

NoSQL

# Module: 165

Utiliser des bases de données  
NoSQL

# Base de données NoSQL

## *La modélisation et relations*

# Base de données NoSQL

## *La modélisation*

```
[  
  {  
    name: "Paul",  
    year: 2007  
  },  
  {  
    title: "Yamaha",  
    type: "moto"  
  }  
]
```

Une collection, peut contenir des documents avec des structure totalement différentes.

Il n'y pas de contrainte de **schéma**!

# Base de données NoSQL

## *La modélisation*

Champs communs

Champs spécifiques

```
[  
  {  
    brand: "Mercedes",  
    type: "voiture",  
    year: 2011  
  },  
  {  
    brand: "Yamaha",  
    type: "moto",  
    year: 2007,  
    avialable: true,  
    price: 1200  
  }  
]
```



# MongoDB

## *La modélisation*

Pour faire le bon choix lors de la modélisation, il est important de se poser ces questions:.

- Que fait mon application?
- Quelles sont les données qui sont stockés?
- Comment ces données seront utilisées par les utilisateurs?
- Quelles données seront le plus importante pour moi?

# MongoDB

## *La modélisation*

Les avantages d'une bonne modélisation:

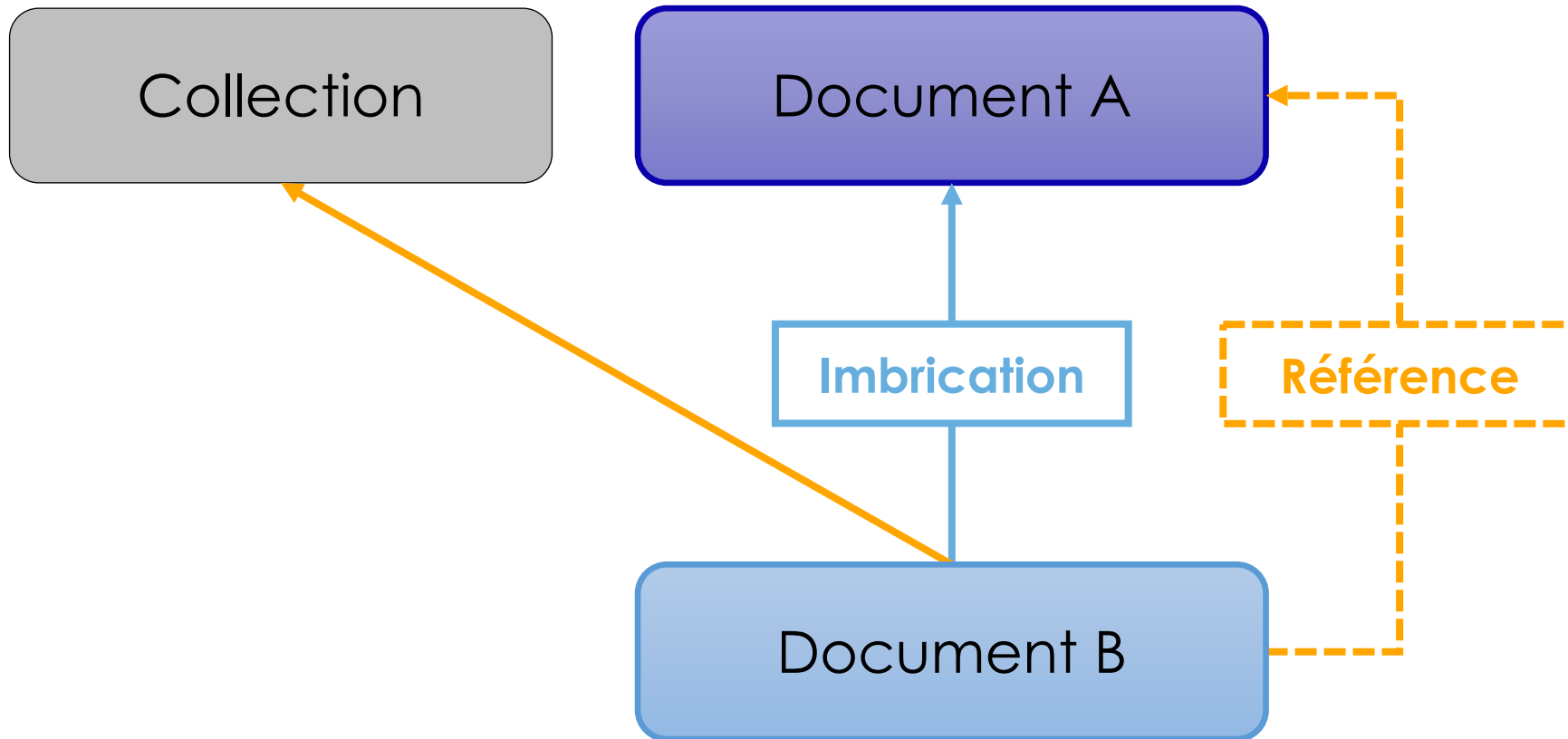
- Facilité de manipulation des données
- Optimisation des requêtes
- Reduction de l'usage des ressources CPU et mémoire
- Reduction des coûts

