Les types de documents illustration avec JSON Schema

E.Coquery

emmanuel.coquery@univ-lyon1.fr

https://forge.univ-lyon1.fr/mif24-bdnosql/mif24-bdnosql

Structure des données

Avoir une description (formelle) de la structure des données

- Pouvoir requêter ces données
- Pouvoir stocker, indexer, etc

Valider les données (vérifier leur conformité à une description)

Schémas

Contraintes sur les données

- Forme
- Types
- Valeurs

Schémas

Contraintes sur les données

- Forme
- Types
- Valeurs

Dans le modèle relationnel

- Nombre et nom des attributs dans une relation
- Type des attributs
- Clés, dépendances

Dans les arbres ?

- Forme = structure : quels type de nœud, à quel endroit
- Types = types de base, mais aussi structure
- Clés : peu / pas utilisé

Schéma ↔ type

JSON Schema

Norme pour (entre autres) spécifier des schémas pour les documents JSON.

- JSON Schema Validation : contraintes de forme et de type
- Syntaxe JSON ⚠
- Sémantique des documents JSON Schema (Validation)
  ≈ types

Types et instances

- Type = description de ce qui est attendu / autorisé

Types et instances

- Type = description de ce qui est attendu / autorisé
- Sémantique d'un type : ensemble de valeurs correspondant au type

Types et instances

- Type = description de ce qui est attendu / autorisé
- Sémantique d'un type : ensemble de valeurs correspondant au type
- Instance d'un type : une des valeurs dans la sémantique du type

Types et instances

- Type = description de ce qui est attendu / autorisé
- Sémantique d'un type : ensemble de valeurs correspondant au type
- Instance d'un type : une des valeurs dans la sémantique du type

string

- Type : `string`
- Sémantique de `string` : l'ensemble des chaîne de caractères
- `"toto"` est une instance de `string`

Types et instances : notations

Sémantique d'un type *type*

$$[\![type]\!]$$

Valeur `val` instance du type *type*

$$\texttt{val} :: type$$

Types de base

- Types des données atomiques
  (non décomposables, *i.e.* $\neq$ tableaux et struct)
- Instances : valeurs de base : integers, chaînes de caractères, booléens, etc

Dans le cours on considère qu'il existe une liste fixée de types de base :

$$bool, int, float, string, date$$

Types des structures

Trois sortes de structures :

- Les *record*s (≈ `struct`)
- Les *dict*s (dictionnaires)
- Les *array*s (tableaux)

Des constructions de types différentes pour chacune

Records

Type = ensemble de couples (champ,type)

Syntaxe

$$< champ_1 : type_1, \ldots, champ_n : type_n >$$

où l'ordre des champs n'est pas important

$$< champ_1 : type_1, \ldots, champ_n : type_n >$$
$$=$$
$$< champ_{p(1)} : type_{p(1)}, \ldots, champ_{p(n)} : type_{p(n)} >$$

avec p permutation $(i.e.\ p : \{1, \ldots, n\} \rightarrow \{1, \ldots, n\}\ bijection)$

Records : sémantique

Sémantique

Tous les objets qui vérifient les conditions suivantes :

- chaque champ listé dans le type est présent
- la valeur associée à chaque champ $champ_i$ est une instance de $type_i$

$$[\![< champ_1 : type_1, \ldots, champ_n : type_n >]\!]$$

$$=$$

$$\{o \mid o \text{ objet et } \forall i, \text{ avec } 1 \leq i \leq n, \ acces(o, champ_i) \in [\![type_i]\!]\}$$

Records : exemples

$$< a : int, b : string >$$

- { "a" : 3, "b" : "toto" } instance
- { "a" : 3 } pas instance
  il manque le champ b
- { "a" : 3, "b" : "toto", "c" : true } instance
  le type ne contraint pas le champ c
- { "a" : " titi ", "b" : "toto" } pas instance
  la valeur du champ a n'est pas une instance de *int*

Dictionnaires

Type = type unique commun à tous les champs

Syntaxe

$\{type\}$ ou bien $dict(type)$

Sémantique

Tous les objets qui vérifient les conditions suivantes :

- la valeur associée à chaque champ $champ_i$ est une instance de $type$

$$[\![\{type\}]\!] \quad = \quad \{o \mid o \text{ objet et } \forall v \in valeurs(o), v \in [\![type]\!]\}$$

Dictionnaires : exemples

$$\{string\}$$

- { "a" : " titi ", "b" : "toto" } instance
- { "a" : 3, "b" : "toto" } pas instance
  le champ a n'a pas le bon type

$$\{< a : int >\}$$

- { "c" : { "a" : 3, "b" : "toto" } } instance
- { "c" : { "b" : "toto" } } pas instance
  le champ a n'est pas défini dans la sous-structure associée au champ c

Tableaux

Type = type des éléments du tableau

Syntaxe

$[type]$ ou bien $array(type)$

Sémantique

Tous les tableaux qui vérifient les conditions suivantes :

- la valeur associée à chaque case du tableau est une instance de *type*

$$[\![[type]]\!] = \{a \mid a \text{ tableau et } \forall v \in valeurs(o), v \in [\![type]\!]\}$$

Tableaux : exemples

$[string]$

- [" titi ", "toto"] instance
- [3, "toto"] pas instance
  la case 0 n'a pas le bon type

Multiplicité des types

Une valeur peut avoir plusieurs types

val =`{"a":"toto","b":"titi"}`

- $\{string\}$
- $< a : string, b : string >$
- $< a : string >$
- ...

