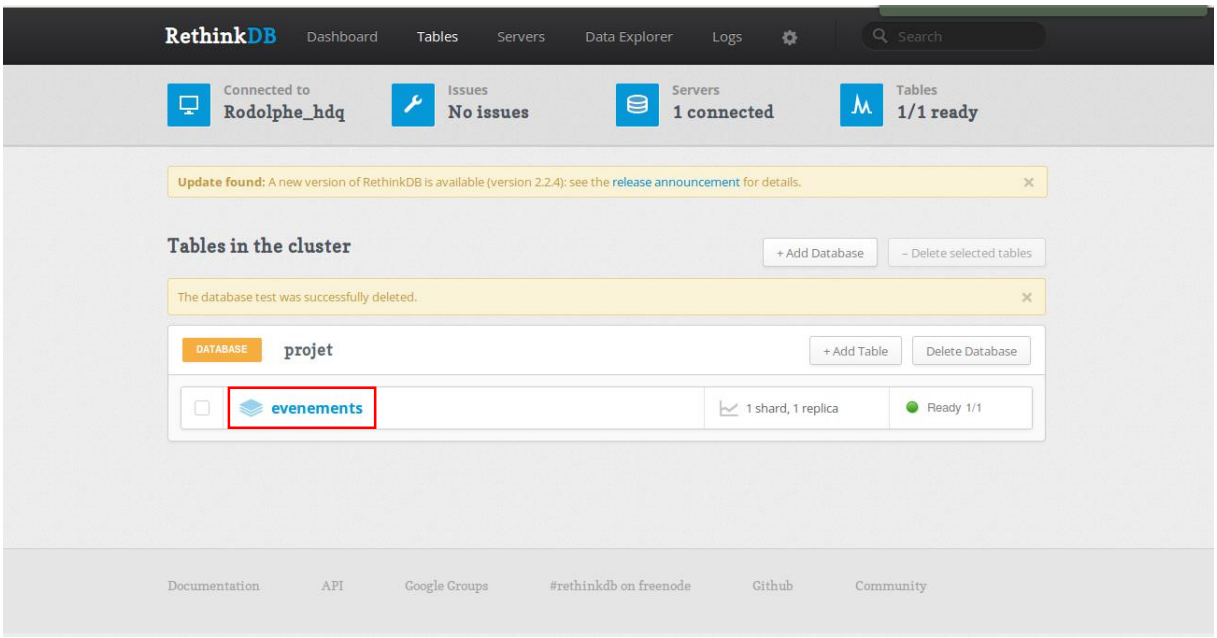
- **Tables** : Décrit les tables présentes dans chaque database.

- **Servers** : liste les serveurs connectés au système et donne leurs caractéristiques et les dernieres modifications de log.

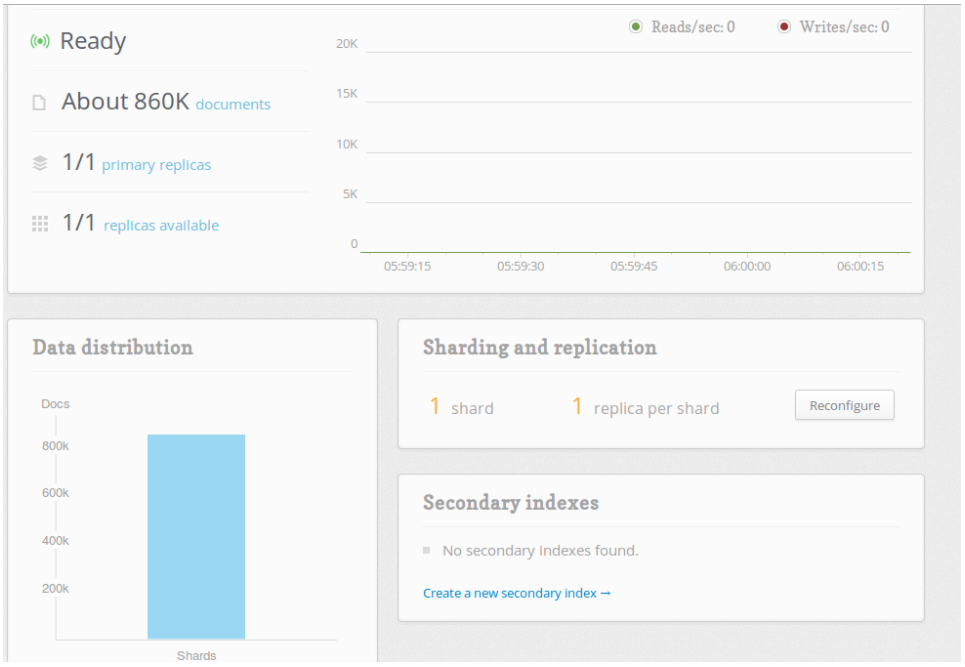
- **Data Exporer** : fenêtre permettant de lancer des requêtes en ReQL

