Etude SGBD Cassandra

Réalisé par :
EL HANAFI SOUFIANE

Sommaire

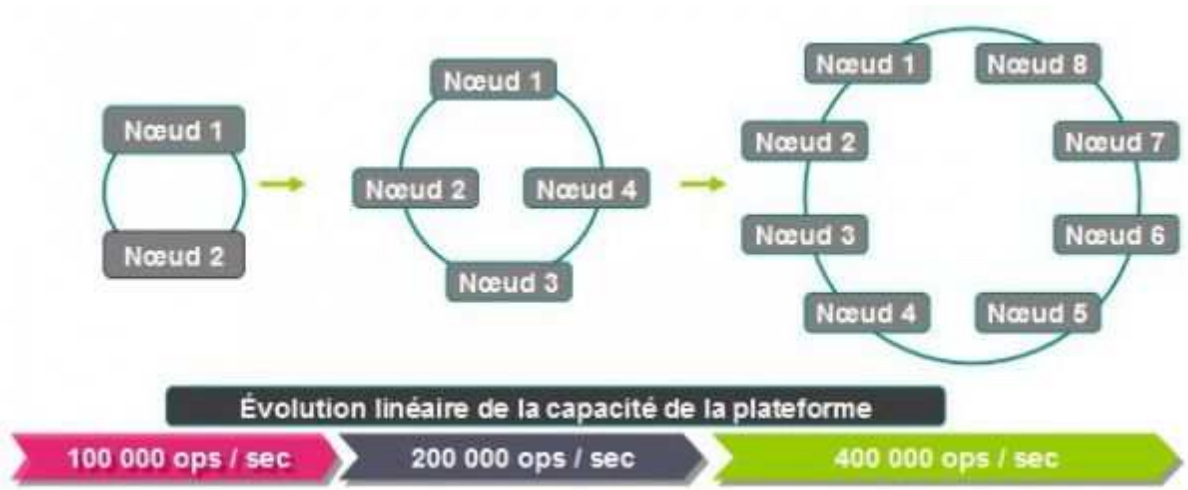
Introduction

Cassandra est une base de données distribuée NoSQL qui permet de stocker une grande quantité de données grâce à sa scalabilité horizontale. Ce terme correspond à la possibilité offerte par l'architecture Cassandra d'ajouter de nouvelles machines qui sont appelées des nœuds. Les machines utilisées sont généralement des machines dites *Community hardware* c'est-à-dire qu'elles correspondent au meilleur rapport/qualité prix.

Initialement Cassandra a été développée en interne par Facebook pour les besoins de sa messagerie interne . L'utilisation de Cassandra était limitée aux exigences de sa messagerie. Par conséquent de nombreuses fonctionnalités n'étaient pas implémentées puisqu'elles n'étaient pas nécessaires. En 2008, Facebook décide d'offrir Cassandra à la fondation Apache. Le projet est resté deux ans dans l'incubateur avant de devenir projet à part entière (dit "top-level projects") de la fondation Apache en 2010. Vous trouverez sur le site d'Apache (http://cassandra.apache.org/) .

1. Architecture de Cassandra

L'architecture Cassandra est basée sur un modèle distribué et décentralisé de type « peer-to-peer ». Tous les éléments d'une plateforme sont supposés être identiques du moins d'un point de vue client , Il n'y pas de notion de maitre, ni d'esclave, ni de processus qui aurait à sa charge la gestion, ni même de goulet d'étranglement au niveau de la partie réseau. L'ensemble des éléments ou nœuds est maintenu dans un état de service cohérent grâce à un protocole de communication interne dit « gossip ». Le but de l'architecture Cassandra est de garantir une haute disponibilité via une grande tolérance aux pannes (: les données d'un nœud (un nœud est une instance de Cassandra) sont automatiquement répliquées vers d'autres nœuds (différentes machines). Ainsi, si un nœud est hors service les données présentes sont disponibles à travers d'autres nœuds.) doublée d'une extensibilité (scalabilité) facile et flexible, le tout avec une promesse de hautes performances aussi bien à l'écriture qu'à la lecture. Le schéma ci-dessous inspiré de la documentation officielle de DataStax illustre le lien linéaire entre les capacités d'une plateforme Cassandra et le nombre de nœuds associés.



1.1 Les composants clés de Cassandra

Les composants clés de Cassandra sont les suivants:

- **Node:** c'est l'endroit où les données sont stockées.
- **Data center :** il s'agit d'une collection de nœuds associés.
- **Cluster**: un cluster est un composant qui contient un ou plusieurs centres de données.
- **Commit log:** Le journal de validation est un mécanisme de récupération de crash dans Cassandra , chaque opération d'écriture est écrite dans le journal de validation.
- **Mem-Table:** c'est une mémoire dans laquelle données vont être écrits après chaque Commit log
- **SSTable:** C'est un fichier disque dans lequel les données sont stocké a partir du Mem-table Lorsque son contenu atteint une valeur seuil.

1.2 Les opérations d'écriture :

Chaque activité d'écriture des nœuds est captée par les journaux de commit écrits dans les nœuds.Plus tard, les données seront capturées et stockées dans la Mem-Table.et quand la Mem-Table est pleine, les données seront écrites dans le fichier de données SStable. Toutes les écritures sont Partitionnées et répliquées automatiquement dans tout le cluster.

1.3 Les opérations de lecture :

Pour rappel, lorsqu'une requête de lecture d'une ligne est reçue par un nœud, la ligne doit être combinée de toutes les SSTables du nœud qui contiennent les colonnes de la ligne en question ainsi que de toutes les Memtables. Pour optimiser ce processus, Cassandra utilise une structure en mémoire appelée les bloom filter : chaque SSTable dispose d'un bloom filter associé qui est

utilisé pour vérifier s'il existe des données dans la ligne avant de faire une recherche (et ainsi de faire des entrées/sorties disque).

1.4 Les opérations de mise a jour :

Plusieurs colonnes peuvent être insérées en même temps. Ainsi, lorsque des colonnes sont insérées ou mises à jour dans une famille de colonnes, l'application cliente précise la clé de ligne pour identifier l'enregistrement à mettre à jour. La clé de la ligne peut donc être vue comme une clé primaire puisqu'elle doit être unique pour chaque ligne dans une famille de colonnes donnée. Cependant, à la différence d'une clé primaire, il est possible d'insérer des données à une clé primaire déjà présente, mais dans ce cas, Cassandra répondra comme une mise à jour (si elle n'existe pas, elle sera créée).
De plus, les colonnes sont écrasées si le timestamp d'une autre version de colonne est plus récent. Il est, cependant, à noter que le timestamp est fourni par le client. Aussi, ces derniers doivent disposer d'une horloge synchronisée par NTP (Network Time Protocol).

1.5 Les opérations de suppression :

Lorsqu'une ligne ou une colonne est supprimée dans Cassandra, il est important de comprendre les points ci-dessous :

- la suppression de la donnée du disque n'est pas immédiate (pour rappel, une donnée insérée est écrite dans la SSTable qui se trouve sur le disque). En effet, la SSTable étant immuable, un marqueur appelé tombstone est écrit pour renseigner le nouveau statut de la colonne. Les colonnes ainsi marquées persistent pendant un certain laps de temps

configurable puis sont réellement supprimées lors du processus *de compaction* ;

- une colonne supprimée peut réapparaître si la routine de réparation de nœud n'est pas exécutée. En effet, marquer une colonne d'un tombstone implique qu'un réplica qui était dans un état non atteignable lors de la suppression ne recevra l'information que lorsqu'il sera de nouveau dans un état joignable. Cependant, si un nœud est injoignable plus longtemps que le temps configuré de conservation des tombstones, alors le nœud peut complètement manquer la suppression et répliquer les données supprimées lors de son retour dans le cluster ;
- la clé de ligne pour une ligne supprimée apparait toujours dans la plage de résultats d'une requête. En effet, quand une ligne est supprimée dans Cassandra, ses colonnes correspondantes sont marquées par un tombstone. Ainsi, tant que les données marquées par un tombstone ne sont pas réellement supprimées par le processus de compaction, la ligne reste vide (i.e. sans colonne associée).

