Bases de données documentaires et distribuées, http://b3d.bdpedia.fr

Modélisation

Rappel sur la conception relationnelle

Pour l'essentiel.

- déterminer les "entités" (film, réalisateurs, acteurs) pertinentes pour l'application ;
- définir une méthode d'identification de chaque entité ;
- préserver le lien entre les entités.

Exemple pour notre application "Les films"

- On distingue deux types d'entités : les films et les réalisateurs.
- On les identifie par un numéro incrémental (l'id)
- On référence, dans chaque film, le réalisateur grâce à son identifiant.

Illustration

La table des films

id	titre	année
1	Alien	1979
2	Vertigo	1958
3	Psychose	1960
4	Kagemusha	1980
5	Volte-face	1997
6	Pulp Fiction	1995

La table des artistes

id	nom	prénom	année
101	Scott	Ridley	1943
102	Hitchcock	Alfred	1899
103	Kurosawa	Akira	1910
104	Woo	John	1946
105	Tarantino	Quentin	1963

Il manque le lien entre les films et les artistes.

Le système de référencement en relationnel

La table des films.

id	titre	année	idRéalisateur
1	Alien	1979	**101**
2	Vertigo	1958	**102**
3	Psychose	1960	**102**
4	Kagemusha	1980	**103**
5	Volte-face	1997	**104**
6	Pulp Fiction	1995	**105**

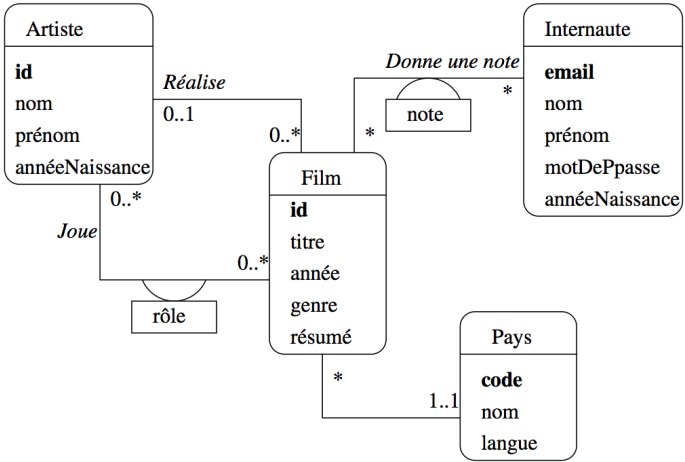
La table des artistes

id	nom	prénom	année
101	Scott	Ridley	1943
102	Hitchcock	Alfred	1899
103	Kurosawa	Akira	1910
104	Woo	John	1946
105	Tarantino	Quentin	1963

Ce n'est pas un lien physique, mais il permet des jointures.

Méthode générale : schémas entité/association

Exemple pour notre base de films.



Schémas relationnels

Exemple pour notre base de films.

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

Exprime des **contraintes** : clé primaire, unicité (nom, prénom), typage, valeurs obligatoires.

Schémas relationnels (2)

La table des films :

```
create table Film  (idFilm integer not null,
                    titre    varchar (50) not null,
                    annee    integer not null,
                    idRealisateur    integer not null,
                    genre varchar (20) not null,
                    resume      varchar(255),
                    codePays    varchar (4),
                    primary key (idFilm),
                    foreign key (idRealisateur)
                        references Artiste);
```

La **contrainte d'intégrité référentielle** `foreign key` garantit la cohérence du référencement.

La normalisation relationnelle

En relationnel, toutes les données sont "à plat". Cela tend à multiplier le nombre de tables contenant les informations sur une même entité.

Exemple : pour représenter toutes les informations sur un film, il faut

- La table `Film` (une ligne pour le film)
- La table `Artiste` (une ligne pour le réalisateur, autant de lignes que d'acteurs)
- La table `Role` (autant de lignes que de rôles)
- et la table `Pays`, et d'autres encore...

Il faut beaucoup **d'écritures**, et autant de **lectures** pour reconstituer l'information.

Représentation relationnelle du film (Pulp Fiction)

Les films :

id	titre	année	idReal
17	Pulp Fiction	1994	37

Les artistes :

id	nom	prénom	naissance
37	Tarantino	Quentin	1963
11	Travolta	John	1954
27	Willis	Bruce	1955

Les rôles

idFilm	idArtiste	rôle
17	11	Vincent Vega
17	27	Butch Coolidge
17	37	Jimmy Dimmick

Impact de la normalisation

Essentiel : effet de la normalisation

- il faut effectuer **plusieurs écritures** pour une même entité (transactions)
- il faut effectuer des **jointures** pour reconstituer une entité.

Reconstitution de l'information sur un film en SQL :

```
select titre, nom, prenom, role
from Film, Artiste, Role
where role='Vincent Vega'
and Film.id = Role.idFilm
and Artiste.id = Role.idActeur
```

Puissance du modèle semi-structuré

Les documents structurés ne sont pas soumis aux contraintes de normalisation.

Un attribut peut avoir **plusieurs valeurs** (en utilisant la structure de tableau)

```
{
  "title": "Pulp fiction",
  "year": "1994",
  "genre": ["Action", "Policier", "Comedie"]
  "country": "USA"
}
```

En relationnel, il faudrait ajouter deux tables.

