ESILV LÉONARD DE VINCI | ÉCOLE D'INGÉNIEURS PARIS-LA DÉFENSE

Dénormalisation

Développement d'application Cloud

Nicolas Picard
Henri Duhamel
Johan Van Der Slooten
Maxence Clusaz

Octobre 2019

1 Schéma Entité/Association original

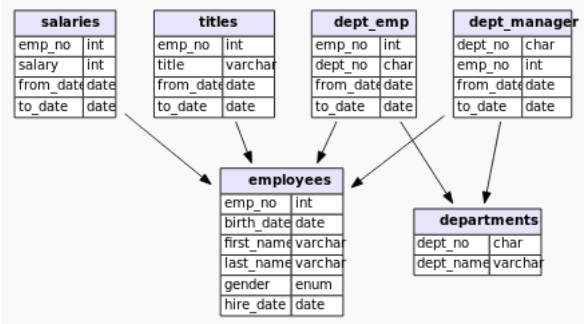

Figure 1: Schéma relationnel de la base de données MySQL

D'après le schéma de la base de données MySQL dont nous disposons, nous obtenons un schéma Entité/Association qui est le suivant.

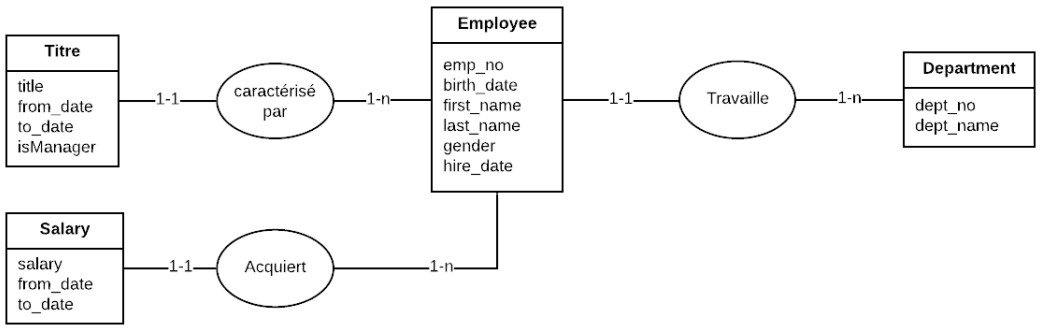

Figure 2: Schéma Entité/Association original

On sait également qu'on dispose de 8 serveurs (VMs sur Azure), 300024 employés dont 24 managers, 9 départements, 2844047 salaires et 443308 titres, qui nous serviront pour nos calculs.
On réserve un serveur de configuration ainsi qu'un autre pour le routeur mongos, ce qui nous donne 6 serveurs à utiliser comme shards. On prend le cas où un département peut être sur 2 shards. Lors de la dénormalisation n°2 on a ajouté un dixième département pour les retraités / licenciés, qu'on nommera "Retired".
Comme nous l'avons vu en cours, nous devons réduire au maximum les coûts de communication réseau lors des requêtes de l'application à la base de données. Compte tenu des données existantes et du faible nombre de relations entre les 6 tables, il est préférable de tout regrouper au sein d'une unique collection. De plus, la clé la plus utilisée dans le schéma relationnel est le numéro d'employé ce qui permet ainsi de placer l'employé au centre de notre schéma et ainsi de créer une collection d'employés. Selon nous, il est possible de réaliser 2 modèles qui sont ceux qui se situent ci-dessous.

