Les bases NoSQL et Python

Youenn Boussard

Alter Way

afpy
Association Francophone Python

Les bases de données

- Avant 1960 : organisation classique sous forme de fichier
- 1960 : 1er base de donnée : militaire , hiérarchique, sous forme d'arbre
- 1970 : Théorie sur l'algèbre relationnelle de Codd
- 1980 : troisième génération des sgbd : les bases de données orientées objets

Le développement Internet

- 1962 – Premier concept d'internet
- 1969 – RFC
- 1974 – Mise au point de la norme IP
- 1981 – 213 ordinateurs connectés
- 1989 – Naissance du World wide Web
- 2004 – Web 2.0, Facebook
- 2006 – twitter

186,7 millions de sites web

Web 2.0 et les limites des modèles relationnels

- Beaucoup de données
  - 

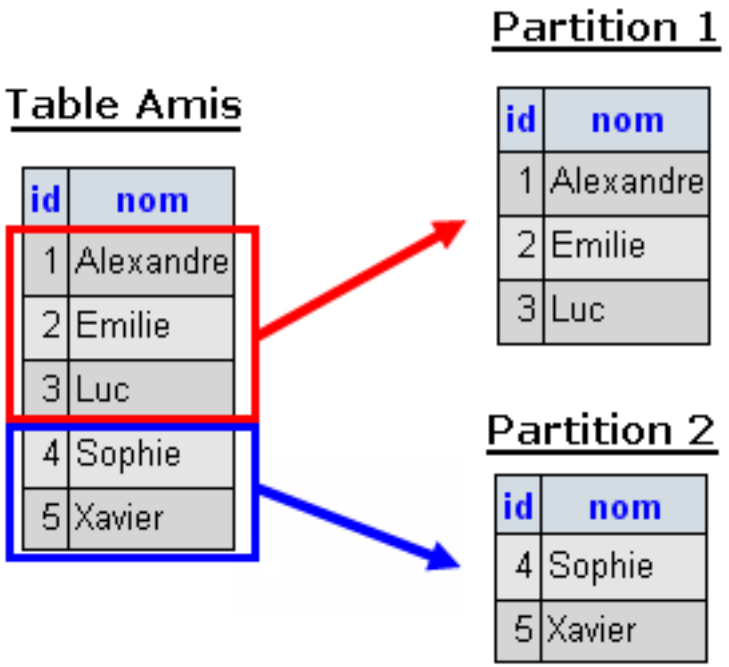
Digg 3TB
  - 

Facebook 50TB
  - 

Ebay 2PB
- Beaucoup d'utilisateurs
- Beaucoup de trafic
- Et les bases relationnelles pour gérer tout cela ?

Le cas de digg™

- Partition verticale maître-esclave avec MySQL



Maître esclave

- Réplication Maître esclave
  - Un unique maître
  - Un ou plusieurs esclave
  - Toute les écritures vont sur le maître, répliquer sur les esclaves
  - Les lectures sont réparties entre les différents maîtres et esclaves
- Cela implique
  - Le maître est critique
  - Le maître est le point d'engorgement du système

Partition Horizontale

- Réparties sur plusieurs noeuds
- Les jointures sont déportées au niveau de l'application
- Cela implique
  - On n'est plus relationnel
  - Le fonctionnel devient compliqué
  - La maintenance aussi !!

Et la disponibilité !!

Table Diggs
- Id
- Itemid
- Userid
- Digdate

Table Friends
- Id
- UserId
- UserName
- FriendId
- FriendName
- Mutual
- Date_created

Des millions de lignes

Des centaines de millions de lignes

Probleme : Afficher
les amis qui on cliqué sur l'un de mes diggs (fonctionnalité du green badge)

Sytème distribué : Trois états

Dr. Eric Brew

CAP THEOREM =
On ne peut avoir que
2 états en simultanée dans un système distribué

