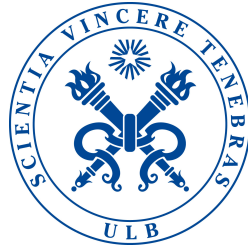


UNIVERSITÉ LIBRE DE BRUXELLES



INFO-H415 - ADVANCED DATABASES

---

# RavenDB & document stores

---

Authors:

Yasin ARSLAN

Jacky TRINH

Professor:

Esteban ZIMÁNYI

# Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
1.1	Présentation . . . . .	3
1.2	What is RavenDB ? . . . . .	3
1.3	Scenario . . . . .	4
<b>2</b>	<b>Design</b>	<b>5</b>
2.1	Document . . . . .	5
2.2	Document Store . . . . .	6
2.3	Collection . . . . .	6
2.4	Indexes . . . . .	7
2.5	Stale Indexes . . . . .	7
2.6	Replication . . . . .	8
2.7	Sharding . . . . .	8
2.8	Encryption . . . . .	9
2.9	Trigger . . . . .	9
2.10	Cascade Delete . . . . .	10
2.11	Supported language client side . . . . .	10
<b>3</b>	<b>Comparison</b>	<b>11</b>
3.1	RavenDB vs MongoDB . . . . .	11
3.1.1	Memory management . . . . .	11
3.1.2	Uncompressed field names . . . . .	11
3.1.3	Global write lock . . . . .	12
3.1.4	Safe off by default . . . . .	12
3.1.5	Offline table compaction . . . . .	12
<b>4</b>	<b>Example</b>	<b>13</b>
4.1	Introduction . . . . .	13
4.2	Server . . . . .	13
4.2.1	License . . . . .	13
4.2.2	Version . . . . .	14

---

4.2.3	Launch . . . . .	14
4.2.4	RavenDB Studio . . . . .	15
4.2.5	Replication . . . . .	17
4.3	Client . . . . .	17
4.3.1	Pyravendb library . . . . .	17
4.3.2	Log in and creating a session . . . . .	17
4.3.3	Get a Document . . . . .	18
4.3.4	Add a Document . . . . .	19
4.3.5	Delete a Document . . . . .	20
<b>5</b>	<b>Conclusion</b>	<b>22</b>

# Chapter 1

## Introduction

### 1.1 Présentation

As part of the course INFO-H415 Advanced Databases, we have been asked to study a database technology and illustrate it with an application developed in a database management system. Nowadays, there is a lot of database systems and each of them has its own setup and is made for specific tasks. For example, one of the most famous, MySQL is an open-source relational database management system. It is based on the relational model, data is stored in two-dimensional tables called relations or tables. All relational databases use SQL ( Structured Query Language ) for querying and maintaining the database. In our case, we will study a different kind of database system called document-oriented database.

### 1.2 What is RavenDB ?

RavenDB is an open source document-oriented database, also called document store. It is mainly used for storing, retrieving and managing document-oriented data. There is no separation between data and the schema, this model is called semi-structured data and has many advantages such as easily modifying the schema or providing a flexible format of data exchange between different types of databases. The popularity of document-oriented database has grown, it is now one of the main categories of NoSQL databases. RavenDB is used by popular customers such as Toyota, Vodafone or JetBrains. Its typical application scenarios are ecommerce, financial or healthcare.

## 1.3 Scenario

As an example of use, we imagined the management of a large supermarket where each order must be registered and it must be possible to access the details. The supermarket hired 2 students from the Université Libre de Bruxelles to design a RavenDB to store all the information and a Python3 script that will allow a registered employee with the right permissions to access it, to modify it or even to add new information. The database should list at least the following information : Employees, Orders, Companies/Partners and Products.

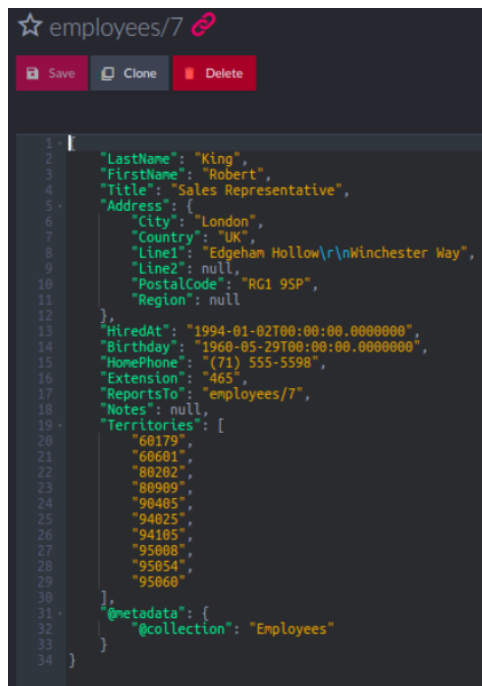
# Chapter 2

## Design

RavenDB uses some new concepts that are totally different from the standard relational database. This section will provide the necessary informations in order to have some understanding of these new concepts and features.