- L**ogs** : accès aux logs

Suite à notre import, on observe la création de la table évènements dans la base de données projet (onglet Tables).



En cliquant sur la table événements, on observe qu'elle possède environs 860k documents et n'est pas répliquée pour l'instant.



d) Paramétrage du système

Afin d'assurer la disponibilité des données et la reprise sur panne, nous allons partitionner notre base en trois fragments et les répliquer trois fois chacun. Pour se faire, nous devons donc créer 2 autres serveurs virtuels sur lesquels les réplicas seront stockés. La manipulation dans rethinkdb est la suivante :

Dans une deuxième console, on lance la commande :

```
rethinkdb --port-offset 1 --directory rethinkdb_data2 --join
localhost:29015
```

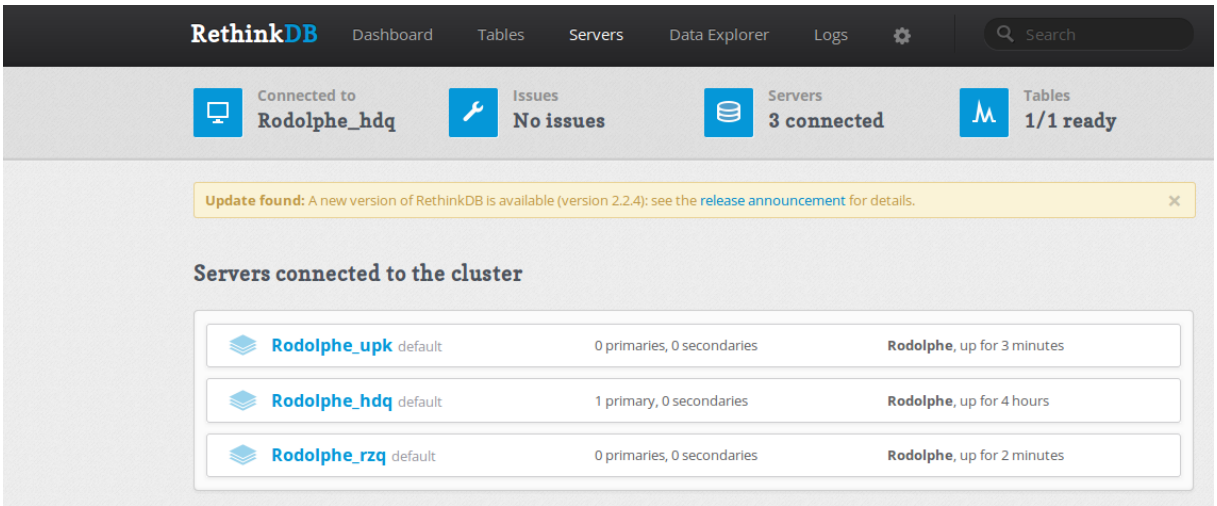
Dans une troisième console :

```
rethinkdb --port-offset 2 --directory rethinkdb_data3 --join
localhost:29015
```