# MongoDB

## *Les relations*

# MongoDB

## *Les relations*





# Base de données NoSQL

## *L'imbrication*

L'imbrication (embedded data) est le fait d'inclure un ou plusieurs documents dans un autre document.

Cela permet de conserver les informations reliées entre elles dans un seul document.

On parle de **modèles de données dénormalisés**.



# Base de données NoSQL

## *L'imbrication: exemple*

```
{
  _id: ObjectId("613a2d6d2e6f373a48967e50"),
  title: "My First Blog Post",
  content: "Lorem ipsum dolor sit amet...",
  author: "John Doe",
  comments: [
    {
      username: "Jane Smith",
      comment: "Great post!",
      date: ISODate("2023-04-20T14:30:00.000Z")
    },
    {
      username: "Bob Johnson",
      comment: "Thanks for sharing!",
      date: ISODate("2023-04-21T10:00:00.000Z")
    }
  ]
}
```

# Base de données NoSQL

## *Les références*

Les références (references) sont des liens d'un document vers un autre document.

user document

```
{
  _id: <ObjectId1>,
  username: "123xyz"
}
```

contact document

```
{
  _id: <ObjectId2>,
  user_id: <ObjectId1>,
  phone: "123-456-7890",
  email: "xyz@example.com"
}
```

access document

```
{
  _id: <ObjectId3>,
  user_id: <ObjectId1>,
  level: 5,
  group: "dev"
}
```