### 2.1 Document

Since RavenDB is a document-oriented database, we should first define it. A document is a self-describing, hierarchical tree data structure which can consist of maps, collections, and scalar values. They are formatted in plain JSON and stored in the database. The Document Store will manage them and to organize it, RavenDB and other Document Databases uses Collections .



```
1  "LastName": "King",
2  "FirstName": "Robert",
3  "Title": "Sales Representative",
4  "Address": {
5    "City": "London",
6    "Country": "UK",
7    "Line1": "Edgeham Hollow\r\nWinchester Way",
8    "Line2": null,
9    "PostalCode": "RG1 9SP",
10   "Region": null
11  },
12  "HiredAt": "1994-01-02T00:00:00.0000000",
13  "Birthday": "1960-05-29T00:00:00.0000000",
14  "HomePhone": "(71) 555-5598",
15  "Extension": "465",
16  "ReportsTo": "employees/7",
17  "Notes": null,
18  "Territories": [
19    "60179",
20    "60601",
21    "80202",
22    "80909",
23    "90405",
24    "94025",
25    "94105",
26    "95008",
27    "95054",
28    "95060"
29  ],
30  "@metadata": {
31    "@collection": "Employees"
32  }
33 }
34
```

Figure 2.1: Example of a document

## 2.2 Document Store

It is designed for storing, retrieving and managing document-oriented information. In RavenDB, a document store initiates and manages the connection between an application and a database instance. With the document store, we can run all operations on an associated server instance.

The document store object possesses only one URL address and this address points to a RavenDB server. The object can, however, operate on the other existing databases related to the server.

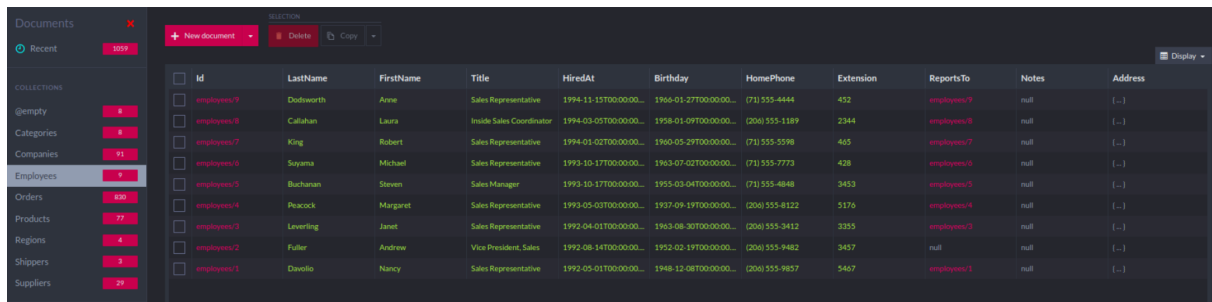
## 2.3 Collection

RavenDB stores all the documents in the same storage space. None of them are physically grouped. In order to be efficient, we certainly need some level of organization. For example, documents containing customers data are strictly different from documents containing products information and we do need to talk about groups of them. Thus, RavenDB allows us to stamp a document with a string value that will define its type (like "Customers" and "Products").

Therefore, a collection is a set of documents that have the same tag. A tag is defined in the *Raven-Entity-Name* metadata which is automatically filled in by the client based on the type of an entity object. Since RavenDB is schema-less, there aren't any problem if the documents within a same collection have completely different structure.

We also have to note that a collection is a virtual concept, so it doesn't affect how or where the documents that are in the same collection are stored. However this concept is useful for three features:

- Studio: An *Example* entity that we have stored will be visible under *Examples* collection (the client automatically pluralizes the collection name). We can here visualize a virtual concept through the studio since each RavenDB database possess the built-in *Raven/DocumentsByEntityName* index. This allows the studio to retrieve only documents from the specified collection and thus group the documents into the collections.
- Indexing: Another purpose of collection is filtering documents during indexing process. When an index is created, we also determine what collection does it involve. During indexing process, only the documents that belong to this collection are indexed. We'll talk later about the indexes.
- Document keys: The document's identifier looks like *examples/17* where *examples* is the collection name and *17* is the identity value. This is the default convention.