2 Modèle de données

2.1 cluster :

Dans l'écosystème Cassandra, le cluster est un groupement logique de plusieurs nœuds. Il est doté d'un nom pour assurer le cloisonnement d'un ensemble de nœuds dans un environnement multi-cluster. Un cluster peut être local à un « data center » ou réparti sur plusieurs. La communication inter nœuds y est possible grâce au protocole « gossip ». Ce dernier permet entre autres la détection des défaillances. En effet, l'ensemble des nœuds s'enregistre auprès d'un manager (Gossiper) responsable de la gestion de leurs états de service. Un cluster est lié à un «

keyspace » (équivalent du schéma de base de données relationnelle) et sa taille peut varier d'un moment à l'autre

2.2 keyspace

Keyspace est le conteneur le plus externe pour les données de Cassandra c'est un un regroupement de column families. Les attributs de base D'un Keyspace dans Cassandra sont:

- **Replication factor:** C'est le nombre de machines dans le cluster qui
  Recevra des copies des mêmes données
- **Replica placement strategy :** c'est la strategie a mettre en place pour la réplication de données dans l'anneau des noeuds , cet argument prend deux valeurs : **SimpleStrategy** si on utilise un seul data center , **network topology strategy** si on a plusieures centres .

The structure of a keyspace

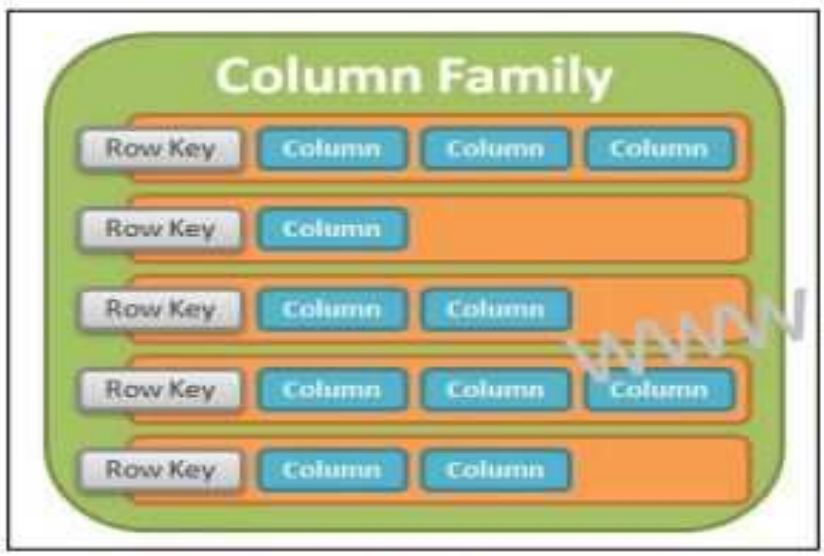
2.3 Column families

Une ColumnFamily est comparable à la notion de table dans le monde relationnel **sauf que le nombre et même (je ne suis pas sûr que ce soit la meilleure idée) les noms de colonnes peuvent varier d'une ligne à une autre.** Plus important et plus vraisemblable, le nombre de colonnes peut varier dans le temps (dans l'hypothèse que vos schémas vont évoluer dans le temps…), Cassandra n'impose pas de relations entre les familles

de colonnes au sens base de données relationnelle : il n'y a pas de clés étrangères et les jointures entre familles de colonnes ne sont pas supportées.

En fait, il y a deux types de familles de colonnes :

- **les familles de colonnes statiques :** elles utilisent un ensemble statique de noms de colonne et sont très similaires à une table d'une base de données relationnelle même si toutes les colonnes n'ont pas à être obligatoirement renseignées. À noter qu'en général, les métadonnées des colonnes sont définies, dans ce cas, pour chaque colonne ;
- **les familles de colonnes dynamiques :** elles permettent, par exemple, de précalculer un ensemble de résultats et de les stocker dans une même ligne afin de pouvoir y accéder plus tard. Chaque ligne correspond donc à un snapshot de données correspondant à une requête donnée. À noter que, pour une famille de colonnes dynamiques, plutôt que d'avoir des métadonnées pour chaque colonne, le type des informations par colonnes (tel que le comparateur et les validateurs) est défini par l'application lorsque la donnée est insérée.



The structure of a column family

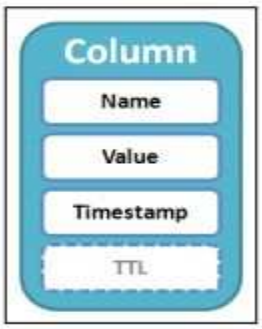
2.4 Column

Une colonne est la plus petite unité du modèle de données de Cassandra. C'est un triplet contenant un nom, une valeur et un timestamp. Ce dernier sert à déterminer la mise à jour la plus récente. La taille du nom peut être contenue jusqu'à 64 KO. La valeur quant à elle peut contenir 2 GO de données, la valeur n'est pas obligatoire. Son omission peut conduire à une amélioration de performance.
Concernant, le type de données exploitables par Cassandra, il existe deux notions :

- le ***validator*** qui est le type de données pour la valeur d'une colonne (ou la clé d'une ligne) ;
- le ***comparator*** qui est le type de données pour le nom d'une colonne.

Ci-dessous, la liste des valeurs possibles pour les validator et comparator (sauf pour le type CounterColumnType qui ne peut être utilisé que comme valeur de colonne) :

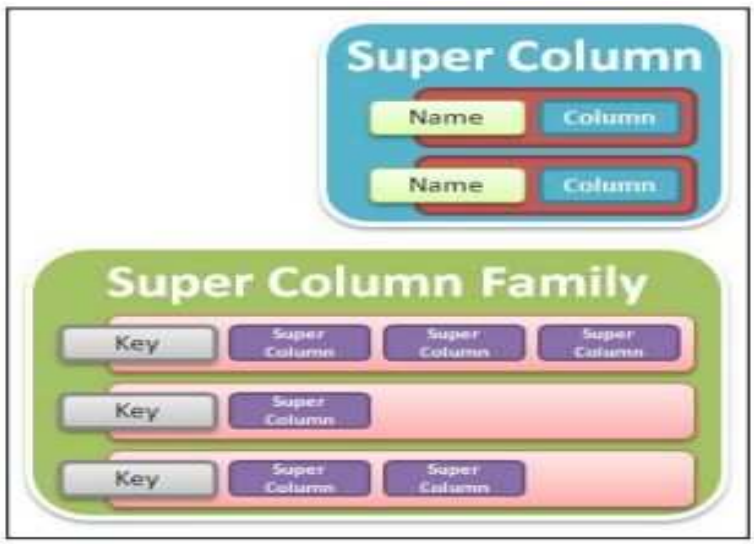
Type Interne	Nom CQL
BytesType	blob
AsciiType	ascii
UTF8Type	text, varchar
IntegerType	varint
Int32Type	int
LongType	bigint
UUIDType	uuid
TimeUUIDType	timeuuid
DateType	timestamp
BooleanType	boolean
FloatType	float
DoubleType	double
DecimalType	decimal
CounterColumnType	counter