Sous-typage sémantique

Definition (Sous-type)

t_1 est un *sous-type* de t_2 si et seulement si $[\![t_1]\!] \subseteq [\![t_2]\!]$

Notation :

$$t_1 \preceq t_2$$

Règles de sous-typage

$$\text{(Refl)}\ \frac{}{\tau \preceq \tau} \qquad \text{(Trans)}\ \frac{\tau \preceq \tau' \qquad \tau' \preceq \tau''}{\tau \preceq \tau''}$$

$$\frac{}{< a_1 : \tau_1, \ldots, a_k : \tau_k, a_{k+1} : \tau_{k+1} > \preceq < a_1 : \tau_1, \ldots, a_k : \tau_k >}$$

(AddField)

$$\text{(STField)}\ \frac{\tau_k \preceq \tau'_k}{< a_1 : \tau_1, \ldots, a_k : \tau_k > \preceq < a_1 : \tau_1, \ldots, a_k : \tau'_k >}$$

$$\text{(STArray)}\ \frac{\tau \preceq \tau'}{[\tau] \preceq [\tau']} \qquad \text{(STDict)}\ \frac{\tau \preceq \tau'}{\{\tau\} \preceq \{\tau'\}}$$

Propriétés du système de règles

Theorem (Correction)

Si $\tau_1 \preceq \tau_2$ est dérivable par les règles de sous-typage alors $[\![\tau_1]\!] \subseteq [\![\tau_2]\!]$.

Preuve : par induction sur les dérivations

Theorem (Complétude)

Si $[\![\tau_1]\!] \subseteq [\![\tau_2]\!]$ alors il existe une dérivation de $\tau_1 \preceq \tau_2$ par les règles de sous-typage.

Preuve : Par (double) induction sur les types + double récurrence pour le cas record $\preceq$record.

Types vs JSON Schema

Plus d'expressivité côté JSON-Schema :

- champs optionels ou obligatoires
- regexs de clés → type pour les dictionnaires
- gestion des nulls
- ...

Des raccourcis pour mutualiser certaines définitions de types

Les aspects typage sont une des applications de JSON Schema parmi d'autres

Digression : types primitifs JSON Schema

⚠ ≠ types de base ⚠

- Plus une sorte de type, *e.g.* `object`
- Liste restreinte : "null", "boolean", "object", "array", "number" ou "string"
- utilisés dans le champ `"type"` de JSON Schema

Ne pas confondre types primitifs JSON Schema et types de base, même si certains se correspondent.

Pour simplifier, on utilisera en JSON Schema les types primitifs "boolean", "number" et "string" comme une représentation des types de base *bool*, *int*, *float* et *string*.

Records en JSON Schema

$$< champ_1 : \tau_1, \ldots, champ_n : \tau_n >$$

```
{
  "type": "object",
  "properties": {
      "champ_1": { /* traduction de τ_1 */ },
      ...
      "champ_n": { /* traduction de τ_n */ }
  },
  "required": [ "champ_1", ..., "champ_n" ]
}
```

Dictionnaires en JSON Schema

$$\{\tau\}$$

```
{
  "type": "object",
  "patternProperties": {
      ".*": { /* traduction de τ */ }
  }
}
```

Tableaux en JSON Schema

$$[\tau]$$

```
{
  "type": "array",
  "items": { /* traduction de τ */ }
}
```

Exemple de document JSON

Collection d'albums de BD

```
{ "gaston": { "titre": "Gaston",
              "albums": [
                { "numero": 1,
                  "titre": "Gaston 1",
                  "auteurs": [ "Franquin", "Jidehem" ] },
                { "numero": 12,
                  "titre": "Le gang des gaffeurs",
                  "auteurs": [ "Franquin" ]}
              ] },
  "luckyluke": { "titre": "Lucky Luke",
                 "albums": [
                    { "numero": 1,
                      "titre": "La mine d'or de Dick Digger",
                      "auteurs": [ "Morris" ]}
                 ] } }
```

Type de la collection de BD

$$
\begin{array}{l}
\{ <titre : string, \\
\quad albums : [< numero : int, \\
\qquad\qquad titre : string, \\
\qquad\qquad auteurs : [string] > \\
\qquad\qquad] \\
\quad > \}
\end{array}
$$

ou bien avec la syntaxe alternative des types

$$
\begin{array}{l}
dict(<titre : string, \\
\qquad albums : array(<numero : int, titre : string, \\
\qquad\qquad\qquad\qquad auteurs : array(string) >) >)
\end{array}
$$

JSON Schema - Préambule

Informations liées à la gestion du document

```
{ "$schema":
        "https://json-schema.org/draft/2020-12/schema",
  "$id": "https://example.com/bds",
  ...
}
```

JSON-Schema pour la collection de BDs

Schéma : collection d'albums de BD

```
{ "$schema": "https://json-schema.org/draft/2020-12/schema",
  "$id": "https://example.com/bds",
  "type": "object",
  "patternProperties": {
     ".*": { "type": "object",
        "properties": {
            "titre": { "type": "string" },
            "albums": { "type": "array",
                "items": { "type": "object",
                    "properties": {
                        "titre": { "type": "string" },
                        "numero": { "type": "number" },
                        "auteurs": { "type": "array",
                            "items": { "type": "string" } }
                    },
                    "required": ["titre","numero","auteurs"]
                } }
            },
        "required": ["titre","albums"]
     }
  }
}
```