Id	LastName	FirstName	Title	HiredAt	Birthday	HomePhone	Extension	ReportsTo	Notes	Address
employees/9	Dollworth	Arne	Sales Representative	1994-11-15T00:00:00...	1960-01-27T00:00:00...	(71) 555-4444	452	employees/7	null	(...)
employees/8	Callahan	Laura	Inside Sales Coordinator	1994-03-05T00:00:00...	1958-01-09T00:00:00...	(206) 555-1189	2344	employees/8	null	(...)
employees/7	King	Robert	Sales Representative	1994-01-02T00:00:00...	1960-05-29T00:00:00...	(71) 555-5598	465	employees/7	null	(...)
employees/6	Suyama	Michael	Sales Representative	1993-10-17T00:00:00...	1965-07-02T00:00:00...	(71) 555-7773	428	employees/6	null	(...)
employees/5	Buchanan	Steven	Sales Manager	1993-10-17T00:00:00...	1955-03-04T00:00:00...	(71) 555-4848	3453	employees/5	null	(...)
employees/4	Peacock	Margaret	Sales Representative	1993-05-03T00:00:00...	1937-09-19T00:00:00...	(206) 555-8122	5176	employees/4	null	(...)
employees/3	Levering	Janet	Sales Representative	1992-04-01T00:00:00...	1963-08-30T00:00:00...	(206) 555-3412	3355	employees/3	null	(...)
employees/2	Fuller	Andrew	Vice President, Sales	1992-08-14T00:00:00...	1952-02-19T00:00:00...	(206) 555-9482	3457	null	null	(...)
employees/1	Diavolo	Nancy	Sales Representative	1992-05-01T00:00:00...	1948-12-08T00:00:00...	(206) 555-9857	5467	employees/1	null	(...)

Figure 2.2: Example of a Collection through the studio

## 2.4 Indexes

In a relational database like MySQL, when sending a query, the database performs with a table scan. It means that all documents are analyzed and those that meet the predicate are selected. It does seem logical and efficient... as long as the number of data stored in the database is small but it becomes really slow when the amount of data reaches a significant size. Thus it is not optimized. That's why RavenDB uses indexes to speed up the queries.

Indexes are functions used in the side of the server and define using which field and what values document can be searched on and represent the only way to satisfy queries in RavenDB. The indexing process is triggered in the background whenever data is added or changed. By doing so, the server can respond quickly even if a lot of data has changed and this allows us to spare costly table scans operations. To achieve fast response times, RavenDB use a mapping function with LINQ-like syntax for every index and the result is then converted to *Lucene* (open-source search software) index entry that lasts for future use. This avoid re-indexation each time the query is issued.

## 2.5 Stale Indexes

Because RavenDB's indexes work this way, the results might be stale. Indeed, RavenDB assumes that the user should never suffer from assigning a big tasks to a server. For them, it is better to be stale than offline. Thus the queries will always have results even if it is not already up-to-date.

Even if it involves re-indexing so many documents, RavenDB will return quickly each client request. Since the previous request has returned so quickly, the next query can be made a millisecond after that, and the results will be returned.



## 2.6 Replication

We can also enable the replication feature when creating a new database. Every now and then, it will replicate our data from our database to another database that we choose.

This feature allows us to track the server where the document was originally written and by doing so we can determine if a replicated document is conflicting with the existing document. Those conflicts will be marked and will require automated or user involvement in order to be resolved.

The replication feature will not replicate any system documents but only the data contained in the database. It will even replicates the indexes every ten minutes (this can, of course, be changed in the configuration).

The replication are done on every transaction commit. RavenDB will look up the list of replication destinations and for each of the destination, it will query the remote instance for the last document that was replicated and then start sending batches of updates that happened since the last replication. This is happening in the background and in parallel.

## 2.7 Sharding

RavenDB has a native sharding support. The concept of sharding allows us to split our data across multiple servers, meaning that each server holds just a portion of your data. It is really useful when we have to handle a lot of data. For example, with the sharding system, a big company having multiple headquarter across the world can store the headquarter's data on a shard which depends on the headquarter's region. The one located in Europe would be stored on one shard, the one located in America would be stored on a second shard, etc...