The elements of a column

2.5 Super colonne

Une super colonne est une colonne spéciale, elle est aussi une paire clé-valeur. Mais une Super colonne stocke une carte de sous-colonnes. Généralement, les familles de colonnes sont stockées sur le disque dans des fichiers individuels. Par conséquent pour ,
Optimiser les performances, il est important de conserver les colonnes que vous souhaitez consulter Ensemble dans la même famille de colonnes, et une super colonne peut être utile ici.
Voici la structure d'une super colonne



The structure of a super column and a super column family

2.6 Comparaison entre la structure de la base de données relationnel et le modèle NoSQL orienté colonne .

model de base de données relationnel	model base de données NoSQL orienté colonne (cassandra)
Un schéma dans un modèle relationnel est fixé . Une fois que nous définissons certaines colonnes pour un Table, lors de l'insertion de données, dans chaque rangée Toutes les colonnes doivent être remplies au moins Avec une valeur nulle.	Cassandra, bien que la colonne families sont définies, les colonnes ne le sont pas. Vous pouvez ajouter librement n'importe quelle colonne à Toute famille de colonnes à tout moment.
traite des données structurées.	Cassandra traite des données non structuré.
Il a un schéma fixe.	Cassandra possède un schéma flexible.
La base de données est le conteneur le plus à l'extérieur	Keyspace est le conteneur le plus à l'extérieur
Les tablessont les entités d'une base de données.	Les tables ou les familles de colonnes sont l'entité D'un espace clé

La colonne représente les attributs d'un relation.	La colonne est une unité de stockage dans Cassandra

3 . Modélisation de données dans cassandra

3 .1 Introduction

Les règles de modélisation dans cassandra sont assez différentes des règles de modélisation dans le modèle relationnel qu'on utilise d'habitude . comme on a déja vue , les règles de conception dans le modèle relationnel ont pour bute de minimiser la duplication de données et minimiser le nombre d'écriture , par contre c'est régles la dans le schéma de cassandra sont pas un souci , elles représentent cependant des points de force , car la modélisation dans cassandra rime avec la denormalisation et la réplication , et Les lectures ont tendance à être plus coûteuses et sont beaucoup plus difficiles à régler , L'espace disque est généralement la ressource la moins chère (par rapport à la CPU, la mémoire, les IOP de disque ou le réseau), et Cassandra est architecte autour de ce fait , ainsi pour obtenir les lectures les plus efficaces, on doit souvent dupliquer les données et effectuer des écritures supplémentaires.

3.2 Application et explications

Dans notre cas on vas prendre comme exemple de modélisation une application web , ou les utilisateurs qui sont de types recruteurs lancent des offres d'emploies , et les utilisateurs normaux peuvent postuler aux offres , chaque utilisateur est caractérisé par son , prénom , username , mdp , ... , une ou plusieurs compétences , une ou plusieurs expériences ,une ou plusieurs formations , il peut maîtriser une ou plusieurs langues , et où les utilisateurs peuvent entrer en contact et envoyez des messages . dans la figure qui suit le diagrammes de classe qui

représente le modèle relationnel la base de données de cette application

3.2.1 Dictionnaire de données dans le modèle relationnel

Entités

Nom	Code
postuler	postuler
offre	offre
User	User
competences	competences
formations	formations
langues	langues
experiences	experiences
conversation	conversation
message	message

Informations sur les attributs des entités

Code	Type	Nom
id_user	int	identificateur de l'utilisateur
Nom_utilisateur	string	nom d'utilisateur
mdp	string	mot de passe

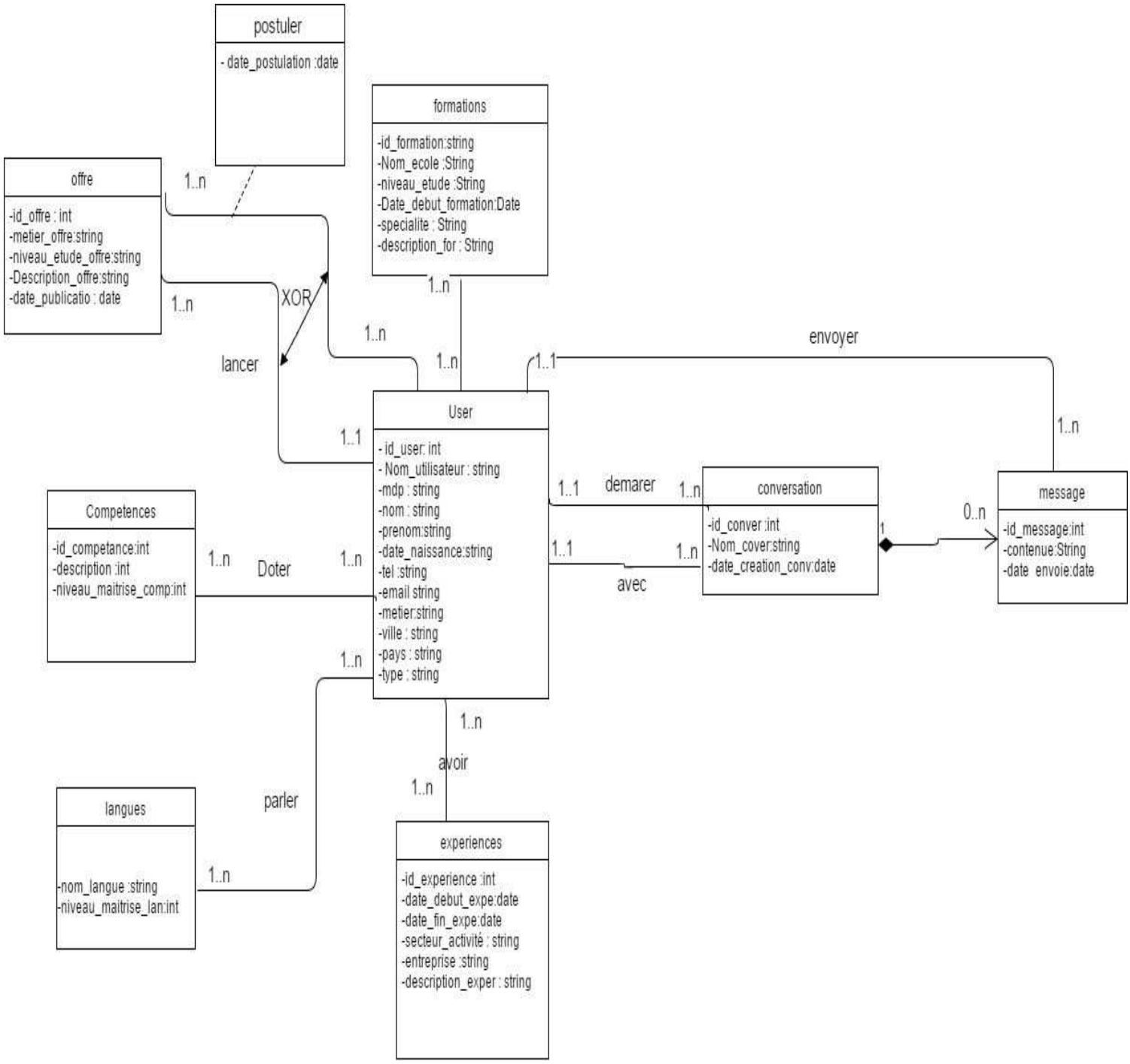
prenom	string	prénom
date_naissance	date	date de naissance
tel	string	telephone
email	string	email
metier	string	métier
ville	string	ville
pays	string	pays
type	string	type du user
id_formation	int	identificateur de la formation
Nom_ecole	string	nom de l'école
niveau_etude	string	niveau d'étude
Date_debut_formation	date	date de début de la formation
specialite	string	spécialité
description_for	string	description de la formation
id_experience		identificateur de l'expérience
date_debut_expe	date	date de début de l'expérience
date_fin_expe	date	date de fin de l'expérience

secteur_activité	string	secteur d’activité
entreprise	string	entreprise
description_exper	string	description de l'expérience