ACID contre BASE

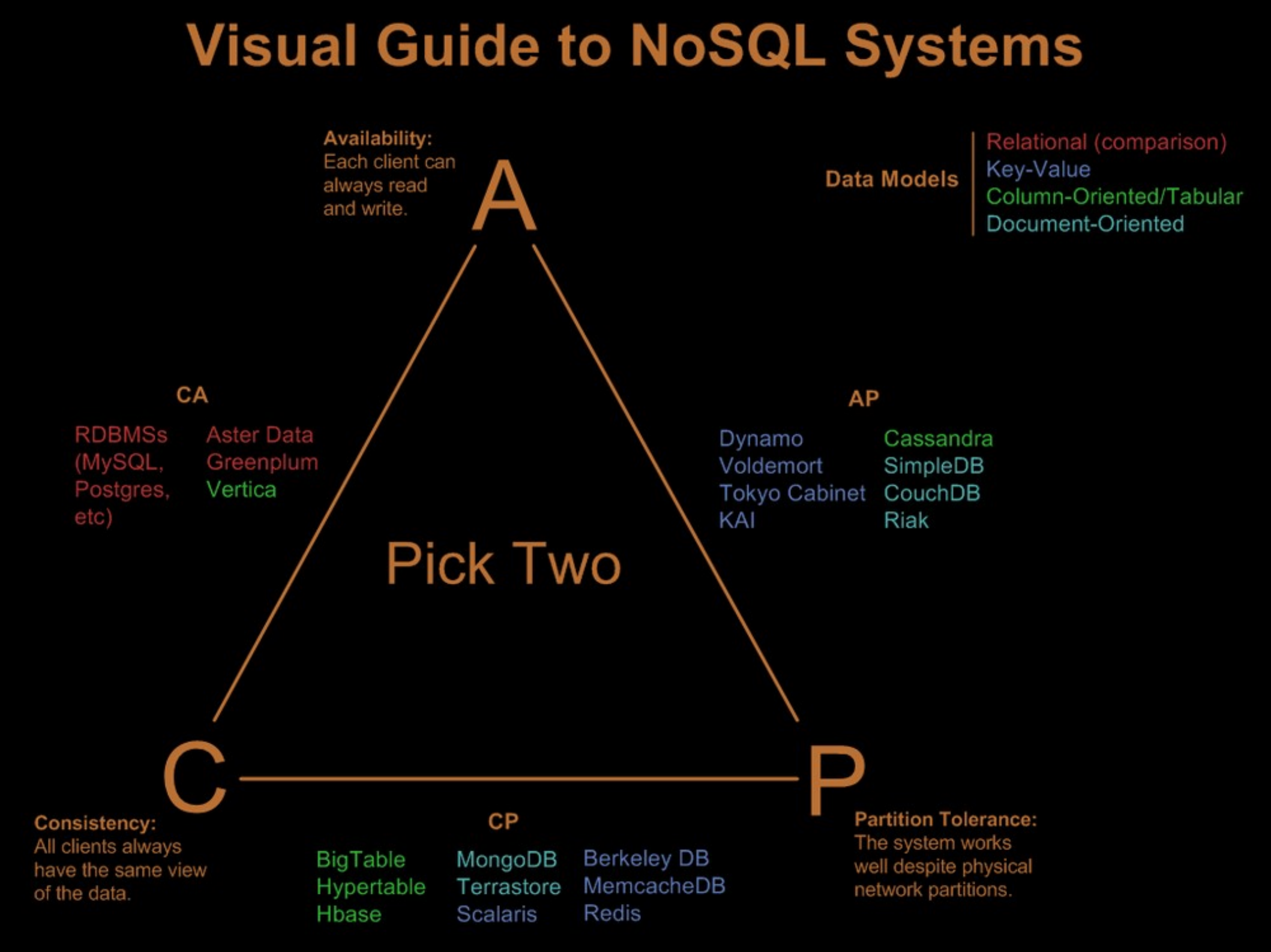
- Atomique
- Cohérente
- Isolée
- Durable