The main idea is to put the data used by the corresponding headquarter on a shard geolocated near the location, so the headquarter get served from a nearby server and respond more quickly to a user. Another advantage of this system is that it reduce the load on each server since it only has to handle a portion of the complete data

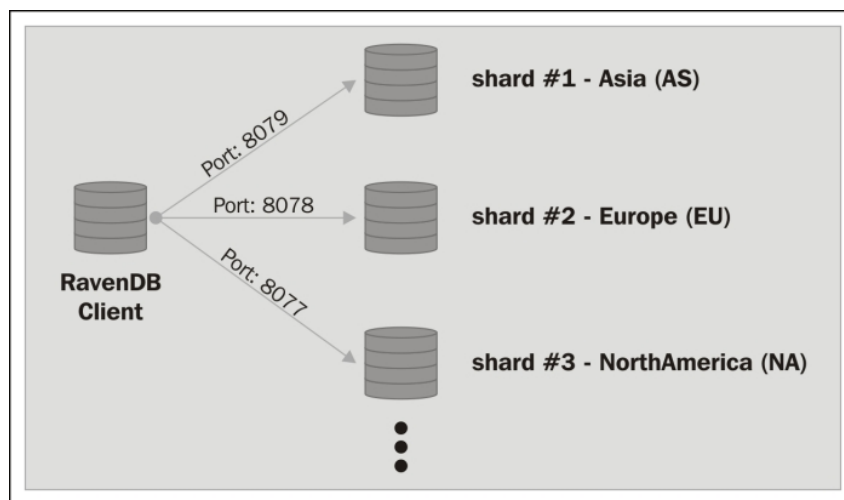


Figure 2.3: Example of multiple shards

## 2.8 Encryption

RavenDB also proposes an *encryption bundle* in order to secure our documents. By default, it uses the popular AES-128 (Advanced Encryption Standard - 128) encryption algorithm but can be changed if needed. The encryption is completely transparent for the end-user and is applied to all documents stored and to all indexes as well.

## 2.9 Trigger

A trigger is a procedural code that is automatically executed upon a specific event on a particular collection which is mostly used to maintain the integrity of the information on the database. RavenDB proposes five categories of triggers:

- *PUT triggers*: As his name indicates, upon a specific event, an addition or a simple modification will be done.
- *DELETE triggers*: Similar in shape to PUT triggers, yet in contrast to them, they control the delete operations.
- *Read triggers*: This type of triggers allows us to control the access to documents and manipulate their context when performing a simple read operations. With this, we can forbid some users from reading some documents and also on the server side, if a document with a link was requested, we are able to stitch such document together with its link to create a single document.
- *Index Query triggers*: Query triggers have been introduced to extend the query parsing capabilities and this give users a way to modify the queries before they are executed against the index

- *Index Update triggers*: This allows us to perform custom action when an index entry has been created or deleted.

## 2.10 Cascade Delete

The cascade delete is also a bundle that allows to delete a specified set of documents and attachments when the document that owns it is deleted. For example, we can use it to delete an attachment that is referenced by a document or to remove a set of child documents referenced by a parent document.

## 2.11 Supported language client side

RavendDB supports the most basic language which is C# and also the popular ones like Java and Python. We can also have access to the database using HTTP requests.

# Chapter 3

## Comparison

### 3.1 RavenDB vs MongoDB

On the surface, RavenDB and MongoDB are really similar although MongoDB is the most popular Document-oriented database right now. They have schemalessness, easy replication, rich query language and can be accessed from multiple languages. But under the hood, they work differently.

#### 3.1.1 Memory management

In MongoDB case, the operating system is the one doing the memory management. By doing so, they avoid to work on a memory manager and thus their memory management is actually poor.

Since RavenDB is a managed application, they don't have a direct control over memory but they do manage it. There are several layers of caching in place since RavenDB knows more than the operating system about the user's own usage scenarios. When making a new request, the latter would never hit the disk because RavenDB keeps track on hot data and makes sure that it stays in memory. This is done for both indexes and obviously documents.

The operating system memory manager is certainly more optimized but the database know what is going on and is able to predict the usage patterns.

#### 3.1.2 Uncompressed field names

It is considered good practice to actually shorten the field names in MongoDB to optimize the space management.

Raven does not compress the field names and even if it does not, in contrary to MongoDB, it is not considered a good practice to do it. We can all relate that it can easily become a horrible mess when there are so many short field names and it becomes hard to figure

out what they mean. It is really not convenient to figure out what a document is about with the actual content.

That is why RavenDB prefers to do a full response and request compression and also allows to do document compression on disks as well.

### 3.1.3 Global write lock

MongoDB possesses a process-wide write lock, meaning that all other operations including read are blocked because of the lock.

RavenDB also have a write lock but it does not block all operations like MongoDB. The only operation that it blocks is actually the write operation and it does not interfere with reads or indexes. Therefore it is more convenient.

### 3.1.4 Safe off by default

Unlike MongoDB, RavenDB has an unique feature called *safe by default*. This feature allows the database to stop users querying for large amounts of data which is never a good idea. Indeed, queries that return thousand of records is inefficient and will take a very long time. *Safe by default* limits the number of records received by default on the client side to 128 (which is configurable). This is attempted to stop a developer from writing poor queries.

