JSON, Document Modeling & Denormalization

NoSQL

A4- S8

ESILV
jihane.mali (at) devinci.fr

Chapter Contents

Chapter 1

JSON Documents

For this Practice work, we will study different problems over the conception of JSON documents

1.1 How to Produce Valid JSON Documents

1.1.1 JSON Validation

For each of the following documents, find the errors and validate the document online:
`http://jsonlint.com/`

1.1.1

```
{
    test : 1
}
```

Correction :

```
{
   "test" : 1
}
```

1.1.2

```
{
    ”name” : ”Nicolas”,
}
```

Correction :

```
{
   "name" : "Nicolas"
}
```

1.1.3

```
{
    ”identity” : {
        ”firstname” : ”Nicolas”
        ”lastname” : Travers
}
```

Correction :

```
{
   "identity" : {
      "firstname" : "Nicolas",
      "lastname" : "Travers"}
}
```

1.1.4

```
{
    ”lectures” : [MESIGI3715, {”title”:”Databases”, MESIGI3942”, 25]
}
```

Correction :

```
{
  "lectures" : ["MESIGI3715", {"title":"Databases"}, "MESIGI3942", 25]
}
```

1.1.5

```
{
    ”type”:”object”,
    ”properties”:{
        ”id”:{
            ”description”:”The unique identifier for a product”,
            ”type”:integer}
        ”name”:{
            ”description”:”Name of the product”
            ”type”:”string”},
        ”price”:{
            ”type”:”number”,
            ”minimum”:0,
            ”exclusiveMinimum”:true}
    ”required”:[”id”,”name”,”price”]
}
```

Correction :

```
{
  "type":"object",
  "properties":{
    "id":{
      "description":"The unique identifier for a product",
      "type":"integer"},
    "name":{
      "description":"Name of the product",
      "type":"string"},
    "price":{
      "type":"number",
      "minimum":0,
      "exclusiveMinimum":true}
  },
  "required":["id","name","price"]
}
```

1.2 JSONSchema (IETF Draft 4.0)

In order to verify typing while an application reads a JSON document, JSONSchema has been created to specify it. Then, an application can verify the content type of each received document and process only valid documents.

For this, you can use the JSONSchema generator:
`https://www.liquid-technologies.com/online-json-to-schema-converter`

1.2.1 Create JSONSchemas

1.2.1 Create JSONSchema for question 1.1.2,

Correction :

```
{
     "$schema": "http://json-schema.org/draft-04/schema#",
     "type": "object",
     "properties": {
          "name": {"type": "string"}
     },
     "required": ["name"]
}
```

The key "name" is required. And this key is of type "string".

1.2.2 The one for question 1.1.3,
Correction :

```
{
     "$schema": "http://json-schema.org/draft-04/schema#",
     "type": "object",
     "properties": {
          "identity": {
               "type": "object",
               "properties": {
                    "firstname": {"type": "string"},
                    "lastname": {"type": "string"}
               },
               "required": ["firstname", "lastname"]
          }
     },
     "required": ["identity"]
}
```

The nested document is itself a schema for "identity" (type object) that is embedded in "properties".

1.2.3 The one for question 1.1.4,
Correction :

```
{
     "$schema": "http://json-schema.org/draft-04/schema#",
     "type": "object",
     "properties": {
          "lectures": {
               "type": "array",
               "items": [
                    {"type": "string"},
                    {"type": "object",
                     "properties": {"title": {"type": "string"}},
                     "required": ["title"]
                    },
                    {"type": "string"},
                    {"type": "integer"}
               ]
          }
     },
     "required": ["lectures"]
}
```

The key ”lectures” is of type ”array” which items are of the **sequence** of types: ”string”, ”object” (with ”title”), ”string” and then ”integer”.