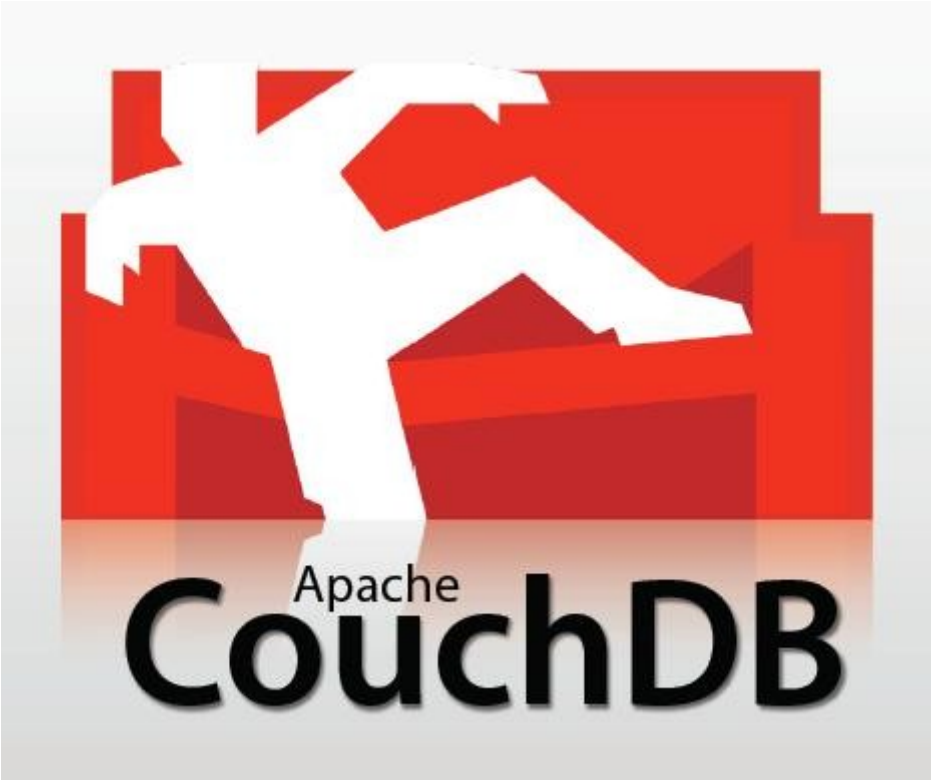
- Basically Available
- Soft state
- Eventually consistent

Une nouvelle Génération de base de donnée

- Non relationnelle
- Distribuée
- Open source
- Scalable horizontablement

Visual Guide to NoSQL Systems




Apache
CouchDB

CouchDB est

- Orienté document → Pas de schéma, représentation des données telle quelle
- Accessible via RESTful
- Distribué, réplication incrémental, résolution de conflit bi directionnel
- Donnée indexable et requetable suivant des vues
- Ecrit en 

Un document

- Un document est un objet contenant un ensemble de clés valeurs


```
{"_id": "j_aime_le_mouvement_no_sql"
"_rev": "1-1"
"Subject": "J'aime le mouvement no sql"
"Author": "Youyou"
"PostedDate": "5/23/2006"
"Tags": ["database", "nosql"]
"Body": "Je présente aujourd'hui une présentation!!."
```


- Une base de données couchdb est une collection à plat de ces documents

RESTFul API

REST est basé sur le protocole HTTP

- Lecture : GET /somedatabase/some_doc_id
- Ecriture / update : PUT

```
PUT /somedatabase/some_doc_id HTTP/1.0
Content-Length: 245
Content-Type: application/json

Body
```

- Créer : POST
- Supprimer : DELETE

Robuste

- N'écrase pas une donnée « commité »
- Si le serveur tombe, il faut juste redémarrer CouchDB → pas de « repair »
- On peut prendre des snapshots avec des simples cp
- Plusieurs niveaux de durabilité : Choix entre synchroniser à toutes les mises à jours ou à la demande

Vues

- Méthodes pour aggréger et requeter les documents de la base de donnée
- Les vues sont définies par des documents spéciaux les « designs documents »
- Les vues sont écrites en javascript.
- Pour maintenir des performances sur les vues, le moteur de vues maintient des indexes sous forme btree

MapReduce

- Introduit par Google
- 2 étape
  - Etape Map → itère sur une liste d'éléments indépendants et accomplit une opération spécifique sur chaque élément
    - function(doc){}
  - Etape Reduce → prend une liste et combine les éléments selon un algorithme particulier
    - fonction(keys, values, rereduce){}

Exemple d'une vue

```
{
  "_id":"_design/company",
  "_rev":"12345",
  "language": "javascript",
  "views":
  {
    "all": {
      "map": "function(doc) { if (doc.Type == 'customer')
emit(null, doc) }"
    },
    "by_lastname": {
      "map": "function(doc) { if (doc.Type == 'customer')
emit(doc.LastName, doc) }"
    },
    "total_purchases": {
      "map": "function(doc) { if (doc.Type == 'purchase')
emit(doc.Customer, doc.Amount) }",
      "reduce": "function(keys, values) { return sum(values) }"
    }
  }
}
```

Exemple de map/reduce

couchdb-python

- Côté client
  - couchdb.client → client pour s'interfacer avec des servers couchdb
  - couchdb.design → pour gérer les design documents
  - couchdb.mapping → fournit des mappings entre les document json et les objets python
- Coté serveur
  - couchdb.view → pour écrire des vues en python plutôt qu'en javascript

couchdb.client.Server

- Représentation d'un server CouchDB

```
>>> server = Server('http://localhost:5984/')
```

- Pour créer une base de données : create

```
>>> server.create('python-tests')
```

