Datastore - Les données chez Google

3 décembre 2014

Google Cloud Platform

- Ensembles d'outils permettant le développement d'applications ;
- PaaS, stockage, analyse, recherche, autentification,... ;
- `https://cloud.google.com/products/app-engine`.

Stockage de données

Il existe plusieurs solutions de stockage :

- Cloud SQL ;
- Datastore ;
- Cloud Storage, Blobstore ;

Datastore

Présentation

3 couches composent le datastore :

- Big Table ;
- Megastore ;
- Datastore.

Big Table

Présentation

Big Table est le système de stockage de Google (2005). Ses caractéristiques sont :

- distribué et multi-dimentionnel ;
- scalable, tolérent aux fautes ;
- autogéré ;
- (éventuellement) consistant ;
- données indexées sous forme de map en **familles de colonnes** ;

NoSQL

Vocabulaire

- Genre $\leftrightarrow$ Table
- Entité $\leftrightarrow$ Donnée (n-uplet)
- Attribut $\leftrightarrow$ Attribut

NoSQL

CRUD : Fonctions de base

- Lire : **get**(clé)
- Écrire, modifier : **put**(clé, valeur)
- Supprimer : **delete**(clé)

Datas

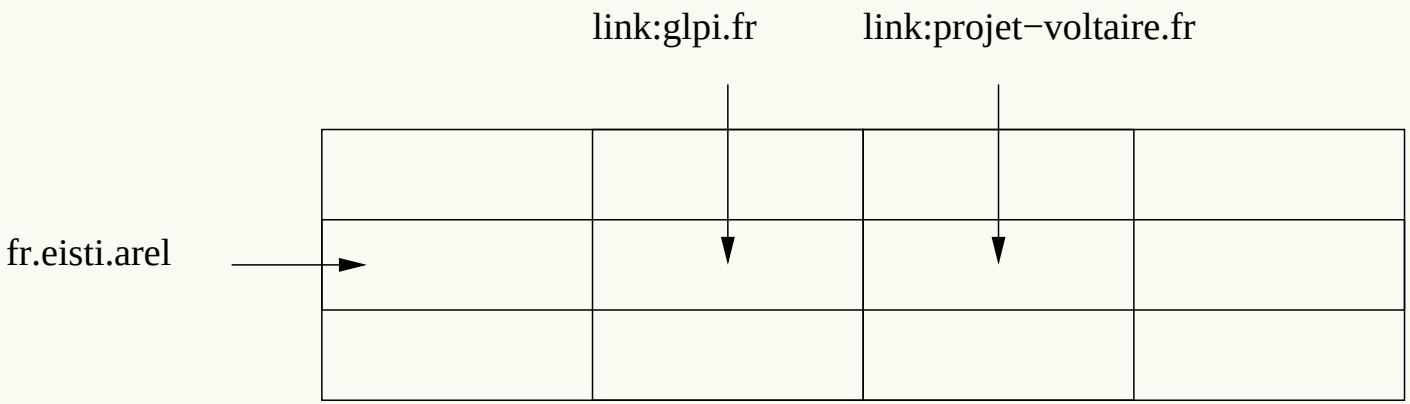
Présentation

Principe du système de stockage :

- (cléLigne : chaine, cléColonne : chaine, timestamp : entier) $\rightarrow$ chaine :
  - cléLigne : famille de colonne ;
  - cléColonne : id de la donnée ;
  - timestamp : version.