2 Schéma Entité/Association - Dénormalisation n°1

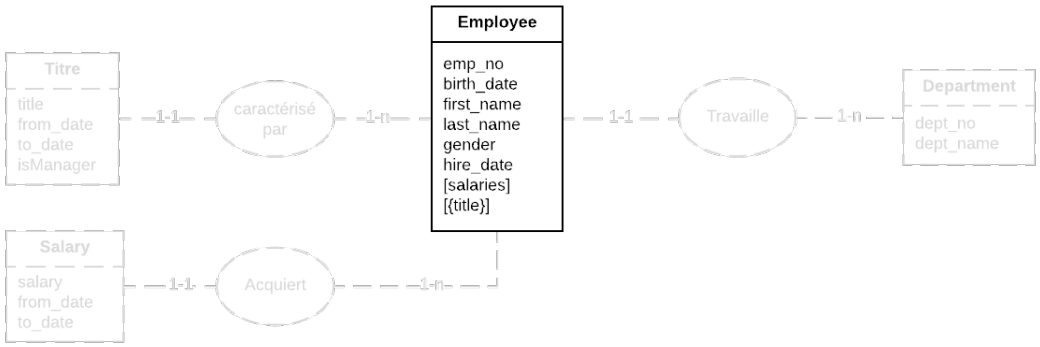

Figure 3: Schéma Entité/Association - Dénormalisation n°1

En effet, selon notre cas d'usage, les données relatives aux employés seront celles qui seront interrogées le plus fréquemment. Nous avons choisi de regrouper les salaires au sein d'un tableau, et les titres au sein d'un tableau "jobs" également. Ce tableau d'objets titre est légèrement différent de ce que l'on peut trouver au sein de la table titre. Nous avons choisi *emp_no* comme clé de sharding, ce qui nous permet de regrouper les employés par numéro d'employé.

2.1 Calculs

R1 - Obtenir sa fiche de paie d'une année choisie en connaissant son département

- Aller : 6 - On interroge les 6 serveurs
- Retour : 1 - Seul le document de l'employé est retourné
- Total : 7

R2 - Obtenir les employés travaillant dans son département

- Aller : 6 - On interroge les 6 serveurs
- Retour : 33336 - Tous les documents des employés du département sont retournés (9 départements)
- Total : 33342

R3 - Obtenir les managers d'un département

- Aller : 6 - On interroge les 6 serveurs
- Retour : 3 - Seul les documents des managers du département sont retournés
- Total : 9

R4 - Obtenir le nom et le titre des derniers employés arrivés dans son département (par volume, par date)

- Aller : 6 - On interroge les 6 serveurs
- Retour : 33336 - Les documents des derniers employés du départememnt sont retournés (on prend le cas où le volume ou la date renseigné sont maximals)
- Total : 33342

R5 - Obtenir la paie moyenne (ou la plus petite et la plus grande) par département si c'est un homme/femme/tranche d'âge

- Aller : 300024 - On interroge tous les employés pour les comparer entre eux
- Retour : 1 - La moyenne ou le plus petit/grand salaire est retourné
- Total : 300025

R6 - Obtenir tous les numéros d'employés qui ont le même titre

- Aller : 6 - On interroge les 6 serveurs
- Retour : 42861 - On a 7 différents titres, donc 300024/7 documents retournés
- Total : 42867

Un document employé sous cette dénormalisation aurait ainsi la forme suivante (tous les salaires ne sont pas affichés par souci d'espace):

```
{
    "emp_no" : 10010,
    "birth_date": "1963-06-01",
    "first_name": "Duangkaew",
    "last_name": "Piveteau",
    "gender": "F",
    "hire_date": "1989-08-24",
    "salaries": [
        50365,
        50953,
        53052
    ],
    "jobs": [
        {
            "title": "Engineer",
            "dept": "d004",
            "dept_name": "Production",
            "from_date": "1996-11-24",
            "to_date": "2000-06-26",
            "isManager": false
        },
        {
          "title": "Engineer",
          "dept": "d006",
          "dept_name": "Quality Management",
          "from_date": "2000-06-26",
          "to_date": "9999-01-01",
          "isManager": false
        }
    ]
}
```