### 3.1.5 Offline table compaction

To let MongoDB compact its data on disk, we need to take the server down. In RavenDB case, all the maintenance tasks are done while the server is still up and it can still answer the requests from users.

# Chapter 4

## Example

### 4.1 Introduction

For this part of the report, we will simulate the initialization and the launch of RavenDB from Scratch. Once done, the python3 application will allow us to connect to access to the database.

### 4.2 Server

#### 4.2.1 License

RavenDB allows to choose between 3 licenses : Community, Professional and Enterprise.

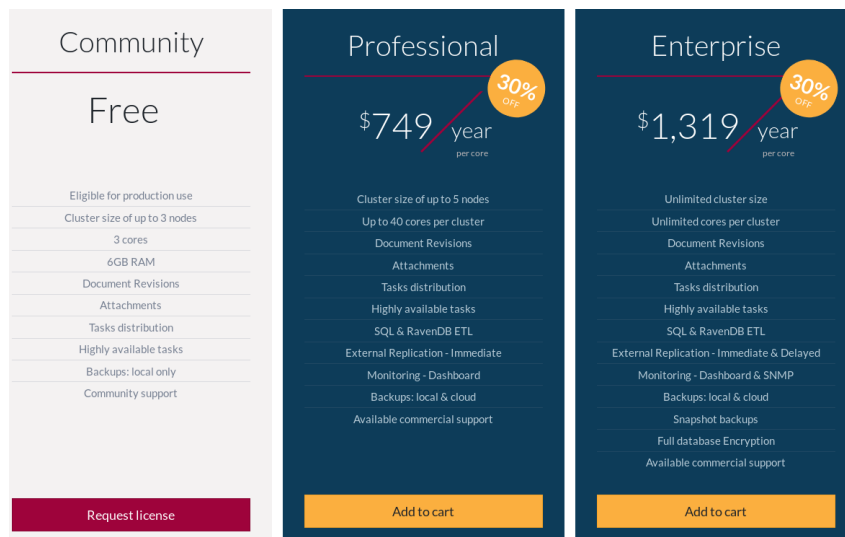


Figure 4.1: RavenDB licenses

For this example, we chose the free license. It is more than enough to introduce the public to RavenDB.

## 4.2.2 Version

We use the latest version of RavenDB, version 3.5.5 that came out in November 2017.

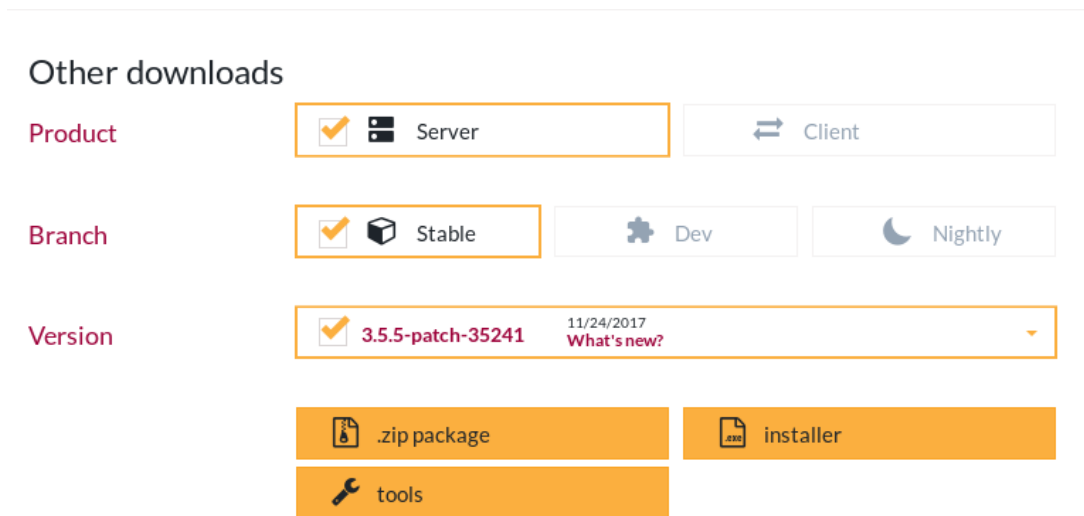


Figure 4.2: RavenDB version

## 4.2.3 Launch

Once the download completed, we can start a bash script to launch our database and automatically open the RavenDB studio.

```
#!/bin/bash

VERSION_PATH="./version.txt"
EXEC_PATH="./Server/Raven.Server"

ASSEMBLY_VERSION=$(eval $EXEC_PATH --version)
VERSION=""

if [[ -f "$VERSION_PATH" ]]; then
    VERSION=$(cat $VERSION_PATH);
fi

if [[ "$ASSEMBLY_VERSION" != "$VERSION" ]]; then
    echo "$ASSEMBLY_VERSION" > "$VERSION_PATH"
    xdg-open "http://ravendb.net/first-run?type=start&ver=$ASSEMBLY_VERSION";
fi

# avoid Firefox already open warning preventing from launching the Studio tab
sleep 2
eval "$EXEC_PATH --browser";
```

