NoSQL
Normalisation / Dénormalisation

NoSQL

Normalisation / Dénormalisation

- la modélisation traditionnelle d'une base de données relationnelle a un objectif de normalisation qui vise à éviter à la fois toute redondance et toute perte d'information
- la redondance est évitée en découpant les données et en les stockant indépendamment les unes des autres dans des tables séparées
- la perte d'information est évitée en utilisant un système de référencement basé sur les clés primaires et clés étrangères
- les données sont contraintes par un schéma qui impose des règles sur le contenu de la base

NoSQL

Normalisation / Dénormalisation

- il n'y a aucune hiérarchie dans la représentation des entités
- utilisons des Films, des Réalisateurs, des Acteurs et des Pays auxquels appartiennent réalisateurs et acteurs
- une entité comme Pays, qui peut être considérée comme secondaire, a droit à sa table dédiée, tout comme l'entité Films qui peut être considérée comme essentielle
- on ne tient pas compte en relationnel de l'importance respective des entités représentées
- la distribution des données dans plusieurs tables est compensée par la capacité de SQL à effectuer des jointures qui exploitent le plus souvent le système de référencement (clé primaire, clé étrangère) pour associer des lignes stockées séparément

NoSQL

Normalisation / Dénormalisation

Table Acteurs

```
create table Acteurs
    (idActeur integer not null,
     nom varchar (30) not null,
     prenom varchar (30) not null,
     anneeNaiss integer,
    primary key (idActeur),
    unique (nom, prenom)
```

Table Réalisateurs

```
create table Realisateurs
    (idReal integer not null,
     nomReal varchar (30) not null,
     prenomReal varchar (30) not null,
     anneeNaissReal integer,
     primary key (idReal),
    unique (nomReal, prenomReal)
```

NoSQL

Normalisation / Dénormalisation

Table Films

```
create table Films
    (idFilm integer not null,
     titre varchar (50) not null,
     annee integer not null,
     idReal integer not null,
    genre varchar (20) not null,
    resume varchar (255),
    codePays varchar (4),
    primary key (idFilm),
    foreign key (idReal) references Realisateurs)
```

Table Rôles

```
create table Roles
    (idFilm integer not null,
     idActeur integer not null,
     nomRole varchar (30),
     primary key (idActeur, idFilm),
     foreign key (idFilm) references Films,
     foreign key (idActeur) references Acteurs)
```

NoSQL

Documents et Collections

- en conception relationnelle, on ne trouve que des valeurs dites atomiques, non décomposables
- par exemple, il ne peut y avoir qu'un seul genre pour un film
- si ce n'est pas le cas, il faut (processus de normalisation) créer une table des genres et la lier à la table des films
- cette nécessité de distribuer les données dans plusieurs tables est une lourdeur souvent reprochée à la modélisation relationnelle
- avec un document structuré, il est très facile de représenter les genres comme un tableau de valeurs, ce qui casse la première règle de normalisation

NoSQL

Documents et Collections

Films

```
{
 "titre": "Pulp fiction",
 "annee": "1994",
 "genre": ["Action", "Policier", "Comédie"]
 "pays": "USA"
}
```

NoSQL

Documents et Collections

Il est également facile de représenter une table par une collection de documents structurés

Voici la table des acteurs en notation JSON :

acteur: {"id": 11, "nom": "Travolta", "prénom": "John"},

acteur: {"id": 27, "nom": "Willis", "prénom": "Bruce"},

acteur: {"id": 37, "nom": "Tarantino", "prénom": "Quentin"},

acteur: {"id": 167, "nom": "De Niro", "prénom": "Robert"},

acteur: {"id": 168, "nom": "Grier", "prénom": "Pam"}

NoSQL

Documents et Collections

- on peut donc encoder une base relationnelle sous la forme de documents structurés
- chaque document peut être plus complexe structurellement qu'une ligne dans une table relationnelle
- une telle représentation, pour des données régulières, n'est pas du tout efficace à cause de la redondance de l'auto-description : à chaque fois on répète le nom des clés alors qu'on pourrait les factoriser sous forme de schéma relationnel et les représenter indépendamment, ce que fait un système relationnel classique

Dans une modélisation relationnelle :

- les films et les acteurs sont séparés dans 2 tables distinctes
- chaque film est lié à son réalisateur par une clé étrangère

En NoSQL, grâce à l'imbrication, on peut adopter cette représentation :

```
"titre": "Pulp fiction",
"annee": "1994",
"genre": "Action",
"pays": "USA",
"realisateur": {
    "nomReal": "Tarantino",
    "prenomReal": "Quentin",
    "anneeNaissReal": "1963"
}
```