- Pour accéder à une base de données

```
>>>  server['mabase']
```

- Pour supprimer une base de données

```
>>> del server['mabase']
```

couchdb.client.Database

- Création d'un nouveau document

```
>>> server = couchdb.client.Server()
>>> db = server.create('test')
>>> doc_id,  doc_rev = db.save({'type' : 'contact', 'name': 'yo'})
```

- Récuperation d'un document

```
>>> db[doc_id]
<Document … >
```

- Un document est comme dictionnaire

```
>>> db[doc_id]['type']
'contact'
```

couchdb.client.ViewResults

- Représentation d'une vue (permanent ou temporaire)

```
>>> map_fun = '''function(doc) {emit([doc.type, doc.name],
    doc.name); }'''

>>> results = db.query(map_fun)
```

- En option, on peut envoyer tous les paramètres d'une vue

```
>>> db.query(map_fun, count = 10 , skip = 5, descending =
    true)
```

- On peut utiliser les slices python pour positionner des startkeys et les endkeys

```
>>> results[keystart:keyend]
```

Mapper les documents Couchdb

- Permet de mapper des structures json avec du python et vice versa

```
>>> Class Person(couchdb.mapping.Document):
…       name = couchdb.mapping.TextField()
…       age = couchdb.mapping.IntegerField()
…       modified   =
    couchdb.mapping.DateTimeField(default=datetime.now)
>>> person = Person(name='youyou', age = 32)
>>> person.store(db)
>>> Person.load(db, 'youyou').rev
....
```

couchdb.mapping.ViewField

- Pour associer une vue à un attribut d'une classe

```
class Person(Document):
    by_name = ViewField('people', '''\
...         function(doc) {
...             emit(doc.name, doc);
...         }''')

>>> Person.by_name(db, count=3)
```

couchdbkit

- Basé sur restkit , librairie http qui n'est pas basé sur urllib2 ou httplib
- couchdbkit.client → API client vers couchdb
- Mapping dynamique des documents
- Extension django
- couchdbkit.consumer → Ecoute les changements effectués sur la base de données
- couchdbkit.loaders → pousse des fichiers de vues sur couchdb

CouchApp

- Utilitaire de déploiement d'application pour couchDB
- Ecrit en python
- Crée un squellete d'application
- Génère du code à l'aide de macros
- Déploie les applications sur des serveurs CouchDB


Cassandra

Introduction

- Utiliser en production par Digg, FaceBook, Twitter, Reddit …
- Tolérant à la panne
- Décentraliser
- Sous contrôle
- Modèle de données efficient et efficace
- Elastique
- Durable

Modèle de données

- Colonne
  - La plus petite unité de données dans cassandra
  - C'est un triplet

```
{  // this is a column
    name: "mailAddress",
    value: "arin@example.com",
    timestamp: 123456789
}

Ou plus simplement

 emailAdress  : "arin@example.com"
```

Modèle de données

- Super Colonne
  - C'est un tuple qui a un nom et une valeur (comme la colonne)
  - Mais la valeur est une liste de colonnes

```
{   // this is a SuperColumn
    name: "homeAddress",
    // with an infinite list of Columns
    value: {
        // note the keys is the name of the Column
        street: {name: "street", value: "1234 x street", timestamp: 123456789},
        city: {name: "city", value: "san francisco", timestamp: 123456789},
        zip: {name: "zip", value: "94107", timestamp: 123456789},
    }
}
```

Famille de colonnes

- Une famille de colonnes est une structure qui contient une liste de lignes (comme les bases de données)

```
UserProfile = { // this is a ColumnFamily
    phatduckk: {   // this is the key to this Row inside the CF
        // now we have an infinite # of columns in this row
        username: "phatduckk",
        email: "phatduckk@example.com",
        phone: "(900) 976-6666"
    }, // end row
    ieure: {   // this is the key to another row in the CF
        // now we have another infinite # of columns in this row
        username: "ieure",
        email: "ieure@example.com",
        phone: "(888) 555-1212"
        age: "66",
        gender: "undecided"
    },
}
```

Modèle de données

- Super Famille de colonnes
  - Une Famille de colonne peut être super ou standard
  - Super c'est que les lignes contiennent des super colonnes aka cle : list(colonne)
  - Une ligne est une liste de colonnes ou de super colonnes identifiées par une clé