Figure 4.3: Bash script to launch RavenDB

#### 4.2.4 RavenDB Studio

RavenDB Studio will allow the user to have a good visual interface of his databases. In this example, we have 3 databases with respectively 3, 0 and 1058 documents. Once the database selected, we can access to the Collection and then to any Document of that Collection.

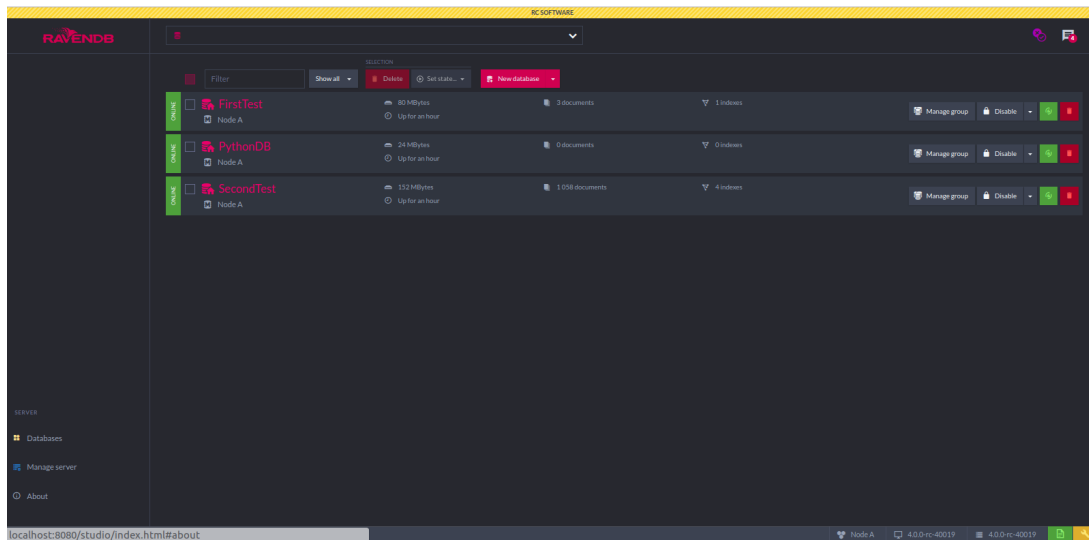


Figure 4.4: RavenDB Studio

Collection Name	Document Count
@empty	8
Categories	7
Companies	91
Employees	9
Orders	830
Products	77
Regions	4
Shippers	3
Suppliers	29

Figure 4.5: All collections in the chosen database



<input type="checkbox"/>	<b>Id</b>	<b>Name</b>	<b>Supplier</b>	<b>Category</b>	<b>QuantityPerUnit</b>	<b>PricePerUnit</b>	<b>UnitsInStock</b>	<b>UnitsOnOrder</b>	<b>Discontinued</b>	<b>ReorderLevel</b>
<input type="checkbox"/>	products/62	Tarte au sucre	suppliers/29	categories/3	48 pies	49.3	17	0	false	0
<input type="checkbox"/>	products/61	Sirup d'érable	suppliers/29	categories/2	24 - 500 ml bottles	28.5	113	0	false	25
<input type="checkbox"/>	products/60	Camembert Pierrot	suppliers/28	categories/4	15 - 300 g rounds	34	19	0	false	0
<input type="checkbox"/>	products/59	Raclette Courdavault	suppliers/28	categories/4	5 kg pkg.	55	79	0	false	0
<input type="checkbox"/>	products/58	Escargots de Bourgogne	suppliers/27	categories/8	24 pieces	13.25	62	0	false	20
<input type="checkbox"/>	products/57	Ravioli Angelo	suppliers/26	categories/5	24 - 250 g pkgs.	19.5	36	0	false	20
<input type="checkbox"/>	products/56	Gnocchi di nonna Alice	suppliers/26	categories/5	24 - 250 g pkgs.	28	21	10	false	30
<input type="checkbox"/>	products/55	Pâté chinois	suppliers/25	categories/6	24 boxes x 2 pies	24	115	0	false	20
<input type="checkbox"/>	products/54	Tourtière	suppliers/25	categories/6	16 pies	7.45	21	0	false	10
<input type="checkbox"/>	products/53	Perth Pasties	suppliers/24	categories/6	48 pieces	32.8	0	0	true	0
<input type="checkbox"/>	products/52	Filo Mix	suppliers/24	categories/5	16 - 2 kg boxes	7	38	0	false	25
<input type="checkbox"/>	products/51	Manjimp Dried Apples	suppliers/24	categories/7	50 - 300 g pkgs.	53	20	0	false	10
<input type="checkbox"/>	products/50	Valkoinen suklaa	suppliers/23	categories/3	12 - 100 g bars	16.25	65	0	false	30
<input type="checkbox"/>	products/49	Masilaku	suppliers/23	categories/3	24 - 50 g pkgs.	20	10	60	false	15
<input type="checkbox"/>	products/48	Chocolade	suppliers/22	categories/3	10 pkgs.	12.75	15	70	false	25
<input type="checkbox"/>	products/47	Zaarse koeken	suppliers/22	categories/3	10 - 4 oz boxes	9.5	36	0	false	0
<input type="checkbox"/>	products/46	Spegelsid	suppliers/21	categories/8	4 - 450 g glasses	12	95	0	false	0
<input type="checkbox"/>	products/45	Rogede slid	suppliers/21	categories/8	1k pkg.	9.5	5	70	false	15
<input type="checkbox"/>	products/44	Gula Malacca	suppliers/20	categories/2	20 - 2 kg bags	19.45	27	0	false	15
<input type="checkbox"/>	products/43	Iyah Coffee	suppliers/20	categories/1	16 - 500 g tins	46	17	10	false	25