Big Table

Illustration



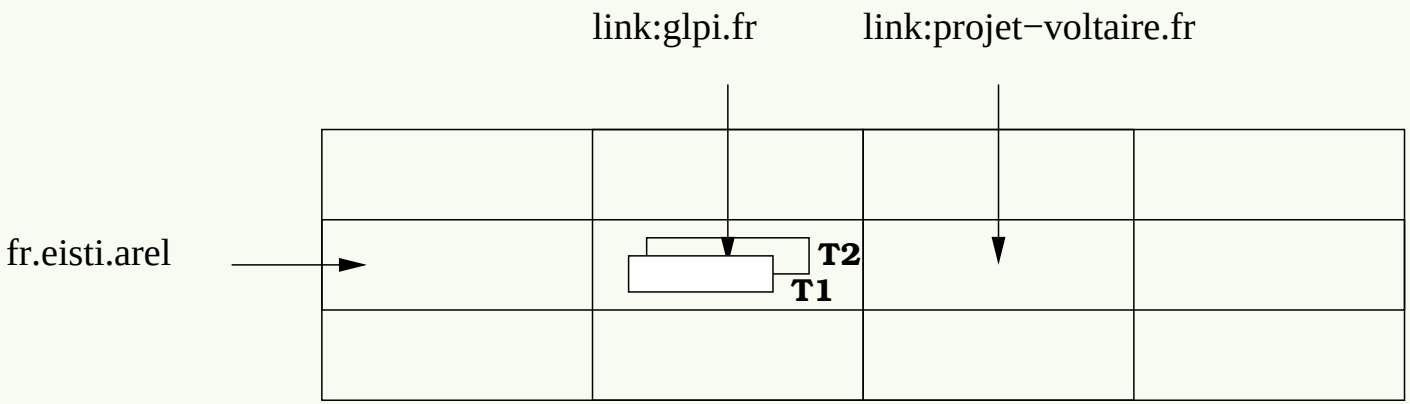
Big Table

Timestamp

- Les familles de colonnes peuvent contenir plusieurs version de la donnée
- Stocke un nombre fini n de versions
- Par défaut, retourne la dernière version

Big Table

Illustration



NoSQL

Requêtage

- Dénormalisation des données ;
- Mises à jour multiples ;
- Jointures souvent coûteuses ;
- Certaines opérations absentes de certains types de stockage (OR, multi comparaisons).

Restrictions

Restrictions de GAE

- comparaisons ($<, >, !=$) sur une propriété (ie par famille), **triée et indexée** ;
- multicritère $\Rightarrow$ index multiple
- opérateurs OR et NOT non supportés ;
- pas de regroupement (expérimental), ni de fonction d'agrégat (stocker l'information si possible).

APIs

Présentation

- GAE propose 3 APIs afin de gérer les données ;
- GAE conseille d'utiliser JDO et JPA ;
- richesse d'annotations, de fonctions pour la manipulation de données ;
- ne dépendent pas de la bdd ;
- pas d'adaptation de code ;
- modèle de données fixe.

Java Data Objects

JDO

- API de persistence de données ;
- 1erstandard Java pour la persistence, depuis Java 1.3 ;
- utilisée pour mapper des objets de plusieurs types de bases (relationnelle, objet, hiérarchique) ;
- langage de requétage : JDQL, SQL ;
- prédéfinition des classes métiers, annotées.

Java Persistence API

JPA

- API de persistence de données ;
- designé par les grands acteurs de l'ORM (Hibernate) ;
- intégré depuis Java 1.5 ;
- utilisée pour mapper des objets d'une base relationnelle uniquement ;
- langage de requétage : JPQL, SQL ;
- prédéfinition des classes métiers, annotées.

Low-level API

Présentation

- manipule les objets "au plus près" du stockage (datastore) ;
- simple et rapide d'utilisation ;
- code non portable vers d'autres systèmes de stockage ;
- basé sur la classe `Entity`.

La classe Entity

Présentation

Unité fondamentale de stockage d'une donnée dans le datastore. Elle est définie par :

- un type (kind) : dénomination de la donnée ;
- une clé (key) : clé primaire de la donnée, composée de :
  - type de la donnée
  - id ou name, identifiant unique
  - éventuellement, un parent
- parent : noeud du système de stockage. S'il est défini, l'entité est créée sous ce noeud ;
- un ensemble de propriétés accessible depuis des accesseurs (`get/setProperty`).

La classe DatastoreService

Présentation

permet de manipuler une entité

- `get` : récupère une entité ;
- `delete` : supprime l'entité ;
- `put` : enregistre l'entité ;
- `prepare` : prépare une requète.

Entity group

Présentation

- permet de spécifier une relation "parentale" entre entités ;
- une transaction est atomique pour un groupe d'entités ;
- propriété optionnelle, le parent : **ancestor path** ;
- ralentit les traitements.