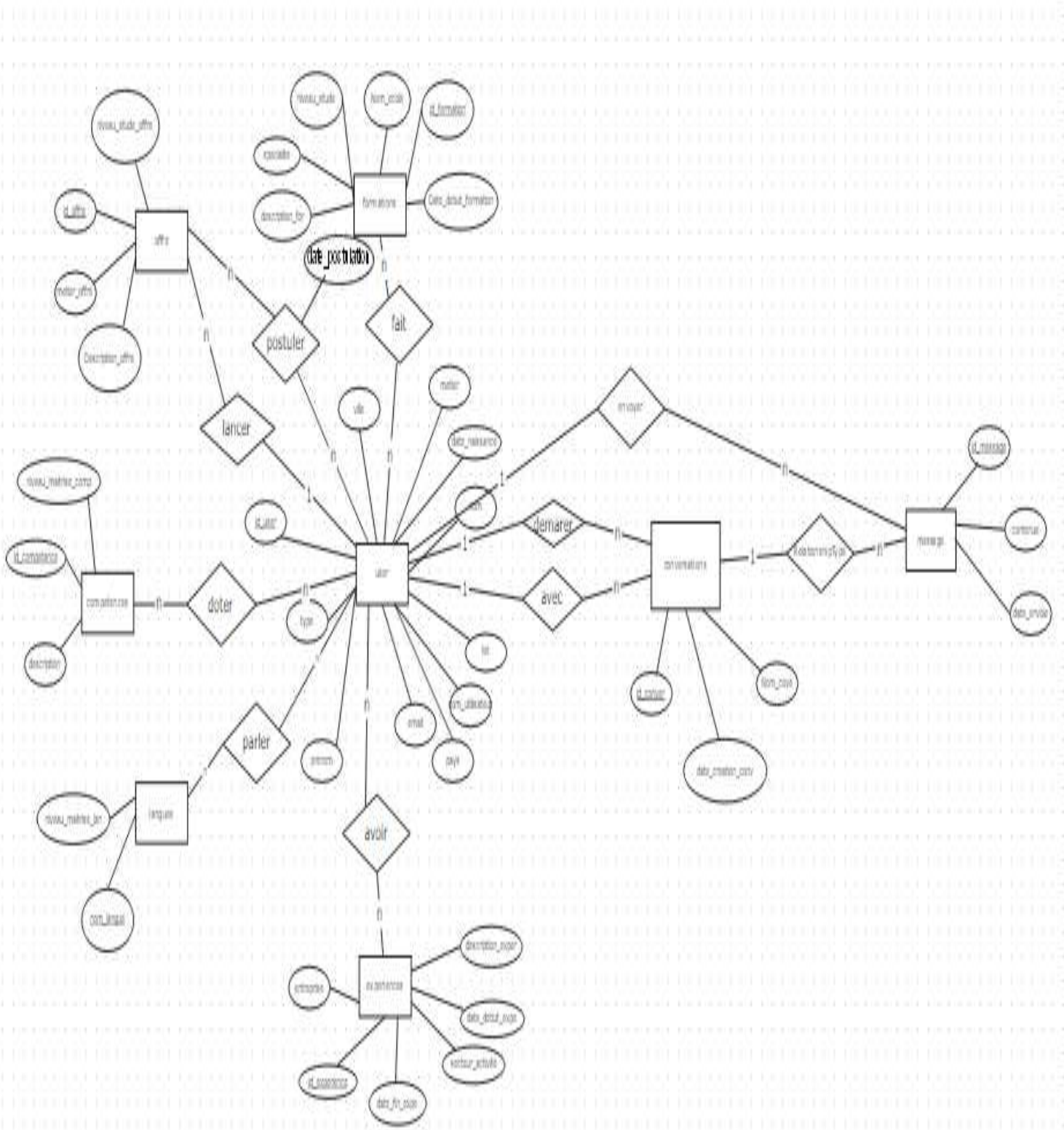
id_competance	int	identificateur de la compétence
description	string	description de la compétence
niveau_maitrise_comp	int	niveau de maitrise de la compétence
nom_langue	string	nom de la langue
niveau_maitrise_lan	int	niveau de maitrise de la langue
id_offre	int	identificateur de l’offre
metier_offre	string	métier de l’offre
niveau_etude_offre	string	niveau d'étude demandé pour cet offre
Description_offre	string	description de l’offre
date_publicatio	date	date de publication
date_postulation	date	date de postulation de la part d’un user
id_conver	int	identificateur de la conversation
nom_cover	string	nom de la conversation
date_creation_conv	int	date de création de la conversation
id_message	int	identificateur du message
contenue	string	contenue du message
date_envoie	date	date d’envoie du message

3.2.2 Modèle conceptuelle

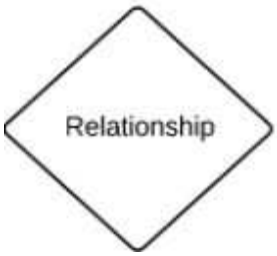
Modèle conceptuel d'une base de données relationel (UML)



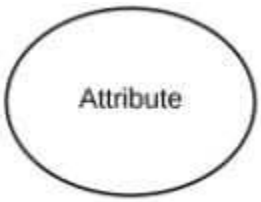
[Modèle conceptuel de données cassandra]



- dans le model conceptuel de cassandra les entités sont représenté avec des rectangles portant leurs noms .



- les relations sont représenté par un losange portant leurs nom .



- les attributs des entités sont représenté comme dans la figure à côté , si un attribut est une clé primaire de l'entité alors son nom doit être souligné .

3.2.3 Le modèle logique en cassandra

Notion de la clé composé et du clustering dans Cassandra

une clé composé est une clé qui contient deux ou plusieures colonnes qui déterminent le clustring (le regroupement)
la première partie de la clé primaire qu'on appelle par **la clé de partition** qui peut être une seul colonne ou composé de plusieures colonnes détermine quel nœud stocke les données. elle est responsable de la distribution de données dans les nœuds.

le reste de la clé détermine le clustering dans la partition . le principale rôle du clustering et de trier les données dans la partition.

Règles à suivre et à prendre en compte lors de la modélisation d'un schéma cassandra .

afin d'avoir un très bon modèle de donnée , il faut prendre en compte deux règles d'une importance majeur :

1 . Répartir les données uniformément autour du cluster

Comme on a déjà mentionné avant que la base de données NoSQL Cassandra est une base de donnée décentralisé c'est a dire que dans un cluster tous les nœuds sont égaux. Il n'y pas de notion de maître, ni d'esclave, alors si on souhaite que tous les nœuds du cluster aient à peu près la même quantité de données , Cassandra rend cela facile, mais ce n'est pas un fait donné, il faut choisir une bonne clé primaire composé d'une clé de répartition , pour une bonne répartition des donnés suivant notre propre cas d'utilisation .