Figure 4.6: An example of Collection

```

1  [
2    {
3      "Name": "Tarte au sucre",
4      "Supplier": "suppliers/29",
5      "Category": "categories/3",
6      "QuantityPerUnit": "48 pies",
7      "PricePerUnit": 49.3,
8      "UnitsInStock": 17,
9      "UnitsOnOrder": 0,
10     "Discontinued": false,
11     "ReorderLevel": 0,
12     "@metadata": {
13       "@collection": "Products"
14     }
15   }
16 ]

```

Figure 4.7: A document from the chosen Collection

## 4.2.5 Replication

When creating a new database, we can enable the replication feature. Then, we have to create a document *Raven/Replication/Destinations* which tell RavenDB where to replicate.

```
1  {
2    "Destinations": [
3      {
4        "Url": "http://server1:8080/"
5      },
6      {
7        "Url": "http://server2:8080/"
8      }
9    ]
10 }
```

Figure 4.8: Exemple of replication

## 4.3 Client

### 4.3.1 Pyravendb library

To make it work on your personal computer, you will need to install pyravendb version 3.5.3.7. It can be installed using pip or by downloading the zip on the pypi website and launching the setup.py file.

### 4.3.2 Log in and creating a session

To have access to the application, you will first need to log in. In this way, we block this part of the application to employees who do not have the required permissions.

```
yasinrjs@yasinrjs-ThinkPad-T440p:~/Bureau/AdvDbProj/Code$ python3 dbScript.py
Enter your ID : Yasin
Enter your password : 11
-> Login refused, please retry.

Enter your ID : Yasin
Enter your password : 123
-> Login Successfull
```

Figure 4.9: Log in phase

To connect on the RavenDB, we first need a RavenDB document store. For any action we want to perform on the RavenDB, we start by obtaining a new Session object from the document store. The Session object will contain everything needed to perform

any operation necessary. In order, you will open the session, do an action and finally apply the changes to the RavenDB server.

```
#####  
# Initialize connection  
store = document_store.DocumentStore(urls=["http://localhost:8080"], database="SecondTest")  
store.initialize()  
#####  
with store.open_session() as session:
```

Figure 4.10: Opening a session

### 4.3.3 Get a Document

To retrieve a Document, you will only need the ID of the Document you are looking for. We decided to generate the .json Document and to store it in the repository ./Get, it gives a better visual for the employees.

```
What would you like to do ?  
1. Get a document, 2. Add a document, 3. Remove a document, Anything else to quit  
Your choice : 1  
----- GET A DOCUMENT -----  
ID of the document : products/42  
--> New file created in ./Get named : products42.json .
```

Figure 4.11: Getting a document

```
products42.json x  
{  
  "QuantityPerUnit": "32 - 1 kg pkgs.",  
  "Name": "Singaporean Hokkien Fried Mee",  
  "Supplier": "suppliers/20",  
  "PricePerUnit": 14,  
  "Discontinued": "True",  
  "Category": "categories/5",  
  "ReorderLevel": 0,  
  "UnitsInStock": 26,  
  "UnitsOnOrder": 0}
```