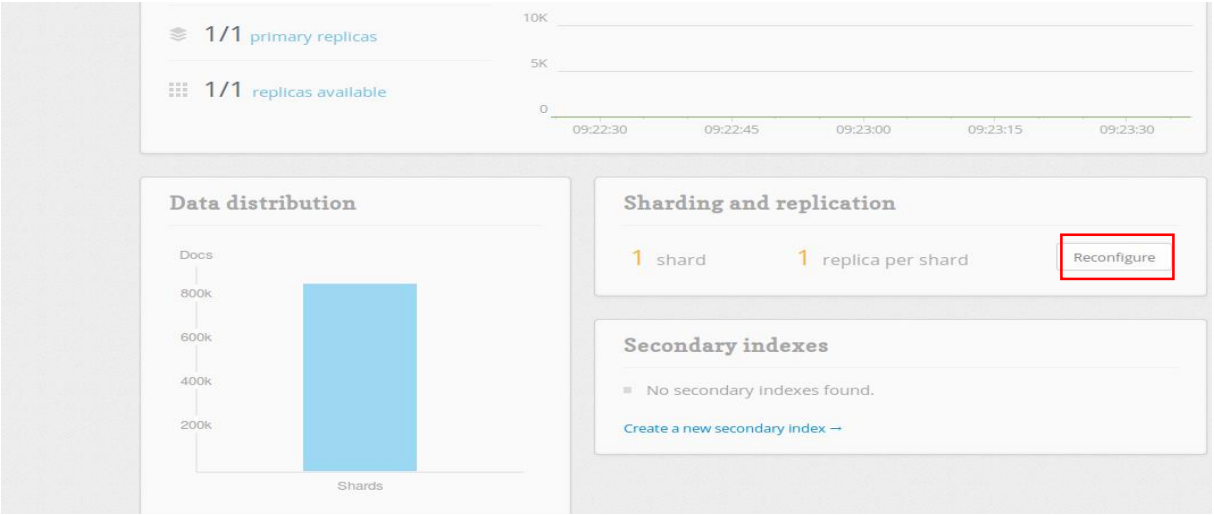
port-offset x *incrémente tous les ports du serveur de x pour éviter que plusieurs nœuds utilisent les mêmes.*

Join localhost :29015 *: on dit à cette instance RethinkDB de se connecter aux autres.*

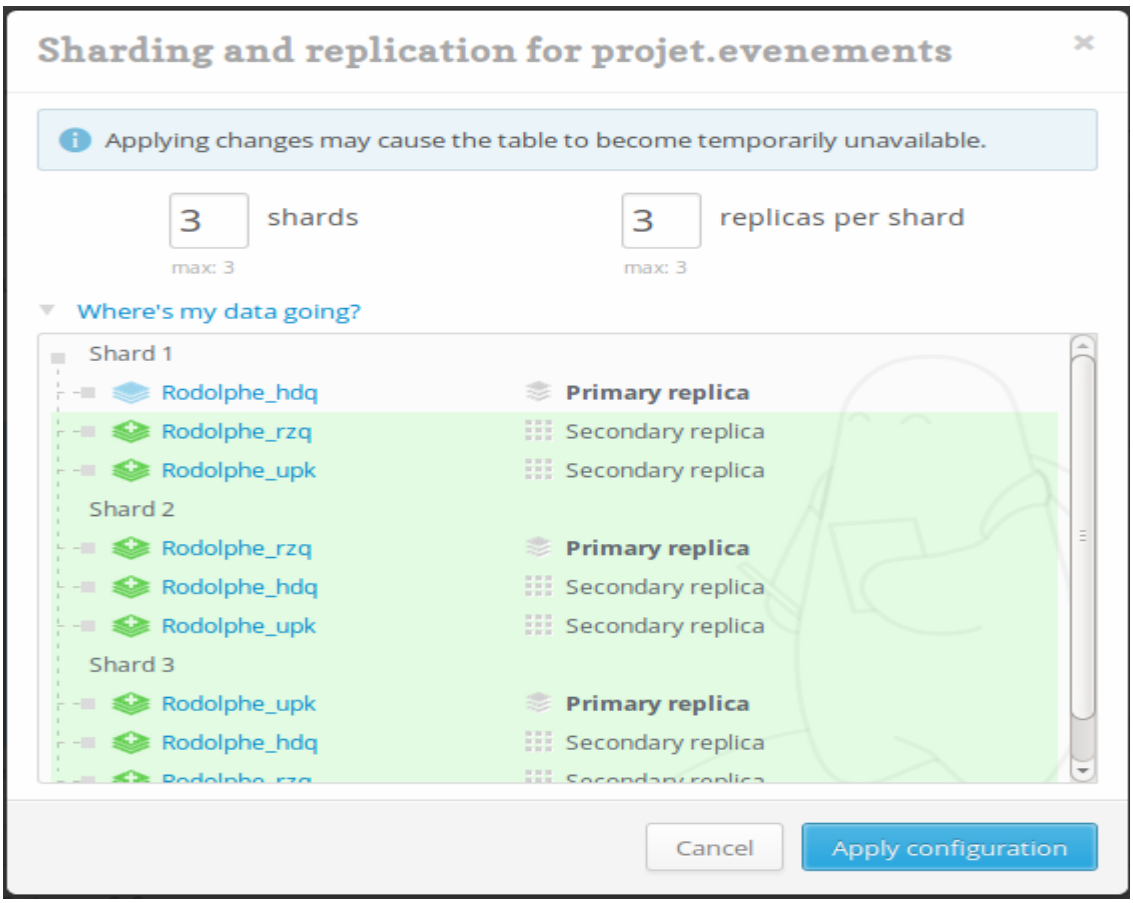
On observe dans **l'onglet Servers** les 3 serveurs disponibles.