Exemple

Manipulation

```
// AJOUT
// Création d'une entité
Entity prof = new Entity("Prof","koko");
prof.put();

Entity adr = new Entity("Adresse", prof.getKey());
adr.setProperty("rue", "2 blvd Lucien Favre");
adr.setProperty("ville", "Pau");
adr.put();

// Enregistrement
```

La classe Query

Présentation

permet de construire une requète sur un type d'entité.

- gestion de tris, de filtres sur une famille ;
- opérateurs de comparaisons, de contenance ;
- la classe `PreparedQuery` exécute une requète et récupère les données.

Une requête est **un scan d'index** !

Exemple

Manipulation

```
// AJOUT
// Création d'une entité
Entity prof = new Entity("Prof","koko");
prof.setProperty("firstName", "Florent");
prof.setProperty("lastName", "Devin");

// Enregistrement
DatastoreService ds =
      DatastoreServiceFactory.getDatastoreService();
Transaction tx = ds.beginTransaction();
ds.put(prof);
tx.commit();
```

Exemple

Sélection

```
// Par critère de restriction
String firstName, lastName;
Query q = new Query("Prof");
q.addFilter("lastName", Query.FilterOperator.EQUAL, "Devin");

PreparedQuery pq = ds.prepare(q);
for (Entity result : pq.asIterable()) {
  firstName = (String) result.getProperty("firstName");
  lastName = (String) result.getProperty("lastName");
  System.out.println(lastName + " " + firstName);
}

// En reconstruisant sa clé :
Key profKey = KeyFactory.createKey("Prof", "koko");
Entity prof = datastore.get(profKey);
System.out.println(prof.getProperty("lastName") + " " +
                   prof.getProperty("firstName"));
```

Exemple

Sélection

```
// tri
Query q = new Query("Prof");
q.addSort("nom", SortDirection.ASCENDING);

List<Entity> res=datastore.prepare(q)
                .asList(FetchOptions.Builder.withDefaults());
//filtre
q = new Query("Prof");
q.addFilter("age", FilterOperator.GREATER_THAN, 33);
q.addSort("age", SortDirection.DESCENDING);
q.addSort("name", SortDirection.ASCENDING);

List<Entity> res=datastore.prepare(q)
                .asList(FetchOptions.Builder.withDefaults());
```

Exemple

Manipulation

```
// Sur un entity group
Key profKey = KeyFactory.createKey("Prof", "koko");
Entity prof = datastore.get(profKey);

Query q = new Query("Adresse")
.setAncestor(prof.getKey());

List<Entity> res=datastore.prepare(q)
                .asList(FetchOptions.Builder.withDefaults());
```

Autres API

Présentation

- non fournies par Google ;
- basées sur la classe Entity (non portables) ;
- gestion simplifiée des traitements (requêtes, relations) ;
- compromis entre JDO, JPA et l'API de bas niveau ;
- définissent une classe mais manipule des entités ;
- Objectify, Obgaektify, Twig, ...
- Optimisations MemCache !

Exemple

Twig persist

```
class Prof {
    @Key String id;
    String fisrtName;
    String lastName;
}

ObjectDatastore ds = new AnnotationObjectDatastore();
//insertion
Key prfKey = ds.store(new Prof("koko", "Florent", "Devin"));

//sélection
Prof prf = ds.load(Prof.class, "koko");
Iterator<Prof> listPrf = ds.find()
    .type(Prof.class)
    .addFilter("firstName", EQUAL, "Florent")
    .returnResultsNow();
```

Exemple

Obgaektify

```
// Ajout d'un nouveau professeur
def prof = new Prof() << [firstName:"Peio",
                          lastName:"Loubière", age:63]
def profKey = prof.store()

// selections
prof = Prof.fetch(profKey)  //ou profKey.fetch()

// profs de plus de 30ans,
// classés par ordre décroissant d'âge
def res = Prof.search(filter:["age>=" :"30"],sort:["-age"])

//suppression
prof.destroy()
```

Liens

Documentation

- Présentation :
  `https://developers.google.com/appengine/docs/java/overview?hl=fr-FR`
- Documentation :
  `https://developers.google.com/appengine/docs/`
- API :
  `https://developers.google.com/appengine/docs/java/javadoc/index`