Super famille de colonne

```
AddressBook = { // this is a ColumnFamily of type Super
    phatduckk: {    // this is the key to this row inside the Super CF
        // the key here is the name of the owner of the address book

        // now we have an infinite # of super columns in this row
        // the keys inside the row are the names for the SuperColumns
        // each of these SuperColumns is an address book entry
        friend1: {street: "8th street", zip: "90210", city: "Beverley Hills", state: "CA"},

        // this is the address book entry for John in phatduckk's address book
        John: {street: "Howard street", zip: "94404", city: "FC", state: "CA"},
        Kim: {street: "X street", zip: "87876", city: "Balls", state: "VA"},
        Tod: {street: "Jerry street", zip: "54556", city: "Cartoon", state: "CO"},
        Bob: {street: "Q Blvd", zip: "24252", city: "Nowhere", state: "MN"},
        ...
        // we can have an infinite # of ScuperColumns (aka address book entries)
    }, // end row
    ieure: {     // this is the key to another row in the Super CF
        // all the address book entries for ieure
        joey: {street: "A ave", zip: "55485", city: "Hell", state: "NV"},
        William: {street: "Armpit Dr", zip: "93301", city: "Bakersfield", state: "CA"},
    },
}
```

L'ensemble des familles de colonnes et des supers familles de colonnes constituent un espace de clés (keyspace)

Modèle de données

- Les clés sont triées lorsqu'elle sont insérées dans le modèle
- Ce tri est respecté quand on recupère les éléments → le model doit être conçu en fonction de cela
- Les lignes sont triées par leur nom
- Les options de tri se font au niveau des familles de colonnes

Twissandra : un twitter like

- http://github.com/ericflo/twissandra
- Cassandra par l'exemple en python
- La façon de structurer les données doit être proche de la façon pour lesquelles on doit les récupérer pour les afficher

Twissandra : un twitter like

- User → colonne family
- La clé de la ligne est l'id de l'utilisateur

```
User = {
    'a4a70900-24e1-11df-8924-001ff3591711': {
        'id': 'a4a70900-24e1-11df-8924-001ff3591711',
        'username': 'ericflo',
        'password': '****',
    },
}
```

Twissandra : un twitter like

- Les amis et les adeptes sont indexés par le user id

```
Friends = {
    'a4a70900-24e1-11df-8924-001ff3591711': {
        # friend id: timestamp of when the friendship was added
        '10cf667c-24e2-11df-8924-001ff3591711': '1267413962580791',
        '343d5db2-24e2-11df-8924-001ff3591711': '1267413990076949',
        '3f22b5f6-24e2-11df-8924-001ff3591711': '1267414008133277',
    },
}
```

Twissandra : un twitter like

- Les tweets sont stokées comme les utilisateurs

```
Tweet = {
    '7561a442-24e2-11df-8924-001ff3591711': {
        'id': '89da3178-24e2-11df-8924-001ff3591711',
        'user_id': 'a4a70900-24e1-11df-8924-001ff3591711',
        'body': 'Trying out Twissandra. This is awesome!',
        '_ts': '1267414173047880',
    },
}
```

Twissandra : un twitter like

- Les tweets sont stokées comme les utilisateurs

```
Tweet = {
    '7561a442-24e2-11df-8924-001ff3591711': {
        'id': '89da3178-24e2-11df-8924-001ff3591711',
        'user_id': 'a4a70900-24e1-11df-8924-001ff3591711',
        'body': 'Trying out Twissandra. This is awesome!',
        '_ts': '1267414173047880',
    },
}
```

Twissandra : un twitter like

- La ligne de temps et la ligne des utilisateurs doivent afficher les tweets par ordre d'arrivée

```
Timeline = {
    'a4a70900-24e1-11df-8924-001ff3591711': {
        # timestamp of tweet: tweet id
        1267414247561777: '7561a442-24e2-11df-8924-001ff3591711',
        1267414277402340: 'f0c8d718-24e2-11df-8924-001ff3591711',
        1267414305866969: 'f9e6d804-24e2-11df-8924-001ff3591711',
        1267414319522925: '02ccb5ec-24e3-11df-8924-001ff3591711',
    },
}
Userline = {
    'a4a70900-24e1-11df-8924-001ff3591711': {
        # timestamp of tweet: tweet id
        1267414247561777: '7561a442-24e2-11df-8924-001ff3591711',
        1267414277402340: 'f0c8d718-24e2-11df-8924-001ff3591711',
        1267414305866969: 'f9e6d804-24e2-11df-8924-001ff3591711',
        1267414319522925: '02ccb5ec-24e3-11df-8924-001ff3591711',
    },
}
```