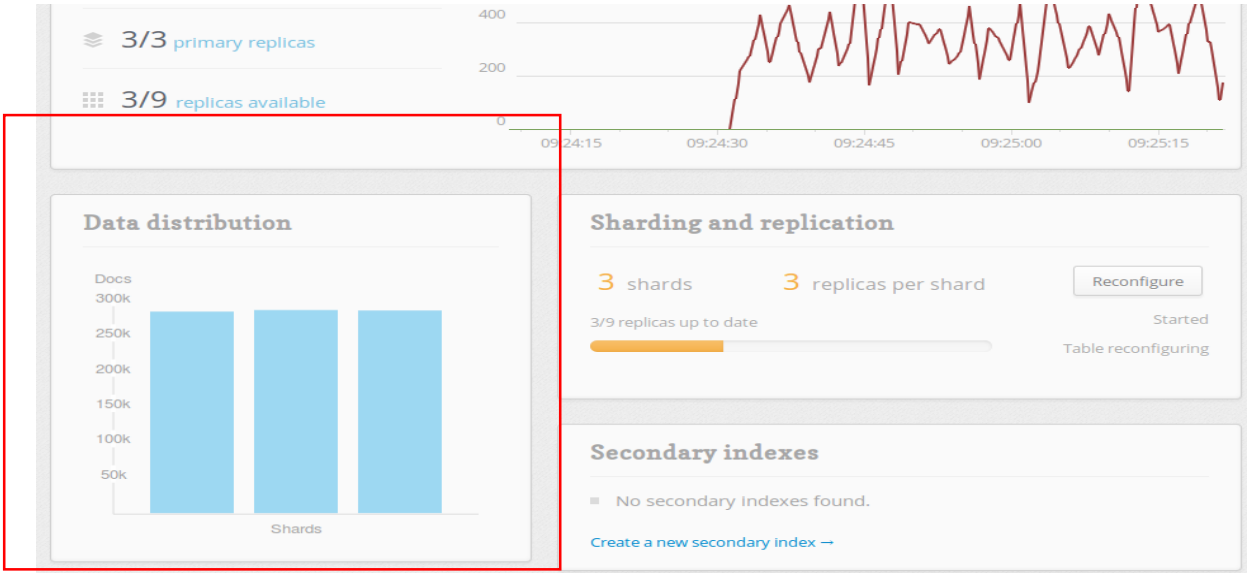
On souhaite maintenant partitionner la table en trois fragments et créer trois replicas de chaque fragment. Pour cela, on clique sur le bouton Reconfigure de la table Evenements



Le menu ci-dessous apparait et nous demande de paramétrer le nombre de partitions et de replicas souhaités. Nous choisissons trois fragments répliqués chacun trois fois. Un listing nous présente la répartition des réplicas des fragments parmi les serveurs.



La validation (apply configuration) lance le partitionnement et la constitution des réplicas. Le partitionnement par intervalle permet d'obtenir des partitions de tailles uniformes.



Concernant la cohérence/performance, on décide de garder les paramètres de base (*Majority, hard, single*) qui offrent un rapport cohérence-performance optimal dans le cadre de ce projet.