Figure 4: Modèle JSON d'un employé - Dénormalisation 1

3 Schéma Entité/Association - Dénormalisation n°2

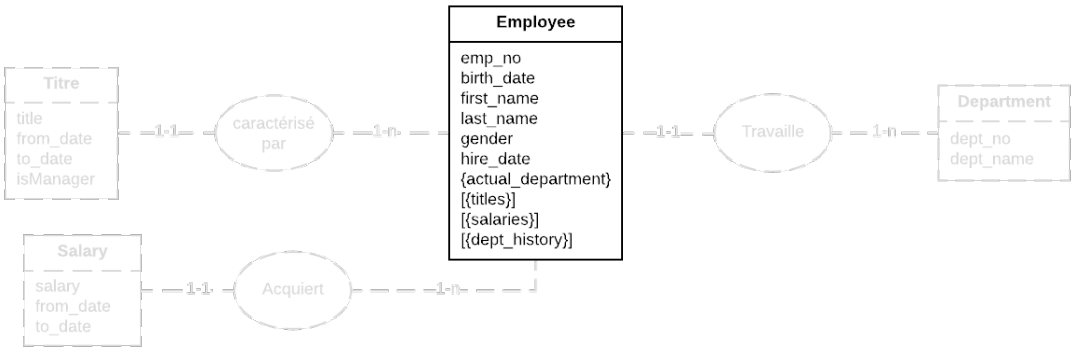

Figure 5: Schéma Entité/Association - Dénormalisation n°2

Dans cette dénormalisation, nous avons choisi de regrouper les titres au sein d'un tableau d'objets qui reprendraient les caractéristiques de leur table originale, ainsi qu'une nouvelle propriété booléenne qui indiquerait si, lorsque la personne a occupé ce poste, celle-ci était manager ou non.
Les salaires on également été reportés au sein d'un tableau "salaries" également. Ce tableau d'objets reprend les mêmes caractéristiques que celles de la table salaries.
On dispose également d'un objet *actual_ department* qui correspond au département dans laquelle la personne travaille en ce moment. Cela nous permet ainsi de définir le numéro de département comme clé de sharding, et ce, sans pour autant avoir 2 collections. Cela va nous permettre de regrouper les employés actuels par département et ainsi limiter le nombre de requêtes necéssaires par la suite. En effet, depuis la version 4.2 de MongoDB, la valeur d'une clé de sharding peut être modifiée, ce qui ne sera donc pas problématique lorsqu'une personne change de département, ce qui, de plus, est assez rare. De même, lorsqu'une personne partira à la retraite, il nous suffira de modifier ces valeurs par des valeurs par défaut, telles que "Retired" et un numéro de département unique tel que "d000". (cf. il est obligatoire d'avoir une valeur dans une clé de sharding)
Pour finir, nous avons choisi de rajouter un tableau *dept_ history*, qui indiquerait dans quels départements une personne a travaillé par le passé, ainsi que les dates de début et de fin de ce poste.

3.1 Calculs

R1 - Obtenir sa fiche de paie pour une année

- Aller : 2 - On interroge les serveurs contenant le département de l'employé
- Retour : Seul le document contenant l'employé est renvoyé
- Total : 3 (43% de la dénormalisation n°1)

R2 - Obtenir les employés travaillant dans un département spécifique

- Aller : 2 - On interroge les serveurs contenant le département de l'employé
- Retour : 30003 - Tous les documents des employés du département sont retournés (10 départements)
- Total : 30005 (90% de la dénormalisation n°1)

R3 - Obtenir les managers d'un département spécifique

- Aller : 2 - On interroge les serveurs contenant le département de l'employé
- Retour : 3 Les documents des managers du départememnt sont retournés
- Total : 5 (33% de la dénormalisation n°1)

R4 - Obtenir le nom et le titre des derniers employés arrivés dans un département spécifique

- Aller : 2 - On interroge les serveurs contenant le département de l'employé
- Retour : 30003 - Les documents des derniers employés du départememnt sont retournés (on prend le cas où le volume ou la date renseigné sont maximals)
- Total : 30005 (90% de la dénormalisation n°1)

R5 - Obtenir la paie moyenne/maximum/minimum par département si c'est un homme/une femme/ par tranche d'âge

- Aller : 30003 - On interroge tous les employés du département pour les comparer entre eux
- Retour : 1 - La moyenne ou le plus petit/grand salaire est retourné
- Total : 30004 (semblable à la dénormalisation n°1)