Twissandra : un twitter like

- Twissandra utilise pycassa , API de haut niveau pour accéder à cassandra
- Pycassa utilise thrift, un framework d'appel de procédure à distance
- Thrift gere 12 languages dont python

pycassa.columnfamily.ColumnFamily

- Opération sur la famille de colonne

```
>>> cf = pycassa.ColumnFamily(client, 'UserName')

## insertion d'une colonne
>>> cf.insert('user1', {'id': 'User'})
1261349837816957

>>> cf.get('foo')
{'column1': 'val1'}

## inserion de plusieurs colonne
>>> cf.insert('user2', {'id': 'User', 'name': 'Name User'})

## recuperation des lignes
>>> list(cf.get_range())
```

pycassa.columnfamilymap.ColumnFamilyMap

- Pour mapper des objets avec des colonnes

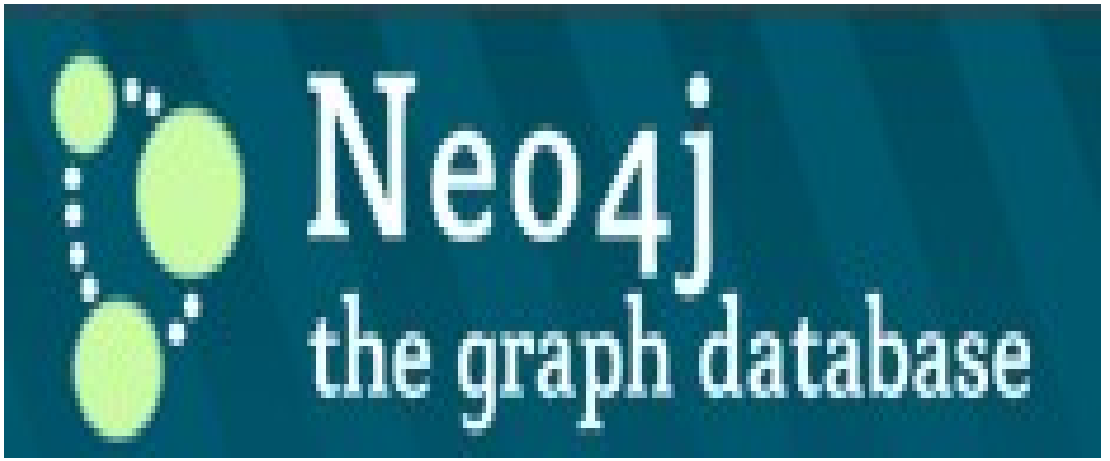
```
>>> class Test(object):
...     string_column       = pycassa.String(default='Your Default')
...     int_str_column      = pycassa.IntString(default=5)
...     float_str_column    = pycassa.FloatString(default=8.0)
...     float_column        = pycassa.Float64(default=0.0)
...     datetime_str_column = pycassa.DateTimeString() # default=None

>>> Test.objects = pycassa.ColumnFamilyMap(Test, cf)

>>> t = Test()
>>> t.key = 'maptest'
>>> t.string_column = 'string test'
>>> t.int_str_column = 18
>>> t.float_column = t.float_str_column = 35.8
>>> from datetime import datetime
>>> t.datetime_str_column = datetime.now()
>>> Test.objects.insert(t)
1261395560186855
```

Les autres librairies python pour cassandra

- Tragedy: http://github.com/enki/tragedy/
- Lazy Boy: http://github.com/digg/lazyboy/tree/master
- Telephus: http://github.com/driftx/Telephus/tree/master (Twisted)