2 . Minimiser le nombre de lecture des partitions .

Comme on a déjà vue dans l'introduction Les lectures ont tendance à être plus coûteuses et sont beaucoup plus difficiles à régler , c'est pour cela il faut minimiser le nombre de lecture de partition pour accéder a l'information quand recherche et pour s'y faire on doit recenser toutes les requêtes en lecture qu'on veut pouvoir exprimer et on construit les tables qui répondent à ce besoin.

Les notions des **clés étrangères et des jointures** n’existent pas dans le modèle logique de cassandra

Les clés et les jointures étrangères sont le produit de la normalisation dans un modèle de données relationnelles. Cassandra n'a ni clés étrangères, ni joint. Au lieu de cela, il encourage et exécute Mieux lorsque le modèle de données est dénormalisé. Les clés étrangères sont utilisées dans une base de données relationnelle pour maintenir l'intégrité référentielle qui définit la relation entre deux tableaux. Ils sont utilisés pour faire respecter les relations dans un modèle de données relationnelles tel que les les différents tables qui sont en relation et qu’on peut effectuer entre elle des jointures afin de répondre à une requête. Cassandra n'a pas le concept d'intégrité référentielle et Par conséquent, les jointures ne sont pas non plus permises .

Application

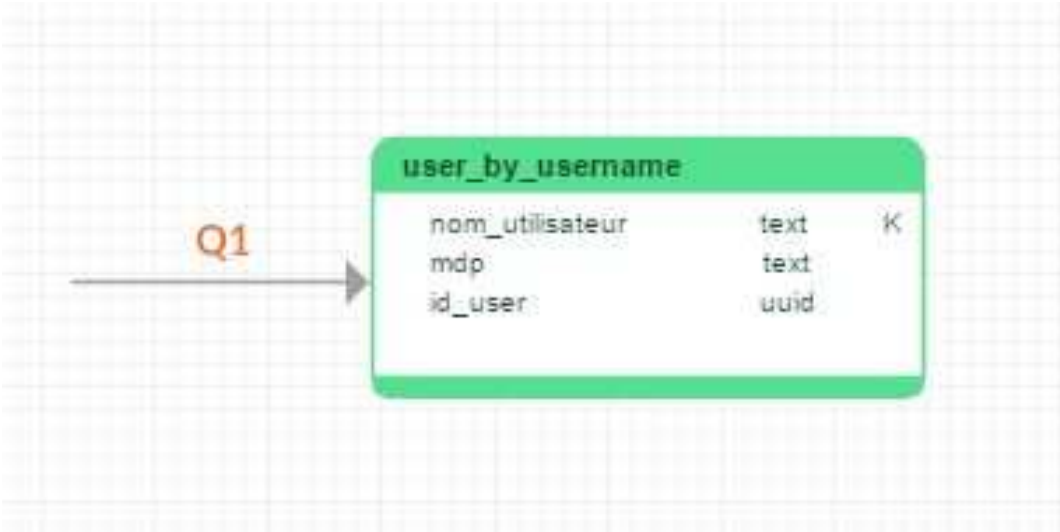
Dans notre cas on va élaborer juste les cas d’utilisation d’un user simple , les requêtes les plus souvent utiliser pour un simple user seront :

Q1 : connexion de l’utilisateur avec un nom d’utilisateur specifique
Q2 :lister toutes les offres par date de publication
Q3: stocker les postulation des offres
Q4 :lister les conversations d’un user depuis l’id_user
Q5 :lister les messages d’une conversation depuis id_conver
Q6 :voir le profil d’un utilisateur depuis son id_user

Requête Q1 :connexion de l’utilisateur avec un nom d’utilisateur spécifique

- attributs qu’on possède : user.id_user
- attributs qu’on recherche :user.mdp , user.id_user

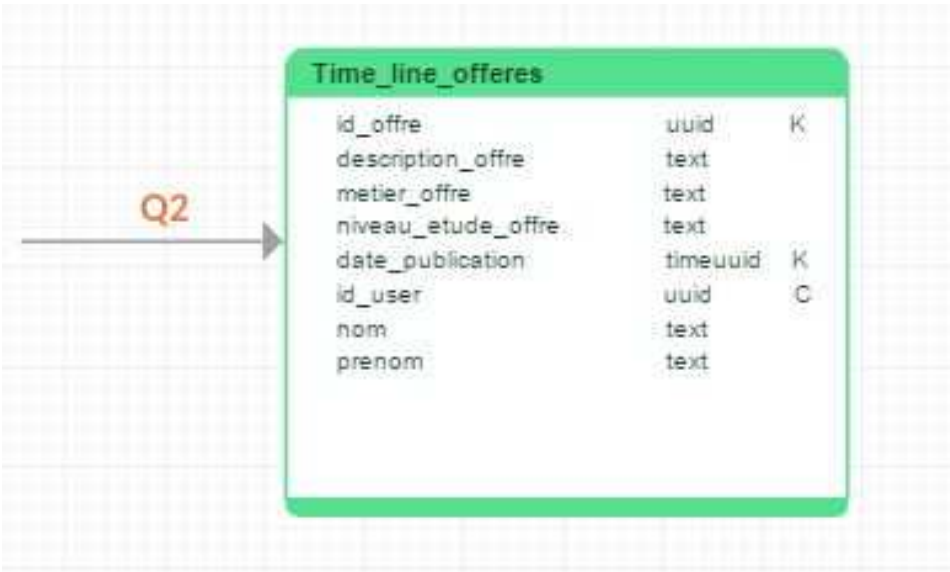
- model logique :



Requête Q2 :lister toutes les offres par date de publication

- attributs qu’on possède : user.id_user
- attributs qu’on recherche : id_offre , metier_offre , niveau_etude_offre , Description_offre , date_publication

- modèle logique:



Requête Q3 : stocker les postulation des offres

- attributs qu'on possède : id_offre
- attributs qu'on veut stocker : date_postulation , id_user , nom ,prenom , date_naissance , nom_utilisateur , tel , ville , mdp , email , metier , date_naissance
- modèle logique :