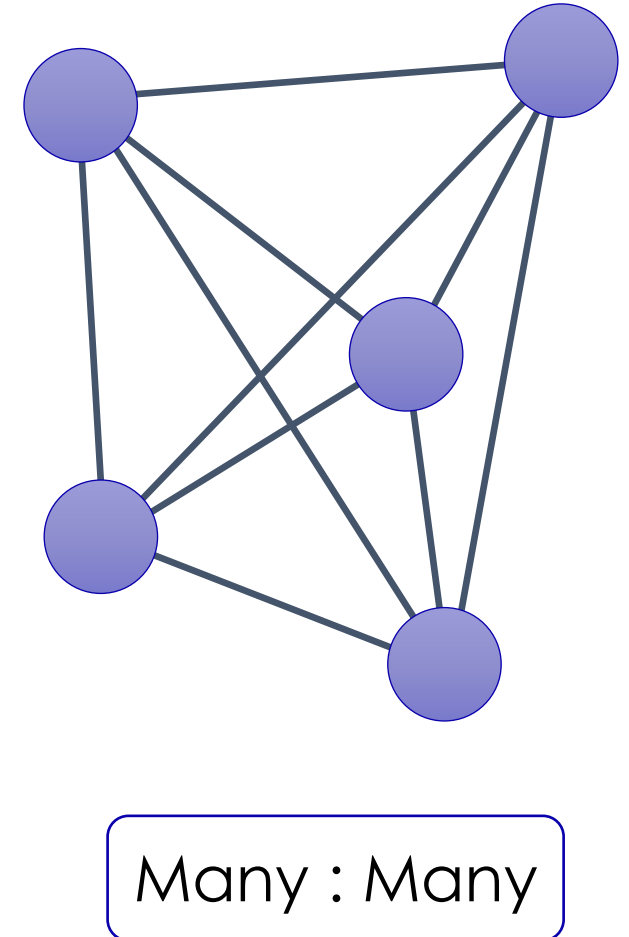
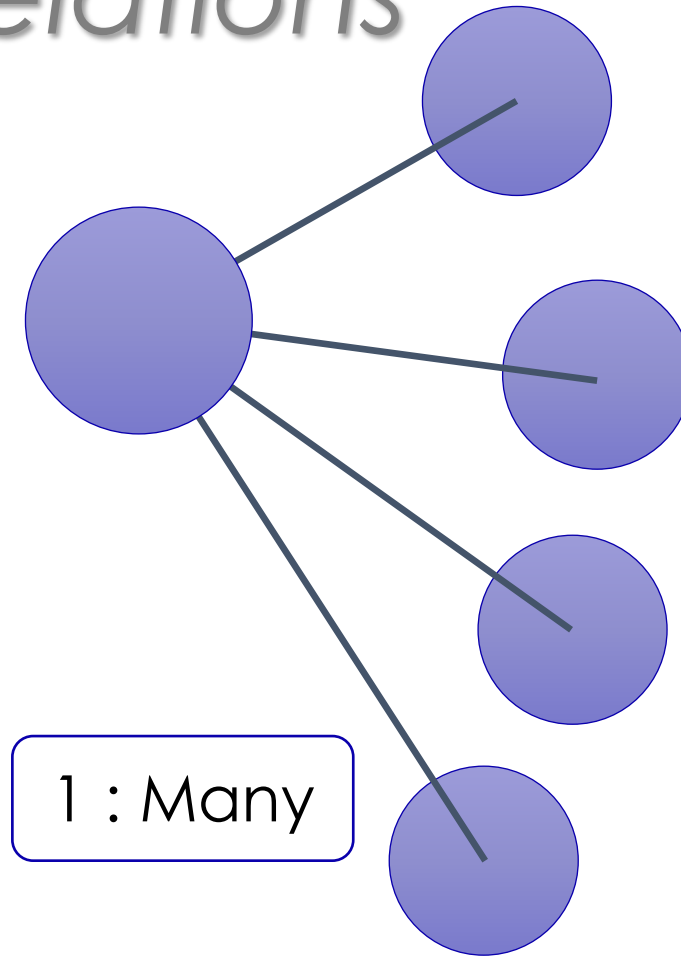
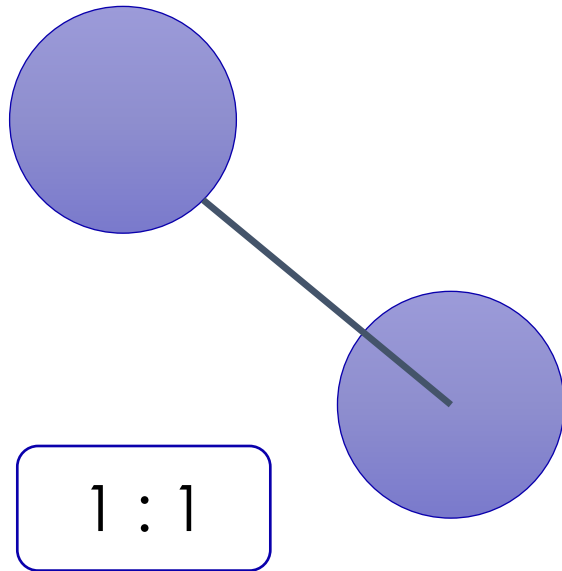
On parle de **modèles de données normalisés**.