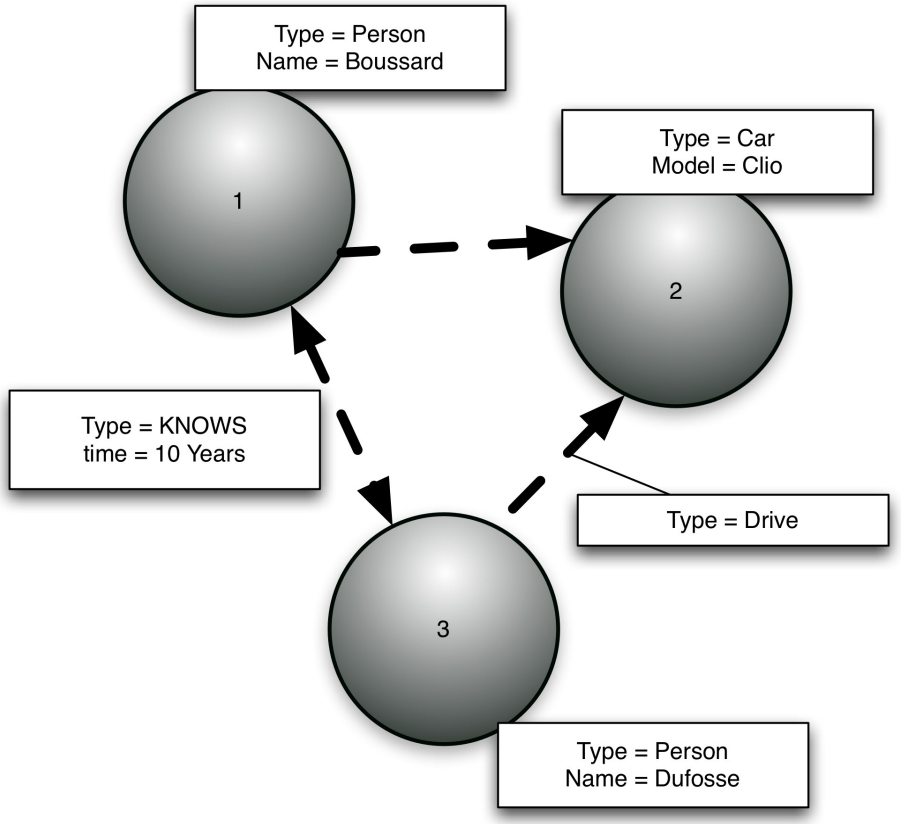

Neo4j
the graph database

Neo4J

- Modèle de données sous la forme de graphe
- Embarqué
- Stocké sur disque
- Scalable
- Framework de traversée
- API simple et pratique

Le modele de données : Un graphe orienté

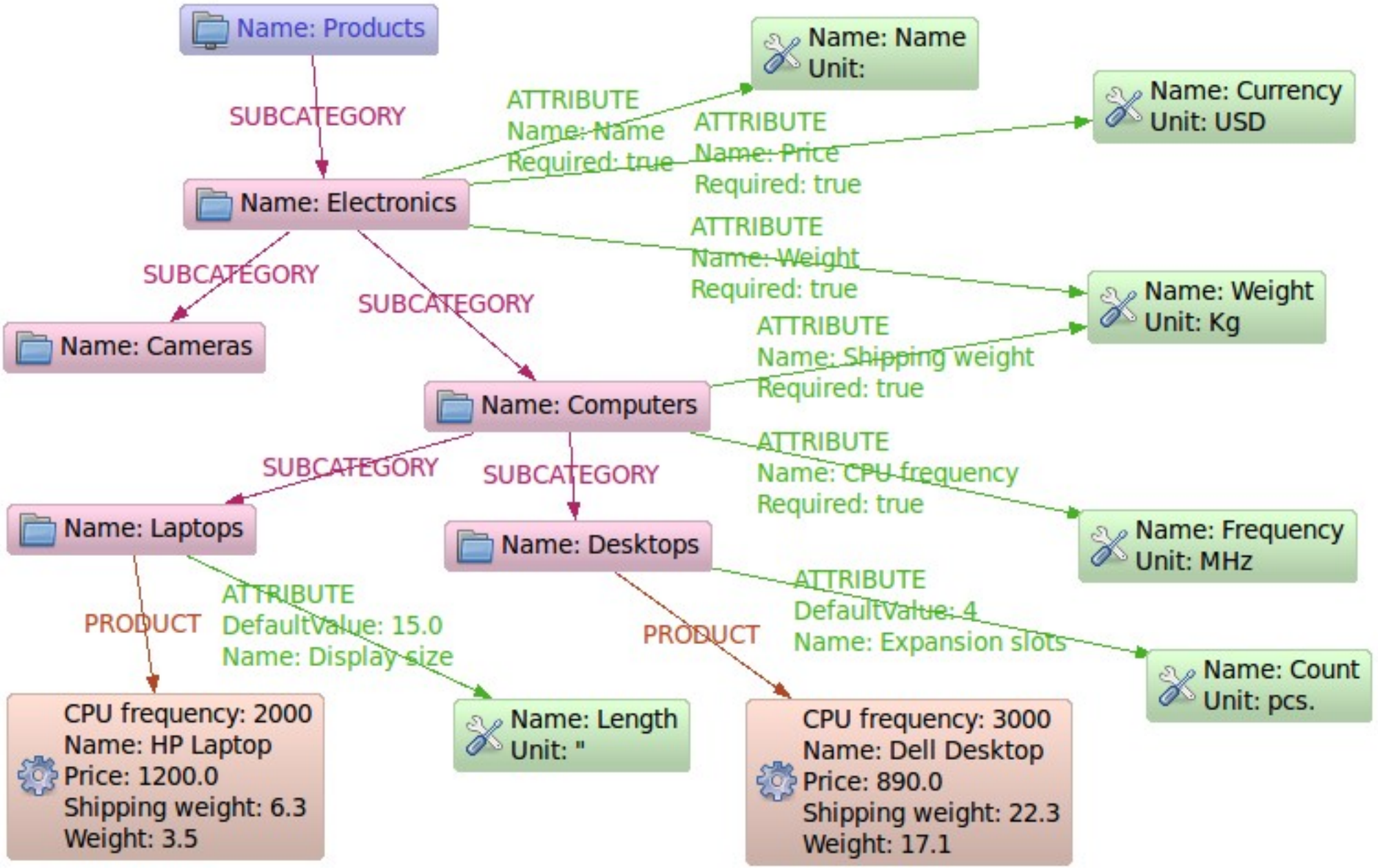
- Représentation sous la forme de:
  - Noeuds
  - Relations entre les noeuds
  - De propriétés (au niveau des relations et noeuds)