Requête Q4 : lister les conversations d'un user depuis l'id_user

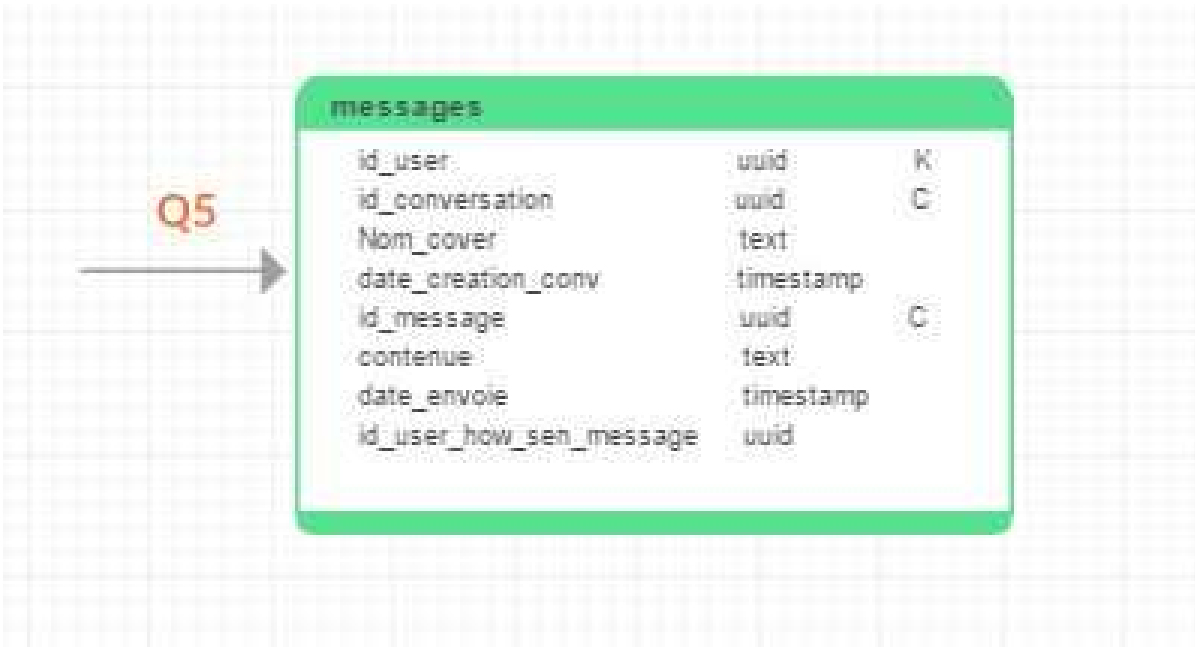
lister les conversations d'un user depuis l'id_user

- attributs qu'on possède : id_user
- attributs qu'on cherche :id_user , id_conversation , nom_cover , date_creation_conv , id_user_dest
- modèle logique :

Requête Q5 :lister les messages d'une conversation depuis id_conversation et id_user

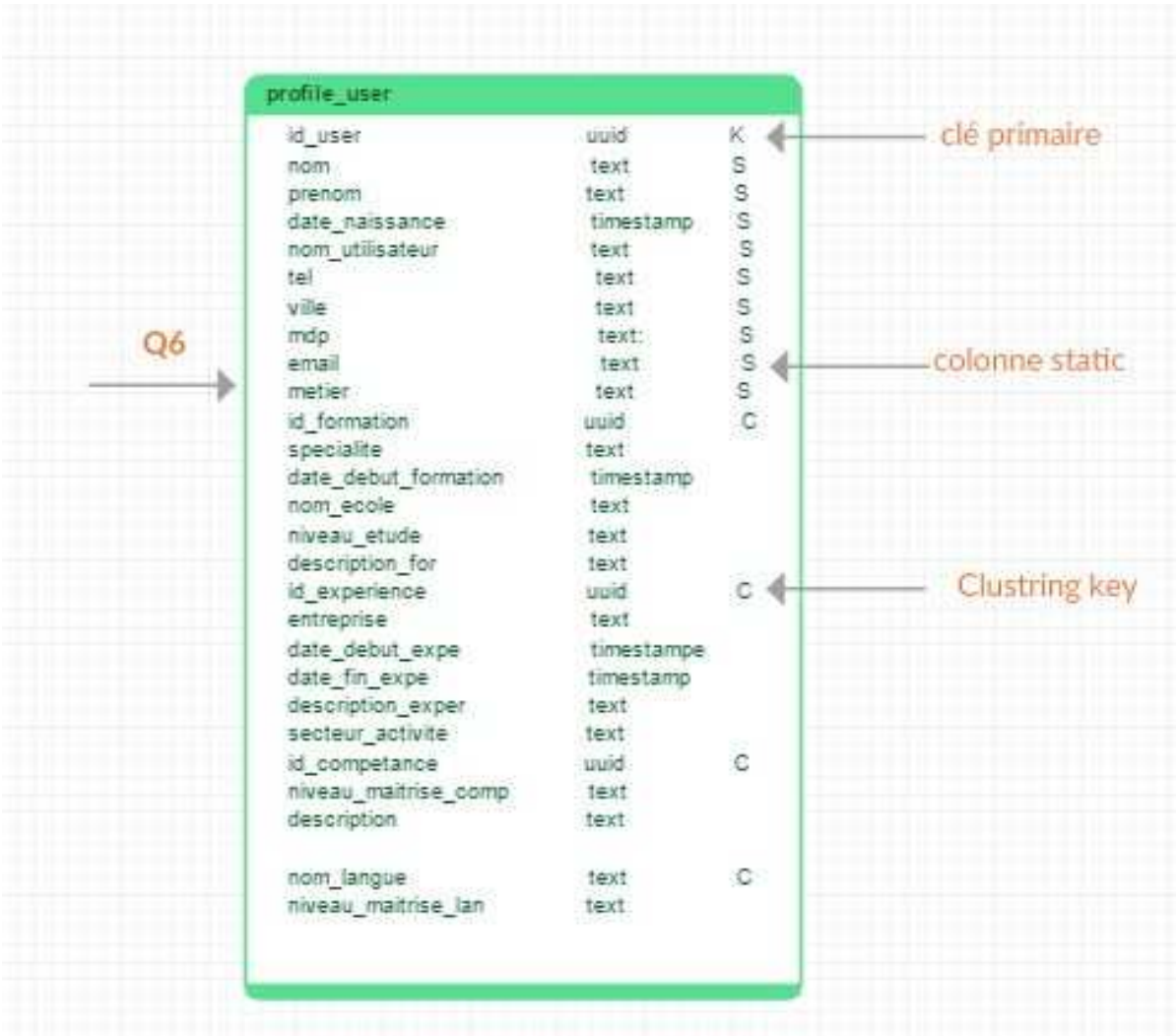
- attributs qu'on possède : id_user , id-conversation
- attributs qu'on cherche :Nom_cover , date_creation_conv,,id_message,contenue, date_envoie,id_user_how_sent_message

- modèle logique :