# MongoDB

## *Types de relations*

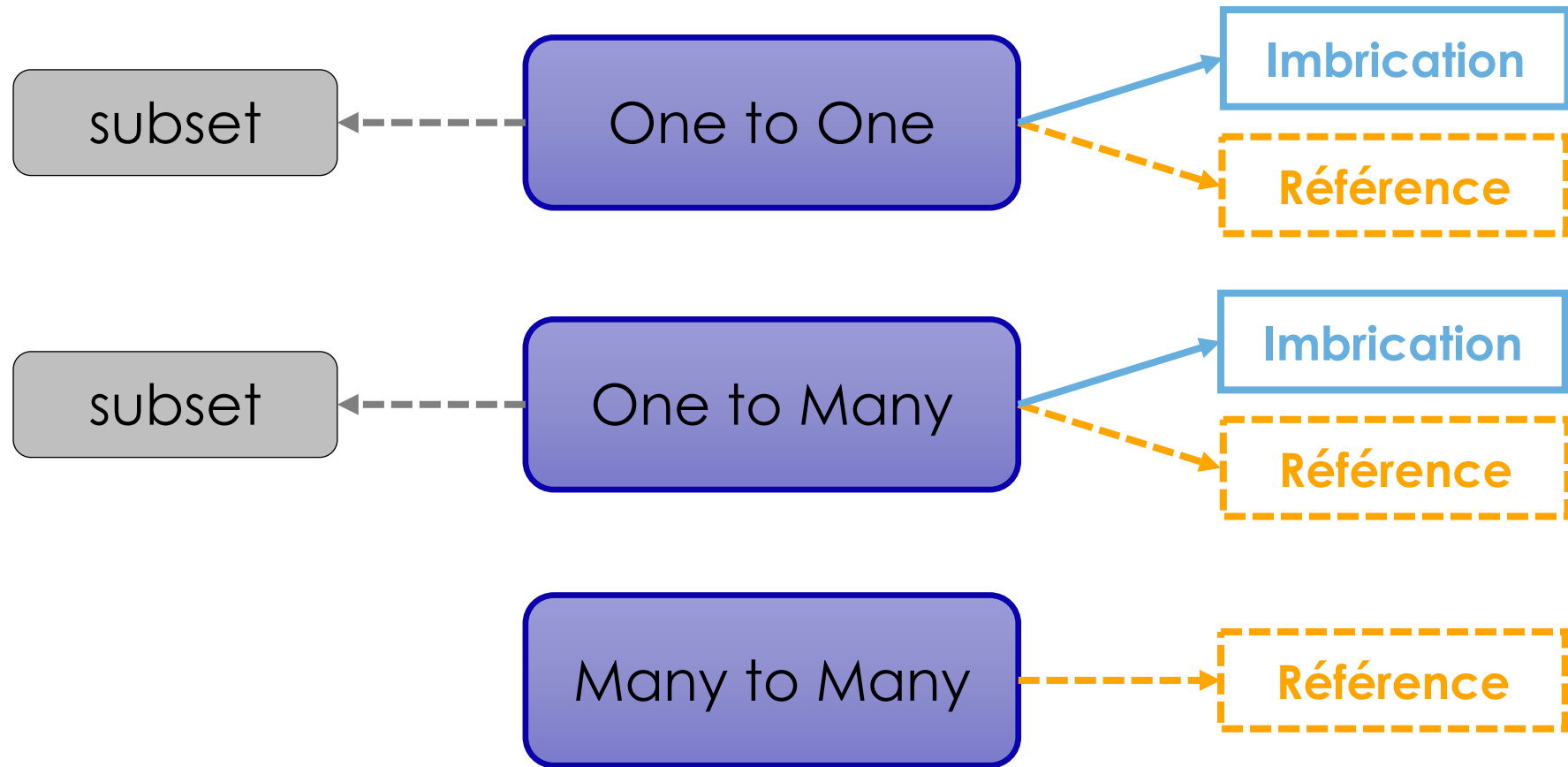
# MongoDB

## *Types de relations*



# MongoDB

## *Types de relations*



# MongoDB

## Relation: One-to-One

```
{
  "_id": {
    "$oid": "6027dbcb9a91e1b9eaec9de5"
  },
  "name": "Jean",
  "title": "CEO",
  "address": {
    "street": "rue du Four",
    "zip": 75000,
    "city": "Paris"
  }
}
```

Dans ce type de relation, dans la grande majorité des cas, on favorisera **l'imbrication**.

L'imbrication permet de récupérer les informations corrélées en une seule requête, permettant une plus grande simplicité d'utilisation par les applications et de meilleures performances.


# MongoDB

## Relation: One-to-Many

Les relations **One-to-Many** signifient qu'un document est associé à plusieurs documents.

```
// User document
{
  "_id": ObjectId("61c5d320f5db23428aaf2677"),
  "name": "John Doe"
}
```

```
// Order documents
{
  "_id": ObjectId("613a2d6d2e6f373a48967e50"),
  "user_id": ObjectId("61c5d320f5db23428aaf2677"),
  "order_date": ISODate("2023-04-23T00:00:00.000Z"),
  "total_amount": 100.00
},
{
  "_id": ObjectId("321a2d6d2e6e373a489673b5"),
  "user_id": ObjectId("61c5d320f5db23428aaf2677"),
  "order_date": ISODate("2023-04-22T00:00:00.000Z"),
  "total_amount": 50.00
}
```





# MondoDB

## *Relation: One-to-Many*

```
{
  _id: ObjectId("613a2d6d2e6f373a48967e50"),
  title: "My First Blog Post",
  content: "Lorem ipsum dolor sit amet...",
  author: "John Doe",
  comments: [
    {
      username: "Jane Smith",
      comment: "Great post!",
      date: ISODate("2023-04-20T14:30:00.000Z")
    },
    {
      username: "Bob Johnson",
      comment: "Thanks for sharing!",
      date: ISODate("2023-04-21T10:00:00.000Z")
    }
  ]
}
```

# MongoDB

## *Relation: One-to-Many*

```
// blogpost document
{
  _id: ObjectId("613a2d6d2e6f373a48967e50"),
  title: "My First Blog Post",
  content: "Lorem ipsum dolor sit amet...",
  author: "John Doe",
  comments: [
    ObjectId("613a2f892e6f373a48967e51"),
    ObjectId("613a2f892e6f373a48967e52")
  ]
}
```

# MondoDB

## *Relation: One-to-Many*

```
{  
  _id: ObjectId("613a2f892e6f373a48967e51"),  
  post_id: ObjectId("613a2d6d2e6f373a48967e50"),  
  username: "Jane Smith",  
  comment: "Great post!",  
  date: ISODate("2023-04-20T14:30:00.000Z")  
}
```

# MongoDB

## *Relation: One-to-Many*

Les relations One-to-Many signifient qu'un document est associé à plusieurs documents.