Création d'indexs

Afin d'accélérer les requêtes en lecture, nous allons créer des indexs sur les champs que nous souhaitons requêter dans la suite de ce projet. La commande suivante permet d'indexer le « champ à indexer» de la « table » du « database »

r.db(«database»).table(«table»).indexCreate(«champ à indexer»)
r.db(«projet»).table(«evenements»).indexCreate(«departement»)
r.db(«projet»).table(«evenements»).indexCreate(«categorie»)
r.db(«projet»).table(«evenements»).indexCreate(«genre»)
r.db(«projet»).table(«evenements»).indexCreate(«commune»)

III. Langage de requête REQL

a) Présentation

ReQL est un langage de requête propre au système NoSQL RethinkDB. Celui-ci présente 3 caractéristiques importantes, mises en avant dans la présentation de la documentation technique :

- ReQL s'intègre dans l'environnement de programmation usuel. En effet, il existe un package ReQL à importer pour conserver les outils et l'environnement de développement habituels.
- Les instructions du langage ReQL sont chainables (l'opérateur utilisé est le point). Cela offre une syntaxe plus simple à interpréter.
- Les requêtes s'exécutent sur le serveur et sont optimisées.

b) Manipulations

Etudions les commandes principales de ce langage

Création de table et ajout de données JSON

- On crée une table « manip dans la database « Projet » avec l'instruction suivante :

```
r.db('projet').tableCreate('manip')
```

- On peut y insérer des documents JSON manuellement avec la fonction insert(documents) :

```
r.db("projet").table("manip").insert([{codepostal:"80000",
population:133448},{codepostal :"69000",population : 484344},{codepostal :
"35000", population : 211373}])
```

Nous insérons ici trois documents JSON constitués d'un code postal et de la population en nombre d'habitants dans la table manip

Filtres sur condition

La fonction filter(conditions) permet de sélectionner les documents respectant les conditions. Dans le cas d'un document JSON dans lequel toutes les informations ne sont pas au même niveau, il faut préciser le schéma dans la requête.

Prenons l'exemple d'un événement dont la structure a été présentée en deuxième partie. Supposons que nous souhaitions faire un filtre sur le champ departement (conserver les documents dont le département est égal à 75). Celui-ci ne se trouve pas au premier plan mais à l'intérieur de la section intitulée fields (voir annexe 1).

La syntaxe suivante ne fonctionne pas

```
~~r.db("projet").table('evenements').filter(r.row('departement').eq("75"))~~
```

Il faut préciser le chemin pour accéder à ce champ

```
r.db('projet').table('evenements').filter(r.row('fields')('departement').eq
("75"))
```

D'autres expressions logiques sont disponibles :

Syntaxe	Correspondance
and	ET logique
or	OU Logique
eq	=
ne	Not =
gt	Plus grand que
ge	Supérieur ou égal
lt	Plus petit que
le	Plus petit ou égal
not	not

Suppression/Mise à jour de documents

On peut supprimer des documents avec la commande delete()

Par exemple, si l'on souhaite supprimer l'ensemble des événements réalisés dans le département 75, la commande est :

```
r.db("projet").table('evenements').filter(r.row('fields')('departement').eq
("75")).delete()
```

La mise à jour d'un document est réalisable. Imaginons que nous souhaitions modifier la population de la ville d'Amiens (la faire passer de 133488 à 135 000) dans notre table manip. Cela se réalise avec la commande update.

```
r.db("projet").table("manip").filter(r.row("codepostale").eq("80000")).upda
te({population:135000})
```

Choix des éléments renvoyés par la requête

Lorsque l'on réalise une requête, le système renvoi le document en entier, à moins de lui spécifier les champs à ramener. Cela s'effectue avec la commande pluck()

Par exemple, si l'on souhaite ramener seulement les champs code postale, commune, genre jour_min et jour_max des événements ayane teu lieu dans le 75, la commande sera :

```
r.db("projet").table('evenements').filter(r.row('fields')('departement').eq
("75")).pluck({"fields":["commune","code_postal","genre","jour_max","jour_m
in"]})
```

On peut également limiter le nombre de documents à renvoyer avec la commande limit(). Si l'on souhaite se limiter à 5 documents dans la requête ci-dessus, la commande devient :

```
r.db("projet").table('evenements').filter(r.row('fields')('departement').eq
("75")).pluck({"fields":["commune","code_postal","genre","jour_max","jour_m
in"]}).limit(5)
```

On peut trier le retour avec la commande orderBy.