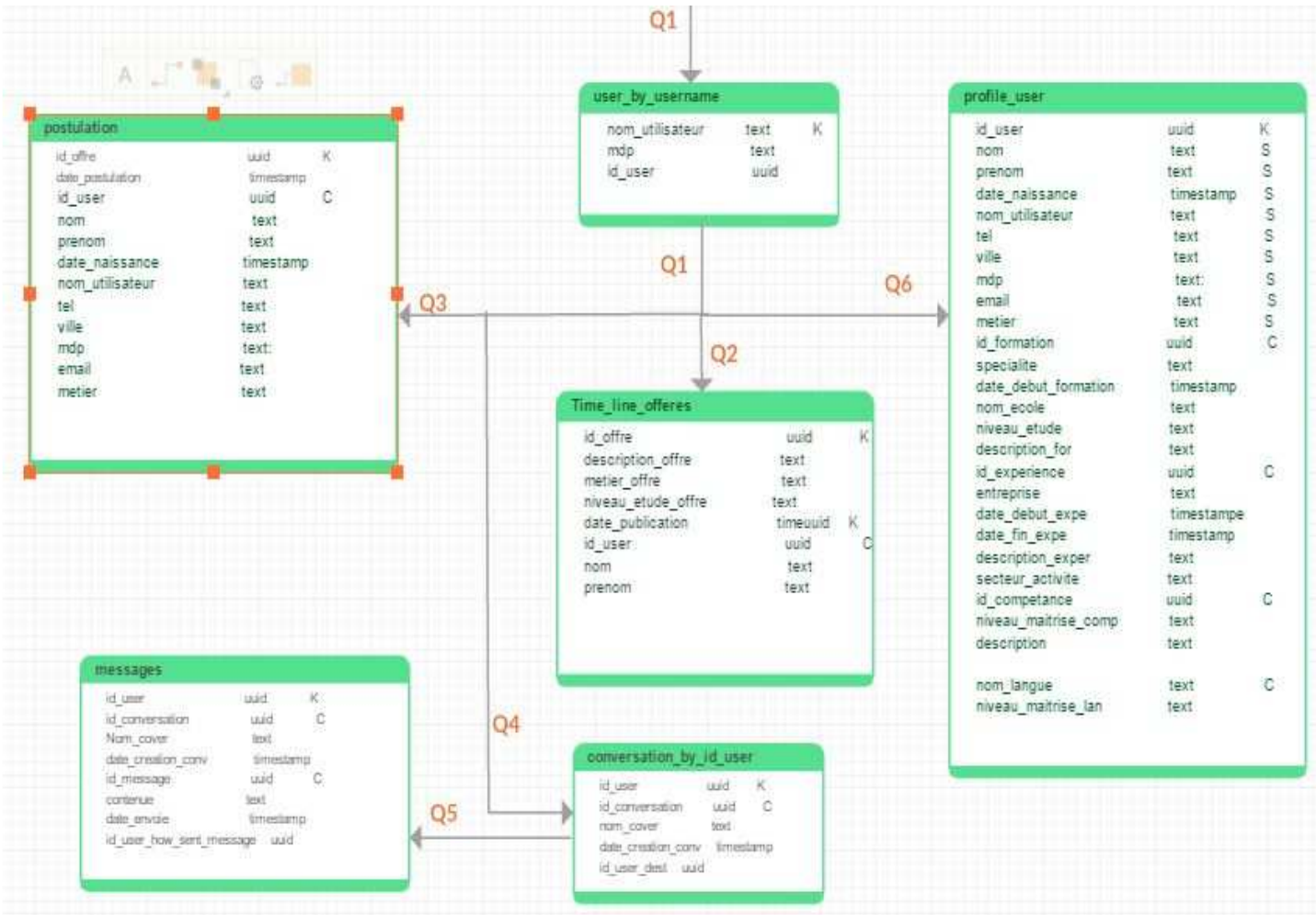
Requête Q6 :voir le profil d'un utilisateur depuis son id_user

- attributs qu'on possède : id_user
- attributs qu'on cherche : Nom_utilisateur ,mdp ,nom,prenom ,date_naissance ,tel ,email ,metier ,ville ,pays .
  id_formation ,Nom_ecole ,niveau_etude , specialite ,Nom_ecole .
  id_experience , date_debut_expe , date_fin_expe , secteur_activité ,entreprise,description_exper.
  nom_langue ,niveau_maitrise_lan .
  id_competance , description ,niveau_maitrise_comp
- modèle logique :



CHEBOTKO DIAGRAM

Le diagramme de chebotko est un diagramme qui permet de concevoir le modèle logique d'une base de données cassandra on se basant sur les requêtes les plus fréquemment utilisé et qui sont le produit de l'interaction de l'utilisateur avec notre application on les appelle par "The application workflows" , et qu'on chématise par des fléches dans notre exemple ce sont les flèches de transition entre une table et une autre(Q1 , Q2 ,Q3 ,Q4 ,Q5,Q6)



3.2.3 Modèle physique

Création d'un Keysapace

Tout d'abord on crée un keyspace qui est le conteneur le plus externe pour les données de Cassandra avec la syntaxe suivante

```
CREATE KEYSPACE IF NOT EXISTS cassandrademocql
WITH REPLICATION = { 'class' : 'SimpleStrategy', 'replication_factor' : 1 };
```

- **Replication factor:** C'est le nombre de machines dans le cluster qui
  Recevra des copies des mêmes données
- **Replica placement strategy :** c'est la strategie a mettre en place pour la réplication de données dans l'anneau des noeuds , cet argument prend deux valeurs : **SimpleStrategy** si on utilise un seul data center , **network topology strategy** si on a plusieures centres .

Création des tables

Création de la table profile_user

```
CREATE TABLE cassandrademocql.profile_user (
    id_user uuid,
    id_formation uuid,
    id_experience uuid,
    nom_langue text,
    id_competance uuid,
    date_debut_expe timestamp,
    date_debut_formation timestamp,
    date_fin_expe timestamp,
    date_naissance timestamp STATIC,
    description text,
    description_exper text,
    description_for text,
    email text STATIC,
    entreprise text,
    mdp text STATIC,
    metier text STATIC,
    niveau_etude text,
    niveau_maitrise_comp text,
    niveau_maitrise_lan text,
    nom text STATIC,
    nom_ecole text,
    nom_utilisateur text STATIC,
    prenom text STATIC,
    secteur_activite text,
    specialite text,
    tel text STATIC,
    ville text STATIC,
    PRIMARY KEY (id_user, id_formation, id_experience, nom_langue, id_competance)
);
```

Création de la table time_line_offeres

```
CREATE TABLE cassandrademocql.time_line_offeres (
    date_publication timeuuid,
    id_offre uuid,
    id_user uuid,
    description_offre text,
    metier_offre text,
    niveau_etude_offre text,
    nom text,
    prenom text,
    PRIMARY KEY (id_offre, id_user, description_offre)
) WITH CLUSTERING ORDER BY ( id_user ASC, description_offre DESC );
```

Création de la table find_user_by_nom_utilisateur

```
CREATE TABLE cassandrademocql.find_user_by_nom_utilisateur (
    nom_utilisateur text,
    mdp text,
    id_user uuid,
    PRIMARY KEY (nom_utilisateur)
);
```

Création de la table conversation_by_id_user

```
CREATE TABLE cassandrademocql.conversation_by_id_user (
    id_user uuid,
    id_conversation uuid,
    nom_cover text,
    date_creation_conv timestamp,
    id_user_dest uuid ,
    PRIMARY KEY (id_user, id_conversation, date_creation_conv)
) WITH CLUSTERING ORDER BY ( id_conversation ASC, date_creation_conv DESC );
```

Création de la table postulation

```
CREATE TABLE cassandrademocql.postulation (
    id_offre uuid,
    date_postulation timestamp,
    id_user uuid,
    nom text,
    prenom text,
    nom_utilisateur text,
    tel text,
    ville text,
    mdp text,
    email text,
    metier text,
    date_naissance timestamp,
    PRIMARY KEY (id_offre, id_user)
);
```

Création de la table message

```
CREATE TABLE cassandrademocql.message(
    id_user uuid,
    id_conversation uuid,
    id_message uuid,
    contenue text,
    date_creation_conv text,
    date_envoie text,
    id_user_how_sent_message text,
    nom_cover text,
    PRIMARY KEY (( id_user, id_conversation ), id_message)
) ;
```

Insertion dans les tables

dans les insertion on va se limiter sur la table profile_user

```
INSERT INTO cassandrademocql.profile_user
   (id_user , nom, prenom , nom_utilisateur ,mdp ,  tel , ville, email  , date_naissance
    ,id_experience , date_debut_expe , date_fin_expe ,description_exper ,metier ,secteur_activite ,
     entreprise
    , id_formation ,date_debut_formation , niveau_etude ,nom_ecole ,  specialite ,   description_for
    ,id_competance ,niveau_maitrise_comp  ,  description
     ,nom_langue , niveau_maitrise_lan  )
  VALUES( now() , 'soufiane' , 'El hanafi' , 'soufiane.elhanafi' , '123456789' , '0613484316' , 'Tanger'
    , 'soufiane8elhanafi@gmail.com'
   , '1995-01-04'
   , now() , '2014-07-01' , '2015-08-01' , ' developpement d une application android ' , 'developpeur'
   , 'informatique' , 'ESSCG'
   , now(),'2012-09-01' , 'bac+5' , 'Ensa Tanger', 'informatique', 'interessante'
    , now(), 'très bien' , 'java'
    , 'francais' , ' bien'
    );
```

après cet insert tout le profile de l'utilisateur a été créé dans une seule ligne qui contient , ces informations , formations , compétences …. et avec une clé id_user généré automatiquement a partir de la fonction uuid() qui pour valeur dans notre cas d'insertion b4883eb1-18fc-41a0-80cc-695d6434fc73.