R6 - Obtenir tous les numéros d'employés qui ont le même titre

- Aller : 6 - On interroge les 6 serveurs
- Retour : 42861 - On a 7 différents titres, donc 300024/7 documents retournés
- Total : 42867 (identique à la dénormalisation n°1)

Un document employé sous cette dénormalisation reprendrait ainsi cette forme (tous les salaires ne sont pas affichés par souci d'espace):


```
{
   "emp_no":110344,
   "birth_date":"1961-09-07",
   "first_name":"Rosine",
   "last_name":"Cools",
   "gender":"F",
   "hire_date":"1985-11-22",
   "titles":[
      {
         "title":"Senior Engineer",
         "from_date":"1985-11-22",
         "to_date":"1988-09-09",
         "isManager":false
      },
      {
         "title":"Manager",
         "from_date":"1988-09-09",
         "to_date":"1992-08-02",
         "isManager":true
      },
      {
         "title":"Technique Leader",
         "from_date":"1992-08-02",
         "to_date":"9999-01-01",
         "isManager":false
      }
   ],
   "salaries":[
      {
         "salary":50365,
         "from_date":"1985-11-22",
         "to_date":"1986-11-22"
      },
      {
         "salary":50953,
         "from_date":"1986-11-22",
         "to_date":"1987-11-22"
      },
      {
         "salary":53052,
         "from_date":"1987-11-22",
         "to_date":"1988-11-22"
      }
   ],
   "actual_department":{
      "dept_no":"d006",
      "dept_name":"Quality Management",
      "from_date":"1987-12-24",
      "to_date":"9999-01-01"
   },
   "dept_history":[
      {
         "dept_name":"Production",
         "from_date":"1985-11-22",
         "to_date":"1987-12-24",
         "dept_no":"d004"
      }
   ]
}
```


Figure 6: Modèle JSON d'un employé - Dénormalisation 2

4 Conclusion

La dénormalisation n°1 possède de nombreux avantages, parmi lesquels on y trouve une modélisation qui est proche de ce qu'on pourrait modéliser humainement parlant. En effet, un "job" comprend un titre, on y est manager ou non, d'une certaine date à une autre, plus ou moins indéterminée, et ce, dans un département donné. On pourrait également y rajouter un salaire, cependant les salaires sont liés à une année dans notre base originale et non à un poste en particulier, ce qui fausserait les données.
Malheureusement, celle-ci possède certains inconvénients dont un qui est la concordance des dates entres salaires et emplois. En effet, on pourrait croire que le salaire est lié au poste mais le salaire est uniquement fixé sur une période, qui va du jour et mois de la date d'embauche d'une année à cette même date de l'année suivante. En modélisant un tableau de salaires, on gagne en efficacité puisque l'ajout et la suppression de valeurs sont simples, cependant il y a une perte d'information.

La dénormalisation n°2 est plus proche du schéma original qu'on pourrait retrouver dans notre base de données MySQL puisqu'il n'y a pas de perte d'information. Les informations concernant le statut de manager se trouve dans la table *dept_ manager* , et pour simplifier notre schéma, nous avons décidé d'en faire un booléen, lié au titre. En effet, on peut être manager même si notre titre n'est pas manager, on peut être Technique Leader tout en manageant une équipe par exemple. Ceci nous permet ainsi de nous débarasser des tables *dept_ emp* et *dept_ manager*. Le tableau de salaires reprend bien les informations initiales puisqu'on peut ainsi retrouver son salaire d'une année et ce, à une date très précise. Enfin, *actual_ department* nous permet de réliser une clé de sharding sur le département actuel d'un employé et ainsi de limiter les coûts et le nombre de requêtes générées en groupant les employés par département actuel. Le tableau *dept_ history* indique les départements dans lesquels une personne a travaillé.

Nous avons choisi de continuer avec la dénormalisation n°2 car celle-ci ne provoque aucune perte d'information et permet d'avoir une représentation assez fidèle de la réalité. De plus, les calculs que nous avons réalisé nous permettent de confirmer que les coûts réseaux seront bien moindres qu'en utilisant la dénormalisation n°1.