Exemple : modélisation des familles de produits

- Règle métier
  - Une famille peut être une sous-famille
  - Dans une famille il peut y avoir des produits
  - Chaque famille de produits peut avoir des propriétés

Exemple d'une instance de la base



Neo4j.py : binding python pour Neo4j

- Peut être utilisé indifférement avec
  - Jython
  - Cpython
- Ouvrir une base de données neo4j

```
graphdb = neo4j.GraphDatabase("/neo/db/path",
                              classpath=["/a/newer/kernel.jar"],
                              jvm="/usr/lib/jvm.so")
```

Neo4j.py : binding python pour Neo4j

- Créer une noeud

```
>>> mac_node = graphdb.node(Name='MacBook')
## ajouter des pro
>>> mac_node['Name']
'MacBook'
```

- Créer une relation entre les noeuds

```
>>> laptop_node = graphdb.node(Name='Laptop'')
## la mac est un produit de la famille des laptop_node
>>> laptop_node.PRODUCTS(mac_node)
```

Neo4j.py : binding python pour Neo4

- Traversals
  - Sont définis en créant une classe qui hérite de neo4j.Traversal
  - Un objet Traversal doit définir les attributs suivants:
    - Types : la liste des relations qui vont être traversées pendant la traversé
    - Order : l'ordre de traversée
    - Stop : la condition de d'arrêt
    - Returnable : La définition des noeuds qui vont être retournés

Neo4j.py : binding python pour Neo4

```
class SubCategoryProducts(neo4j.Traversal):
    "Traverser that yields all products in a category and its sub categories."
    types = [neo4j.Outgoing.SUBCATEGORY, neo4j.Outgoing.PRODUCT]
    def isReturnable(self, pos):
        if pos.is_start: return False
        return pos.last_relationship.type == 'PRODUCT'

## get products of  laptop_node
>>> p = [x for x in SubCategoryProducts(laptop_node)]
```

Références

- NoSQL
  - http://www.julianbrowne.com/article/viewer/brewers-cap-theorem
  - http://natishalom.typepad.com/nati_shaloms_blog/2009/12/the-common-principles-behind-the-nosql-alternatives.html
  - http://horicky.blogspot.com/2009/11/nosql-patterns.html

References

- CouchDB
  - http://www.unixgarden.com/index.php/web/couchdb-la-b
  - http://davidwatson.org/2008/02/python-couchdb-rocks.ht
  - http://wiki.apache.org/couchdb/Introduction_to_CouchDB
  - http://labs.mudynamics.com/wp-content/uploads/2009/0
  - http://horicky.blogspot.com/2008/10/couchdb-cluster.htm

Références

- Cassandra
  - http://www.slideshare.net/Eweaver/cassandra-presentati
  - http://spyced.blogspot.com/2009/03/why-i-like-cassandra.html
  - http://arin.me/blog/wtf-is-a-supercolumn-cassandra-data
  - http://wiki.apache.org/cassandra/ThriftExamples#Python
  - ttp://www.slideshare.net/stuhood/cassandra-talk-austin-jug
  - http://www.rackspacecloud.com/blog/2010/05/12/cassandra-by-example/

Références

- Neo4J
  - http://www.slideshare.net/thobe/persistent-graphs-in-pyt
  - http://python.mirocommunity.org/video/1597/pycon-2010
  - http://blog.neo4j.org/2010/03/modeling-categories-in-gra