si on effectue un select :

```
select * from cassandrademocql.profile_user ;
```

id_user	id_formation	id_experience	nom_langue	id_competance	date_naissance	email	mdp	metier	nom
b4883eb1-18fc...	ccbae1da-2d1...	ec8694e5-0cef...	francais	5d0d7f6f-ee41...	1995-01-04 00:...	soufiane8elha...	123456789	developpeur	soufiane

nom_utilisateur	prenom	tel	ville	date_debut_expe	date_debut_for...	date_fin_expe	description	description_ex...	description_for
soufiane.elhan...	El hanafi	0613484316	Tanger	2014-07-01 00:...	2012-08-31 23:...	2015-07-31 23:...	java	developpeme...	interessante

entreprise	niveau_etude	niveau_maitris...	niveau_maitris...	nom_ecole	secteur_activite	specialite
ESSCG	bac+5	trés bien	bien	Ensa Tanger	informatique	informatique

maintenant si nous voulons ajouter par exemple d'autres formations , tout sera stocké dans la même ligne et ce la grâce a la notion de …….. offerte par cassandra , prenons l'exemple de l'insertion d'une nouvelle compétence , une nouvelle langue par exemple pour l'utilisateur avec l' id_user : b4883eb1-18fc-41a0-80cc-695d6434fc73

```
INSERT INTO cassandrademocql.profile_user
    (id_user
    ,id_experience , date_debut_expe , date_fin_expe ,description_exper ,metier
    ,secteur_activite , entreprise
    , id_formation
    ,id_competance ,niveau_maitrise_comp  ,  description
     ,nom_langue , niveau_maitrise_lan  )
  VALUES( b4883eb1-18fc-41a0-80cc-695d6434fc73
   , uuid() , '2015-07-01' , '2016-08-01' , ' developpement d une application web avec php ' , 'developpeur' , 'informatique'
   , 'Delphi'
   , uuid()
   , uuid(), 'trés bien' , 'php'
    , 'anglais' , ' bien'
    );
```

si après on effectuent un select on spécifiant l'id_user par la valeur : b4883eb1-18fc-41a0-80cc-695d6434fc73 , on remarque que tous les informations sont stocké dans la même ligne la différence c'est qu'on a ajouté des nouvelles valeurs au colonnes

```
select * from cassandrademocql.profile_user where id_user = b4883eb1-18fc-41a0-80cc-695d6434fc73;
```

le résultat est le suivant :

id_user	id_formation	id_experience	nom_langue	id_competance	date_naissance	email	mdp	metier	nom
b4883eb1-18fc...	c6e0bdf3-85c7...	c00dc17d-114...	anglais	87e378c7-ff32...	1995-01-04 00:...	soufiane8elha...	123456789	developpeur	soufiane
b4883eb1-18fc...	ccbae1da-2d1...	ec8694e5-0cef...	francais	5d0d7f6f-ee41...	1995-01-04 00:...	soufiane8elha...	123456789	developpeur	soufiane

nom_utilisateur	prenom	tel	ville	date_debut_expe	date_debut_for...	date_fin_expe	description	description_ex...	description_for
soufiane.elhan...	El hanafi	0613484316	Tanger	2015-07-01 00:...	<<null>>	2016-07-31 23:...	php	developpeme...	<<null>>
soufiane.elhan...	El hanafi	0613484316	Tanger	2014-07-01 00:...	2012-08-31 23:...	2015-07-31 23:...	java	developpeme...	interessante

entreprise	niveau_etude	niveau_maitris...	niveau_maitris...	nom_ecole	secteur_activite	specialite
Delphi	<<null>>	trés bien	bien	<<null>>	informatique	<<null>>
ESSCG	bac+5	trés bien	bien	Ensa Tanger	informatique	informatique

Les mises a jour :

Mise a jour d’un Keysapace

La modification d'un KEYSPACE permet de paramétrer certaines options comme la stratégie de placement ou le facteur de réplication. Le renommage d'un KEYSPACE n'est pas encore supporté.

```
ALTER  KEYSPACE | SCHEMA  keyspace_name
WITH REPLICATION = map
|  WITH DURABLE_WRITES =  true | false
AND  DURABLE_WRITES =  true | false
```

Mise à jour d’une table

Synopsis

```
ALTER TABLE keyspace_name. table_name instruction;
```

Parameters

instruction is:

```
( TYPE   cql_type
| ADD column_name cql_type
| DROP column_name
|  RENAME column_name TO column_name
|  WITH property [ AND property ] . . . } )
```

comme application on pourra ajouter une colonne à la table profile_user qu'on va appeler type qui détermine si l'utilisateur et un simple utilisateur ou un recruteur , j'ai oublié de l'ajouter dans le modèle

```
ALTER TABLE cassandrademocql.profile_user ADD type text;
```

Mise à jour d'une colonne dans une ligne

```
UPDATE keyspace_name.table_name
USING option AND option
SET assignment, assignment, ...
WHERE row_specification
IF column_name = literal AND column_name = literal . . .
IF EXISTS
```

comme application on change le mot de passe pour l'utilisateur avec le id_user : b4883eb1-18fc-41a0-80cc-695d6434fc73

```
UPDATE cassandrademocql.profile_user
  SET mdp = '987456321'
  WHERE id_user=b4883eb1-18fc-41a0-80cc-695d6434fc73 ;
```

Les suppression

Supprimer un Keysapace

```
DROP KEYSPACE cassandrademocql ;
```

Supprimer d’une Table

```
DROP TABLE IF EXISTS cassandrademocql.profile_user  ;
```

Supprimer une ligne d’une table

```
delete from cassandrademocql.profile_user where id_user = b4883eb1-18fc-41a0-80cc-695d6434fc73;
```

Conclusion

Cassandra reste une base de données NoSQL est très rapide pour manipuler un volume important de données. Elle permet d'avoir des schémas de données flexible grâce à sa représentation en colonnes. De plus son architecture lui permet d'évoluer sans problème dans un environnement distribué, elle intègre des mécanismes de réplication de données et la possibilité de mettre en cluster plusieurs serveurs Cassandra, elle privilégie toujours pour ses accès disque, les accès séquentiels aux accès aléatoires, ce qui permet d'éviter une partie des latences importantes dues aux mécaniques des disques durs. Ainsi, lors d'une écriture, les données ne sont pas écrites directement sur disque mais stockées dans une table en mémoire ; un ajout dans un commitlog se comportant en append-only (et donc de manière séquentielle) permet d'assurer la durabilité de l'écriture. Lorsque la table en mémoire est pleine, elle est écrite sur le disque , mais La principale limitation de Cassandra concerne les tailles des colonnes et des super-colonnes, toutes les données pour une valeur de clé, doivent tenir sur le disque d'une seule machine. Parce que la valeur des clés seules détermine les noeuds responsable de la réplication des données, la quantité de données associées à une clé a cette limitation.