Afficher par ordre croissant les communes :

```
r.db("projet").table('evenements').filter(r.row('fields')('departement').eq
("44")).pluck({"fields":["commune","code_postal","genre","jour_max","jour_m
in"]}).orderBy(r.row('fields')("commune"))
```

et par ordre décroissant

```
r.db("projet").table('evenements').filter(r.row('fields')('departement').eq
("44")).pluck({"fields":["commune","code_postal","genre","jour_max","jour_m
in"]}).orderBy(r.desc(r.row('fields')("commune")))
```

La commande distinct() permet d'éviter les doublons.

Fonctions mathématiques courantes

Fonction	Description
count()	Renvoie le nombre d'items
sum()	Renvoie la somme des éléments du champ entrée en paramètre

avg	Renvoie la moyenne des éléments du champ entrée en paramètre
min	Renvoie le minimum des éléments du champ entré en paramètre
max	Renvoie le maximum des éléments du champ entré en paramètre

Exemple :

```
r.db("projet").table("manip").avg("population")
```

Renvoie la moyenne des populations contenues dans la table manip.

Agrégation

On peut réaliser des opération par groupe avec la commande group(). Quelques exemples ci-dessous :

Compter le nombre d'événements publiés par genre d'événements :

```
r.db("projet").table("evenements").group(r.row('fields')('genre')).count()
```

Compter le nombre d'événements publiés par département :

```
r.db("projet").table("evenements").group(r.row('fields')('departement')).co
unt()
```

Jointures

Un avantage du langage ReQL est que l'on peut réaliser des jointures. Pour l'exemple, nous allons créer une nouvelle nouvelle table densité et y insérer les densités de population.

```
r.db("projet").tableCreate("densite")

r.db("projet").table("densite").insert([{cp:"80000", densite:2683},{cp
:"69000",densite : 10460},{cp : "35000", densite : 4195}])
```

L'objectif est de faire une jointure entre les tables **manip** (contenant le codepostal et la population de villes) et la table **densite** (contenant les champs cp et densite). La jointure sera réalisée entre les champs codepostal de la table manip et le champ cp de la table **densite**

Une jointure entre deux tables peut être réalisée dans RethinkDB sur les clés primaires ou les indexs secondaire. Pour cela, nous allons indéxer le champ codepostal de la table manip et le champ cp de la table densite.

```
r.db("projet").table("densite").indexCreate("cp")

r.db("projet").table("manip").indexCreate("codepostal")
```

La jointure utilise la commande eqJoin :

```
r.db("projet").table("manip").eqJoin("codepostal",r.db("projet").table("den
site"),{index:"cp"})
```

en jaune, nous avons le nom de la table de gauche et le nom de son champ à utiliser en jointure.

En gris nous trouvons le nom de la table de droite et le nom de son champ à utiliser en jointure

Le résultat conserve la distinction entre ce qui vient de la table de gauche et celle de droite. Ci-dessous le résultat obtenu pour la première ville :


```
{
   "left": {
      "codepostal": "35000" ,
      "id": "13ac9a19-4768-4aba-b672-1ebdad778fd1" ,
      "population": 211373
   } ,
   "right": {
      "cp": "35000" ,
      "densite": 4195 ,
      "id": "eed7d2c9-464b-4c3a-b930-dffc9371d2e7"
   }
```


Pour réunir ces informations facilement, il suffit d'utiliser la commande zip()

```
r.db("projet").table("manip").eqJoin("codepostal",r.db("projet").table("den
site"),{index:"cp"}).zip()
```


```
{
   "codepostal": "35000" ,
   "cp": "35000" ,
   "densite": 4195 ,
   "id": "eed7d2c9-464b-4c3a-b930-dffc9371d2e7" ,
   "population": 211373
}
```


On observe que les informations sont remises au même niveau.