- Références côté **One**
- Référence côté **Many**
- L'imbrication côté **One**
- L'imbrication côté **Many**

# MongoDB

## Relation: *Many-to-Many*

Les relations **Many-to-Many** signifient que plusieurs documents sont associés à plusieurs documents.

```
// User documents
{
  "_id": "user1",
  "name": "John Doe",
  "group_ids": ["group1", "group2"]
},
{
  "_id": "user2",
  "name": "Jane Smith",
  "group_ids": ["group1", "group3"]
}
```

```
// Group documents
{
  "_id": "group1",
  "name": "Technology Enthusiasts",
  "user_ids": ["user1", "user2"]
},
{
  "_id": "group2",
  "name": "Fitness Fanatics",
  "user_ids": ["user1"]
},
{
  "_id": "group3",
  "name": "Foodies",
  "user_ids": ["user2"]
}
```

# MongoDB

## Relation: *Many-to-Many*

Les relations **Many-to-Many** signifient que plusieurs documents sont associés à plusieurs documents.

- Imbrication dans un **Many**.
- Imbrication dans l'autre **Many**.
- Références côté d'un **Many**.
- Références côté de l'autre **Many**.

# MongoDB

## Relation: One-to-Zillions

Ce type de relation est simplement un type particulier de relation One-to-Many dans le cas où le côté **Many** est énorme.

```
{
  _id: ObjectId('1234'),
  brand: "Audi",
  model: "S5"
}
```

```
{
  _id: ObjectId('679'),
  time: 1612794381816,
  position: {
    lat:
    lng:
  },
  car_
}
```

```
{
  _id: ObjectId('789'),
  time: 1612794382816,
  position: {
    lat:
    lng:
  },
  car_i
}
```

```
{
  _id: ObjectId('466'),
  time: 1612794211816,
  position: {
    lat: 48.6124,
    lng: 2.3701
  },
  car_id: ObjectId('1234')
}
```

# MongoDB

## Modélisation

On peut prendre par exemple d'un document décrivant un étudiant.

Ici l'on voit que l'on a 3 numéros de téléphone.

```
{  
  "_id": ObjectId("00000001"),  
  "student": "Marnie Dupree",  
  "age": 18,  
  "student_id": 123456,  
  "email": "mdupree@college.edu",  
  "home_phone": "0241234567",  
  "cell_phone": "0791234567",  
  "city": "Lausanne",  
  "zip": 1002,  
  "street": "Av. d'Ouchy 24",  
  "emergaency_contact_name": "Jacques Dupree",  
  "emergency_contact_number": "0795673412",  
  "emergency_contact_relation": "father",  
  ...  
}
```



# MongoDB

## Modélisation

Une optimisation est de mettre tous les numéros de téléphone dans un tableau (array).

Mais l'on perd l'information de quel type de téléphone.

```
{
  "_id": ObjectId("00000001"),
  "student": "Marnie Dupree",
  "age": 18,
  "student_id": 123456,
  "email": "mdupree@college.edu",
  "phone": ["0241234567", "0791234567", "0795673412"],
  "city": "Lausanne",
  "zip": 1002,
  "street": "Av. d'Ouchy 24",
  "emergency_contact_name": "Jacques Dupree",
  "emergency_contact_relation": "father",
  ...
}
```

# MongoDB

## Modélisation

Donc , il serait préférable dans ce tableau d'y ajouter des objets avec pour chaque numéro un \*type\*.

```
{
  "_id": ObjectId("00000001"),
  "student": "Marnie Dupree",
  "age": 18,
  "student_id": 123456,
  "email": "mdupree@college.edu",
  "phone": [
    {"number": "0241234567", "type": "home"},
    {"number": "0791234567", "type": "cell"},
    {"number": "0795673412", "type": "emergency"}
  ],
  "city": "Lausanne",
  "zip": 1002,
  "street": "Av. d'Ouchy 24",
  "emergency_contact_name": "Jacques Dupree",
  "emergency_contact_relation": "father",
  ...
}
```