1.2.4 Interestingly, the document from question 1.1.5 is a JSONSchema. Create a JSON document that produces this JSONSchema.
Correction :

```
{
    "id" : 1,
    "name" : "Elastic gloves",
    "price" : 25.0
}
```

1.3 From Relational to JSON Documents

1.3.1 Modelize

Here a relationnal schema for our Amazon database:

PRODUCT	**id**	name	price	category	label
	1	Sony W800 Digital Camera	108	Camera	Sony
	2	Star Wars: The complete saga	88	Movies & TV	Lucasfilm
	3	Inside Out	23	Movies & TV	Disney PIXAD
	4	Samsung Gear Fit Smart Watch	99	Smart Watches	Samsung
	5	Apple Watch Sport 42mm	435	Smart Watches	Apple

CLIENT	**id**	firstname	lastname	address	zip_code	country
	1	George	Lucas	Skywalker ranch, Nicasio, California	94946	USA
	2	Harrison	Ford	P.O. Box 49344, Los Angeles, California	90049	USA
	3	Sean	Connery	Creative Artists Agency, 2000 Avenue of the Stars, Los Angeles, California	90067	USA
	4	Scarlett	Johansonn	Richbell Rd, Mamaroneck, New York	10543	USA

ORDER	**id_client**	**id_product**	**date**	amount
	1	2	2011-09-11	1
	1	3	2015-10-27	1
	2	2	2011-09-11	1
	3	5	2015-07-12	2
	4	5	2015-07-11	1

STOCK	**id_product**	**zip_code**	stock
	1	90049	123
	2	90049	25000
	2	10025	100000
	4	10025	3245
	4	90049	5342

1.3.1 For each relation, give a corresponding JSon example,
Correction :

```
{
  "id" : 1,
  "name" : "Sony W800 Digital Camera",
  "price" : 108,
  "category" : "Camera",
  "label" : "Sony"
}

{
  "id" : 1,
  "firstname" : "George",
  "lastname" : "Lucas",
  "address" : "Skywalker ranch, Nicasio, California",
  "zip_code" : 94946,
  "country" : "USA"
}

{
  "id" : {
    "id_client" : 1,
    "id_product" : 1,
    "date" : "2011-09-11"
  },
  "amount" : 1
}

{
  "id" : {
    "id_product" : 1,
    "zip_code" : 90049
  },
  "stock" : 123
}
```

1.3.2 A product can have several categories. Modify the example by adding a new value “High-Tech”,
Correction :

```
{
  "id" : 1,
  "name" : "Sony W800 Digital Camera",
  "price" : 108,
  "category" : ["Camera", "High-Tech"],
  "label" : "Sony"
}
```

1.3.3 Modify the client that has got a detailed address with: number, street, city, state/departement, zip_code, country
Correction :

```
{
  "id" : 1,
  "firstname" : "George",
  "lastname" : "Lucas",
  "address" : {
    "number" : 1,
    "street" : "Skywalker ranch",
    "city" : "Nicasio",
    "zip_code" : 94946,
    "state" : "California",
    "country" : "USA"
  }
}
```

1.3.2 Denormalize

In NoSQL databases, joins are hard to apply and very costly. To avoid this, denormalizing documents by **merging** two schemas together enables quick filtering (embedding documents is equivalent to a join).
For the following questions, propose a merged document which ease the required join:

1.3.1 Store all the orders corresponding to a given client,
Correction :

```
{
  "id" : 1,
  "firstname" : "George",
  "lastname" : "Lucas",
  "address" : {
    "number" : 1,
    "street" : "Skywalker ranch",
    "city" : "Nicasio",
    "zip_code" : 94946,
    "state" : "California",
    "country" : "USA"
  },
  "orderss" : [
    {"id_product" : 1, "date" : "2011-09-11", "amount" : 1},
    {"id_product" : 3, "date" : "2015-10-27", "amount" : 1}
  ]
}
```

1.3.2 Store the stock over all locations (zip_code) for a given product,

Correction :

```
{
  "id" : 1,
  "name" : "Sony W800 Digital Camera",
  "price" : 108,
  "category" : "Camera",
  "label" : "Sony",
  "stocks" : [
    {"zip_code" : 90049, "stock" : 123},
    {"zip_code" : 10025, "stock" : 250},
  ]
}
```