Figure 4.12: Json file produced after getting a Document

Pyravendb allows you to get a Document in different way. You can either get it as it is stored in the database or you can directly convert it to a Python object. This allows the programmer to work with Python objects and not json.

```

class Categorie(object):
    def __init__(self, Name, Description):
        self.Name = Name
        self.Description = Description

def loadItem(session, ID):
    data = session.load(ID)
    return data

def loadCategorie(session, ID):
    data = session.load(ID, object_type=Categorie)
    return data

```

Figure 4.13: Getting a Document as json or object Python3

#### 4.3.4 Add a Document

To add a Document, the employees will have to create it manually. Once the type selected and the new Document created, it will be sent to the database and stored as json.

```

What would you like to do ?
1. Get a document, 2. Add a document, 3. Remove a document, Anything else to quit
Your choice : 2
----- ADD A DOCUMENT -----
Select a collection :
    1. Product
    2. Categorie
Your choice : 2
Name : Drugs
Description : Never order this
--> New Categorie added with success !

```

Figure 4.14: Adding a new Document

Id	Name	Description
categories/513-A	Drugs	Never order this
categories/8	Seafood	Seaweed and fish
categories/7	Produce	Dried fruit and bean curd
categories/6	Meat/Poultry	Prepared meats
categories/4	Dairy Products	Cheeses
categories/3	Confections	Desserts, candies, and sweet breads
categories/2	Condiments	Sweet and savory sauces, relishes, spreads, and seasonings
categories/1	Beverages	Soft drinks, coffees, teas, beers, and ales

Figure 4.15: Visual on the new Document

You can either use a json format to send it to the database or use a Python object. RavenDB will know where to add the Document given the object type.

```
def storeItem(session, data):
    session.store(data)
    session.save_changes()
```

Figure 4.16: Store a Document Python3

### 4.3.5 Delete a Document

When you want to delete an existing Document, the employees will only have to give the ID of the Document they want to remove. If the said Document exists, it will create a backup json file and store it in the ./Delete repository.

```
What would you like to do ?
1. Get a document, 2. Add a document, 3. Remove a document, Anything else to quit
Your choice : 3
ID of the document : products/60
--> Backup created in ./Delete named : products60.json .
--> products/60 deleted with success !
```

Figure 4.17: Deleting a Document

```
products60.json x
{"UnitsOnOrder": 0,
  "Name": "Camembert Pierrot",
  "Supplier": "suppliers/28",
  "PricePerUnit": 34,
  "Category": "categories/4",
  "QuantityPerUnit": "15 - 300 g rounds",
  "ReorderLevel": 0,
  "Discontinued": "False",
  "UnitsInStock": 19}
```

Figure 4.18: Backup created in case of bad move.

It works the same as Adding a new Document. You can either give it Json format or directly Python since it has to first load it and then delete from the database.

```
def deleteItem(session, data):  
    data = loadItem(session, data)  
    session.delete(data)  
    session.save_changes()
```

Figure 4.19: Delete a document Python3

# Chapter 5

## Conclusion

To conclude, it was a very pleasant experience to learn something new. Indeed, we always thought that the standard relational SQL database was the only kind of database that exists. We were kind of surprised to learn that there exists a vast panoply of database. Discovering what RavenDB could offer was great, their technologies are, as their motto implies, optimized for efficiency. It was really easy to use and was also user-friendly. The first thing we did after downloading and installing the server was to try out the studio made by RavenDB. That's where we understood the concept of collection and indexes. With RavenDB studio, we also could create a random database with random data to play with it and to learn the mechanisms behind. We were convinced by the optimization in time provided by the implementation of the indexes.

For the client side application, RavenDB proposes different supports and we had to choose one. We decided to go for Python because it was the newest to join the list but we encountered a few problems because RavenDB's API for this language was kind of poor. They provided the first version of the API with the release of RavenDB version 3.5 in the end of 2016. Indeed, for example, we could not modify a document but we had to do a trick in order to have the same result: deleting and then adding. While this was a feature in other supports such as Java by deleting it and adding a new one.

# Bibliography

- [1] *Raven's Documentation*, <https://ravendb.net/docs/article-page/3.5/python/>
- [2] *Why I said goodbye to MongoDB*, <https://dzone.com/articles/why-i-said-goodbye-mongodb>
- [3] *Python client*, <https://github.com/ravendb/RavenDB-Python-Client>
- [4] *LINQ*, <https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/linq/introduction-to-linq-queries>
- [5] *Lucene*, <http://lucene.apache.org/>
- [6] *Document Oriented Database*, [https://en.wikipedia.org/wiki/Document-oriented\\_database](https://en.wikipedia.org/wiki/Document-oriented_database)