# MongoDB

## Relation: *Imbriqué* vs *Référence*

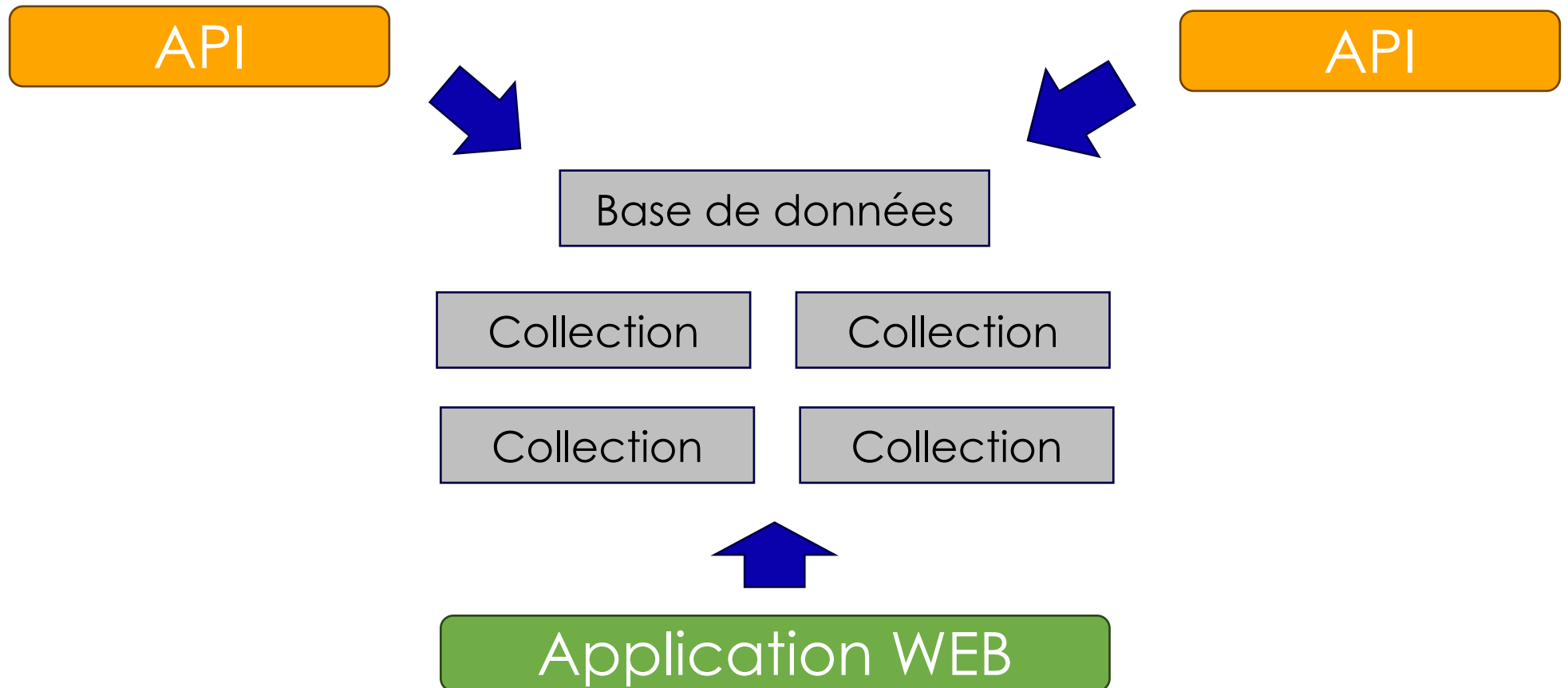
	Imbriqué	Référence
Avantages	<ul style="list-style-type: none"><li>- Performance de lecture</li><li>- Consistance atomique</li><li>- Pas de jointures</li><li>- Une seule requête</li></ul>	<ul style="list-style-type: none"><li>- Économie d'espace</li><li>- Facilité de mise à Jour</li></ul>
Inconvénients	<ul style="list-style-type: none"><li>- Duplication de données</li><li>- Mises à jour coûteuses</li><li>- Taille d'un document</li></ul>	<ul style="list-style-type: none"><li>- Complexité des requêtes</li><li>- Performance de lecture</li><li>- Consistance non atomique</li></ul>

# MongoDB

## *Validation des schémas*

# MongoDB

## *Validation des schémas*



# MongoDB

## *Validation des schémas*

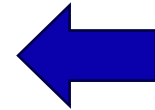
**Exemple:**  
Joy, Django ...

**Exemple:**  
Formik, Angular ...

Validation côté Mongod

Validation côté  
application (back)

Validation côté  
application (front)



ORM

# MongoDB

## *Validation des schémas*

### Création

insertOne,  
insertMany, etc.