- On imbrique un objet dans un autre
- On utilise un document unique et structuré
- On oublie le référencement par clés primaires / clés étrangères
- Ceci est remplacé par l'imbrication qui associe physiquement les entités films et réalisateurs

Film


```
{
 "titre": "Pulp fiction", "année": "1994", "genre": "Action", "pays": "USA",

        "Réalisateur": {
                "nomReal": "Tarantino", "prénomReal": "Quentin", "anneeNaissReal": "1963"
        }

        "Acteurs": {
                {"prénom": "John", "nom": "Travolta", "anneeNaiss": "1954", "rôle": "Vincent Vega"},
                {"prénom": "Bruce", "nom": "Willis", " anneeNaiss": "1955", "rôle": "Butch Coolidge" },
                {" prénom": "Quentin", "nom": "Tarantino", "anneeNaiss": "1963", "rôle": "Jimmy Dimmick"}
        }
}
```

Avantage 1 : plus besoin de jointures

- il est inutile de faire des jointures pour reconstituer l'information puisqu'elle n'est plus dispersée dans plusieurs tables

Film


```
{
 "titre": "Pulp fiction", "année": "1994", "genre": "Action", "pays": "USA",
        "Réalisateur": {
                "nomReal": "Tarantino", "prénomReal": "Quentin", "anneeNaissReal": "1963"
        }
        "Acteurs": {
                {"prénom": "John", "nom": "Travolta", "anneeNaiss": "1954", "rôle": "Vincent Vega"},
                {"prénom": "Bruce", "nom": "Willis", " anneeNaiss": "1955", "rôle": "Butch Coolidge" },
                {" prénom": "Quentin", "nom": "Tarantino", "anneeNaiss": "1963", "rôle": "Jimmy Dimmick"}
        }
}
```

Avantage 2 : plus besoin de transactions

- une seule écriture du document suffit pour créer toutes les données du film Pulp fiction
- en relationnel, il faut écrire 1 fois dans la table Films, 3 fois dans la table Acteurs, 3 fois dans la table Rôles, …

Film

```
{
 "titre": "Pulp fiction", "année": "1994", "genre": "Action", "pays": "USA",
        "Réalisateur": {
                "nomReal": "Tarantino", "prénomReal": "Quentin", "anneeNaissReal": "1963"
        }
        "Acteurs": {
                {"prénom": "John", "nom": "Travolta", "anneeNaiss": "1954", "rôle": "Vincent Vega"},
                {"prénom": "Bruce", "nom": "Willis", " anneeNaiss": "1955", "rôle": "Butch Coolidge" },
                {" prénom": "Quentin", "nom": "Tarantino", "anneeNaiss": "1963", "rôle": "Jimmy Dimmick"}
        }
}
```

Avantage 3 : la lecture

- **une seule lecture suffit** pour récupérer toutes les informations relatives à un film

Film


```
{
 "titre": "Pulp fiction", "année": "1994", "genre": "Action", "pays": "USA",
          "Réalisateur": {
                    "nomReal": "Tarantino", "prénomReal": "Quentin", "anneeNaissReal": "1963"
          }
          "Acteurs": {
                    {"prénom": "John", "nom": "Travolta", "anneeNaiss": "1954", "rôle": "Vincent Vega"},
                    {"prénom": "Bruce", "nom": "Willis", " anneeNaiss": "1955", "rôle": "Butch Coolidge" },
                    {" prénom": "Quentin", "nom": "Tarantino", "anneeNaiss": "1963", "rôle": "Jimmy Dimmick"}
          }
}
```

Avantage 4 : les documents sont autonomes

- 1 document par film
- il est facile de déplacer les documents pour les répartir au mieux dans un système distribué

Film


```
{
 "titre": "Pulp fiction", "année": "1994", "genre": "Action", "pays": "USA",
          "Réalisateur": {
                    "nomReal": "Tarantino", "prénomReal": "Quentin", "anneeNaissReal": "1963"
          }
          "Acteurs": {
                    {"prénom": "John", "nom": "Travolta", "anneeNaiss": "1954", "rôle": "Vincent Vega"},
                    {"prénom": "Bruce", "nom": "Willis", " anneeNaiss": "1955", "rôle": "Butch Coolidge" },
                    {" prénom": "Quentin", "nom": "Tarantino", "anneeNaiss": "1963", "rôle": "Jimmy Dimmick"}
          }
}
```