IV. Références

Les éléments présentés dans ce projet se basent sur les sources suivantes :

- Site du système RethinkDB : https://www.rethinkdb.com/
- Site de l'enseignement NFE204 : Bases documentaires et NoSQL, du professeur Philippe Rigaux (CNAM PARIS) : http://www.bdpedia.fr/

Conclusion

RethinkDB est un système NoSQL intuitif et agréable à utiliser. Sa procédure d'import facilite grandement l'acquisition des données et son interface à partir du navigateur permet une gestion simple et efficace des bases de données. Avantage important, sa documentation claire et bien structurée permet de trouver rapidement les informations souhaitées.

Concernant son langage de requête ReQL, celui-ci est simple à manipuler grâce à ses instructions chainables. Il permet de réaliser de nombreuses opérations que cela soit en sélection de documents ou en opérations. Son apprentissage est rapide et sa syntaxe intuitive.

Ayant effectué des études en mathématiques, le cours RCP216 était un défi pour moi. Il m'a fallu acquérir des notions informatiques et travailler de nombreux prérequis. Fortement intéressé par ces sujets, j'ai pris plaisir à étudier le fonctionnement des bases de données distribuées et la recherche d'information. Ce projet, réalisé en autonomie, m'a permis d'utiliser les connaissances acquises dans ce cours et d'acquérir de la dextérité et de la confiance dans ce domaine porteur.

V. Annexe 1 : Exemple d'un événement au format JSON

{

"datasetid": "agenda_culturel" ,
"fields": {
"categorie": "Dédicace, animation autour d'un livre" ,
"code_insee": "61038" ,
"code_postal": "61130" ,
"commune": "Bellême" ,
"coordonnees_gps": [
48.37574 ,
0.561334
] ,
"date_modification": "2015-12-09" ,
"departement": "61" ,
"etat": "V" ,
"facette_debut": "2015-12-12" ,
"genre": "Livre, lecture publique..." ,
"genre_ident": "147" ,
"ident": 4866413 ,
"jour_1": "2015-12-12" ,
"jour_2": "2015-12-13" ,
"jour_max": "2015-12-13" ,
"jour_min": "2015-12-12" ,
"nav_lieu": "61/Bellême" ,
"organisme_ident": "273163" ,
"organisme_nom": "Amis du Perche de l'Orne, Société savante d'histoire, art et environnement" ,
"organisme_sous_type": "Association" ,
"photo_1_credit": "David Commenchal (Amis du Perche)" ,
"photo_1_legende": "Jean-François Suzanne, au Salon du livre du Perche" ,
"photo_1_path": http://photos.infolocale.fr/infolocale/annonce/2015/1208/4866413/_1.jpg, »
"photo_2_credit": "Michel Ganivet (Amis du Perche de l'Orne)" ,
"photo_2_legende": "Jean-François Suzanne présente "le Perche de 1914 à 1918"" ,
"photo_2_path": http://photos.infolocale.fr/infolocale/annonce/2015/1208/4866413/_2.jpg, »
"photo_3_credit": "Fédération des Amis du Perche" ,
"photo_3_legende": "Couverture du livre "Le Perche de 1914 à 1918"" ,
"photo_3_path": http://photos.infolocale.fr/infolocale/annonce/2015/1208/4866413/_3.jpg, »
"rubrique": "Dédicaces" ,
"rubrique_ident": "2" ,
"style": "Voir visiter" ,
"tarif_mention_gratuit": "non" ,
"tarif_mention_payant": "non" ,

"texte_fin": "Samedi 12 décembre de 14h à 21h et dimanche 13 décembre de 10h à 18h, Salle Philippe-de-Chennevière, place de l'Europe, Bellême. Contact : jean-francois.suzanne@orange.fr, www.amisduperche.fr." ,
"texte_milieu": "Président du comité scientifique de la fédération des Amis du Perche, Jean-François Suzanne a co-dirigé le colloque interrégional 2014 de Mortagne et la publication de l'ouvrage « Le Perche de 1914 à 1918 ». Le livre évoque la vie des percherons." ,
"theme": "Loisir" ,
"titre": "J-F Suzanne présente les Actes du colloque 2014 au marché de Noël"
} ,
"geometry": {
"coordinates": [
0.561334 ,
48.37574
] ,
"type": "Point"
} ,
"id": "0143afff-cf37-4810-ac3f-c4dbaf04adda" ,
"record_timestamp": "2015-12-12T05:34:00+01:00" ,
"recordid": "20d829e5a44f22fdb26f5618a37cc0099f518ddc"

}