Collection



Validation



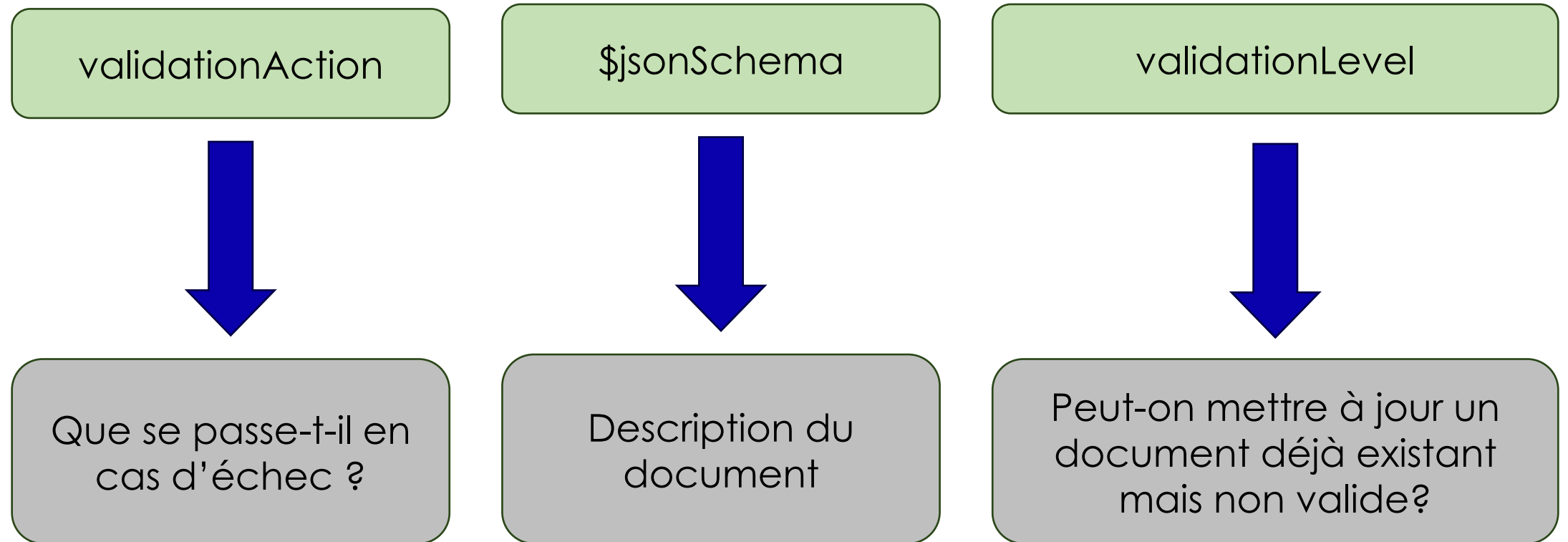
Documents

### Mise à jour

updateOne,  
updateMany, etc.