Inconvénient 1 : hiérarchisation des accès

- la représentation des films et des acteurs n'est pas symétrique
- les films sont à la racine des documents, les réalisateurs et les acteurs sont imbriqués
- l'accès aux films est privilégié : l'accès aux réalisateurs et aux acteurs passe par les films
- ce n'est pas le cas en relationnel où toutes les informations sont au même niveau

Film


```
{
 "titre": "Pulp fiction", "année": "1994", "genre": "Action", "pays": "USA",
        "Réalisateur": {
                "nomReal": "Tarantino", "prénomReal": "Quentin", "anneeNaissReal": "1963"
        }
        "Acteurs": {
                {"prénom": "John", "nom": "Travolta", "anneeNaiss": "1954", "rôle": "Vincent Vega"},
                {"prénom": "Bruce", "nom": "Willis", " anneeNaiss": "1955", "rôle": "Butch Coolidge" },
                {" prénom": "Quentin", "nom": "Tarantino", "anneeNaiss": "1963", "rôle": "Jimmy Dimmick"}
        }
`
```

Inconvénient 2 : perte d'autonomie des entités

- il n'est plus possible de représenter un réalisateur si on ne connaît pas au moins un de ses films
- inversement, en supprimant un film, on peut supprimer définitivement les données d'un acteur qui n'aurait tourné que dans ce film là

Film


```
{
 "titre": "Pulp fiction", "année": "1994", "genre": "Action", "pays": "USA",
        "Réalisateur": {
                "nomReal": "Tarantino", "prénomReal": "Quentin", "anneeNaissReal": "1963"
        }
        "Acteurs": {
                {"prénom": "John", "nom": "Travolta", "anneeNaiss": "1954", "rôle": "Vincent Vega"},
                {"prénom": "Bruce", "nom": "Willis", " anneeNaiss": "1955", "rôle": "Butch Coolidge" },
                {" prénom": "Quentin", "nom": "Tarantino", "anneeNaiss": "1963", "rôle": "Jimmy Dimmick"}
        }
}
```

Inconvénient 3 : redondance

- la même information est présente plusieurs fois
- Quentin Tarantino sera représenté autant de fois qu'il a tourné de films (ou fait l'acteur)
- il sera plus difficile de faire les mises à jour et suppressions

Film


```
{
 "titre": "Pulp fiction", "année": "1994", "genre": "Action", "pays": "USA",
        "Réalisateur": {
                "nomReal": "Tarantino", "prénomReal": "Quentin", "anneeNaissReal": "1963"
        }
        "Acteurs": {
                {"prénom": "John", "nom": "Travolta", "anneeNaiss": "1954", "rôle": "Vincent Vega"},
                {"prénom": "Bruce", "nom": "Willis", " anneeNaiss": "1955", "rôle": "Butch Coolidge" },
                {" prénom": "Quentin", "nom": "Tarantino", "anneeNaiss": "1963", "rôle": "Jimmy Dimmick"}
        }
}
```

Inconvénient 4 : comment faire … ?

- par exemple comment connaître tous les films tournés par Tarantino ?
- il n'y a pas d'autre solution que de lire tous les documents

Film


```
{
 "titre": "Pulp fiction", "année": "1994", "genre": "Action", "pays": "USA",
        "Réalisateur": {
                "nomReal": "Tarantino", "prénomReal": "Quentin", "anneeNaissReal": "1963"
        }
        "Acteurs": {
                {"prénom": "John", "nom": "Travolta", "anneeNaiss": "1954", "rôle": "Vincent Vega"},
                {"prénom": "Bruce", "nom": "Willis", " anneeNaiss": "1955", "rôle": "Butch Coolidge" },
                {" prénom": "Quentin", "nom": "Tarantino", "anneeNaiss": "1963", "rôle": "Jimmy Dimmick"}
        }
}
```

NoSQL

Imbrication des structures

Schéma ?

- les systèmes NoSQL ne proposent pas forcément de schéma, ou en tout cas rien d'équivalent aux schémas relationnels
- il existe un gain apparent : on peut tout de suite, sans modélisation, commencer à insérer des documents
- mais rapidement la structure de ces documents change, et on ne sait plus trop ce que l'on a mis dans la base qui peut rapidement devenir une poubelle de données
- si on veut éviter cela, c'est au niveau de l'application effectuant des insertions qu'il faut effectuer la vérification des contraintes qu'un système relationnel peut nativement prendre en charge