Puissance du modèle semi-structuré (2)

Il est également facile de représenter des données régulières.

id	nom	prénom
37	Tarantino	Quentin
167	De Niro	Robert
168	Grier	Pam

Facile à représenter sous forme d'un document (très régulier).

```
[
  artiste:  {"id": 37, "nom": "Tarantino", "prenom": "Quentin"},
  artiste:  {"id": 167, "nom": "De Niro", "prenom": "Robert"},
  artiste:  {"id": 168, "nom": "Grier", "prenom": "Pam"}
]
```

Table relationnelle = arbre de hauteur constante. Trois niveaux : ligne, attribut, valeur.

Plus puissant que la représentation relationnelle ?

On peut donc aussi représenter des données régulières

Probablement pas du tout efficace. **Pourquoi ?**

$\Rightarrow$ peut être utile pour échanger des informations ; pas pour les stocker.

Premier constat

- Dans une base relationnelle, les données sont très contraintes / structurées : **on peut stocker séparément la structure (le schéma) et le contenu (la base)**.
- Une représentation arborescente XML / JSON est plus appropriée pour des données de structure **complexe** et / ou **flexible**.

Documents structurés = imbrication des structures

Grâce à **l'imbrication des structures**, il est possible de représenter dans une même unité un film **et** son metteur en scène.

```
{ "title": "Pulp fiction",
    "year": "1994",
    "genre": "Action",
    "country": "USA",
    "director": {"last_name": "Tarantino",
                 "first_name": "Quentin",
                 "birth_date": "1963"
               }
}
```

Dans les bases documentaires : on essaie de créer des unités d'information **autonomes** pour limiter les écritures et éviter d'avoir à faire des jointures.

Intérêt ?

Les systèmes NoSQL sont conçus pour passer à l'échelle par distribution. C'est en partie incompatible avec les jointures et les transactions.

Représentation sous forme de document structuré

On peut coder toutes les informations sur un film.

```
{
    "_id": "movie:17",
    "title": "Pulp Fiction",
    "year": "1994",
    "director":         {
            "last_name": "Tarantino", "first_name": "Quentin",
            "birth_date": "1963"
            },
    "actors": [
       {
         "first_name": "John", "last_name": "Travolta",
         "birth_date": "1954",  "role": "Vindent Vega"
       },
       {
        "first_name": "Bruce", "last_name": "Willis",
        "birth_date": "1955", "role": "Butch Coolidge"
       },
       {
        "first_name": "Quentin", "last_name": "Tarantino",
        "birth_date": "1963", "role": "Jimmy Dimmick"
       }
    ]
```

Bilan - à retenir

Les avantages de la représentation par document structuré.

- **Plus besoin de jointure (?)** : il est inutile de faire des jointures pour reconstituer l'information puisqu'elle n'est plus dispersée, comme en relationnel, dans plusieurs tables.
- **Plus besoin de transaction (?)** : une écriture (du document) suffit ; une lecture suffit pour récupérer l'ensemble des informations.
- **Adaptation à la distribution**. Si les documents sont autonomes, il est très facile des les déplacer pour les répartir au mieux dans un système distribué.

Un système NoSQL est beaucoup plus facile à implanter qu'un système relationnel. Une opération *put()*, une opération *get()* et c'est parti.

Les inconvénients du modèle

La représentation par document a deux inconvénients forts.

- **Chemin d'accès privilégié** : les films apparaissent près de la racine des documents, les artistes sont enfouis dans les profondeurs ;
  L'accès aux films est donc privilégié
- **Les entités ne sont plus autonomes** : pas moyen de créer un réalisateur s'il n'y a pas au moins un film.
- **Redondance** : la même information doit être représentée plusieurs fois, ce qui est tout à fait fâcheux (Quentin Tarantino est représenté deux fois).

La redondance mène à des incohérences.

Privilégier un chemin d'accès est bon pour certaines applications, mauvais pour d'autres.

Comment faire pour obtenir la liste des films de Quentin Tarantino avec la représentation précédente ?

Les autres inconvénients

Pas de langage de requête ?
Il faut faire un programme pour tout, même la moindre mise à jour !

Pas de schéma ?
On peut mettre n'importe quoi dans la base ; c'est l'application qui doit faire les contrôles et les corrections.

Pas de transaction ?
Ne convient pas pour beaucoup d'applications (commerce électronique).

Bilan, NoSQL ou relationnel ?

Quand utiliser (ou pas) une base documentaire ?

- **Des données très spécifiques, peu ou faiblement structurées** (texte, données multimédia, graphes)
- **Peu de mises à jour, beaucoup de lectures.** ; la redondance ne pose pas de problème.
- **on veut traiter de très gros volumes de manière "scalable"**.
- **De forts besoins en temps réel**.

Conditions non remplies : un système relationnel est probablement une bien meilleure option.

Bien retenir

NoSQL = *Not Only SQL*. Personne (de sérieux) ne prétend que ces systèmes vont remplacer les systèmes relationnels, sauf pour certaines niches.

Quelques éléments de réflexion

Toujours se poser (au moins) les questions suivantes :

- S'agit-il de représenter une structure régulière (p.e. une table) ?
  Dans ce cas il n'est pas nécessaire d'intégrer la structure et le contenu.
- Est-ce que je dois souvent modifier les données existantes ?
  Dans ce cas j'ai sans doute besoin de transactions, et je risque de payer cher toute redondance.
- La structure d'un document est-elle fixe et prévisible ?
  Si oui, le relationnel est bien adapté.

La principale niche des bases NoSQL

Comme entrepôts de données où on accumule des informations dans l'optique de traitements analytiques.