# MongoDB

## *Validation des schémas*

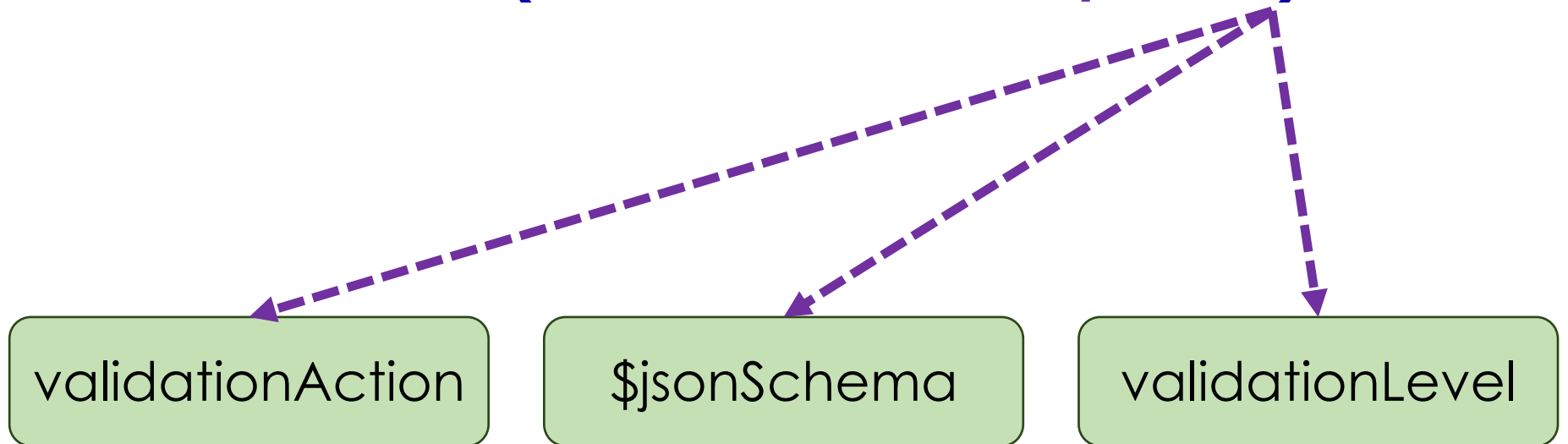




# MongoDB

## À la création de la collection

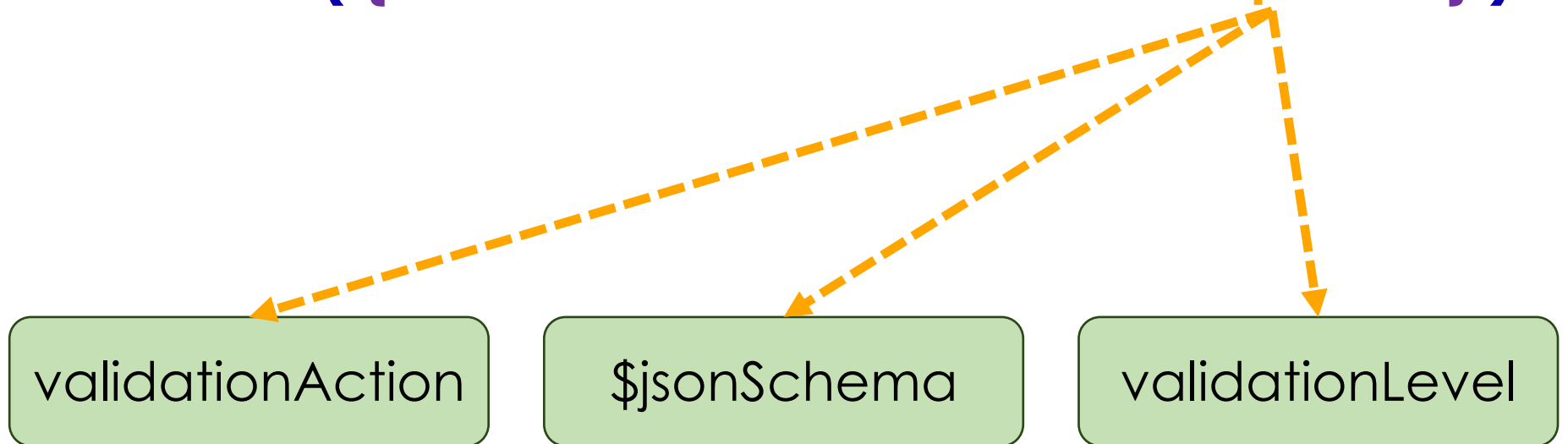
**db.createCollection( nomCollection, options )**



# MongoDB

## À la mise à jour de la collection

```
db.runCommand( { collMod: 'nomCollection', options } )
```



# MongoDB

## Exemple d'un \$jsonSchema

```
use('sample_supplies');

const monSchema = {
  properties : {
    _id: { bsonType: "objectId" },
    saleDate : { bsonType: "date"},
    items : { bsonType: "array"},
    storeLocation : { bsonType: "string"},
    customer : { bsonType: "object"},
    purchaseMethod : { bsonType: "string"},
    couponUsed : { bsonType: "bool"},
  },
  additionalProperties: false
}
```

```
db.sales.find(
  {$jsonSchema: monSchema}
);
```

# MongoDB

## *Exemple 1*

### *validator*

```
db.createCollection('users', {
  validator: {
    $jsonSchema: {
      required: ["name", "lastname", "address"],
      properties: {
        address: {
          bsonType: "object",
          required: ["street", "zip", "city"],
          properties: {
            zip: {
              bsonType: "int"
            }
          }
        }
      }
    }
  },
  validationAction: "warn"
});
```

# MongoDB

## *Exemple II*

### *validator*

```
db.createCollection("contacts", {
  validator: { $jsonSchema: {
    bsonType: "object",
    required: [ "phone" ],
    properties: {
      phone: {
        bsonType: "string",
        description: "must be a string and is required"
      },
      email: {
        bsonType : "string",
        pattern : "@mongodb\\.com$",
        description: "must be a string and end with
 '@mongodb.com' "
      }
    }
  } },
  validationAction: "error"
});
```