Large-Scale and Multi-Structured Databases

Document Databases

Prof. Pietro Ducange

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

UNIVERSITÀ DI PISA

CROSSLAB Innovation for industry 4.0

Main Features

Document databases are ***non-relational*** databases that store data as structured documents, usually in ***XML*** or ***JSON*** formats.

They ensure a ***high flexibility*** level and allow also to handle ***complex*** data ***structures*** (despite key-value databases).

Document databases are ***schema-less.***

They allow ***complex operations***, such as queries and filtering.

Some document databases allows also ***ACID transactions***.

XML Documents

- XML stands for ***eXtensible Markup Language***.
- A markup language specifies the ***structure*** and ***content*** of a ***document***.
- ***Tags*** are added to the document to provide the ***extra information***.
- XML tags give a reader some idea what some of the ***data means***.
- XML is capable of representing almost ***any form*** of information.

XML Documents: Use cases

1. XML and Cascading Style Sheets (CSS) allowed ***second-generation websites*** to separate data and format.

2. XML is also the basis for many data ***interchange protocols*** and, in particular, was a foundation for web service specifications such as ***SOAP*** (Simple Object Access Protocol).

3. XML is a ***standard*** format for many document types, eventually including word processing documents and spreadsheets (***docx***, ***xlsx*** and ***pptx*** formats are based on XML).

Advantages of XML

- XML is text (Unicode) based.
  - Can be transmitted efficiently.
- One XML document can be displayed differently in different media and software platforms.
- XML documents can be modularized. Parts can be reused.

An Example of XML Document

```xml
<item>
   <title>Kind of Blue</title>
   <artist>Miles Davis</artist>
   <tracks>
      <track length="9:22">So What</track>
      <track length="9:46">Freddie Freeloader</track>
      <track length="5:37">Blue in Green</track>
      <track length="11:33">All Blues</track>
      <track length="9:26">Flamenco Sketches</track>
   </tracks>
</item>
<item>
   <title>Cookin'</title>
   <artist>Miles Davis</artist>
   <tracks>
      <track length="5:57">My Funny Valentine</track>
      <track length="9:53">Blues by Five</track>
      <track length="4:22">Airegin</track>
      <track length="13:03">Tune-Up</track>
   </tracks>
</item>
<item>
   <title>Blue Train</title>
   <artist>John Coltrane</artist>
   <tracks>
      <track length="10:39">Blue Train</track>
      <track length="9:06">Moment's Notice</track>
      <track length="7:11">Locomotion</track>
      <track length="7:55">I'm Old Fashioned</track>
      <track length="7:03">Lazy Bird</track>
   </tracks>
</item>
```

XML Ecosystem

XPath: useful to ***navigate*** through elements and attributes in an XML document.

XQuery: is the language for ***querying*** XML data and is built on XPath expressions.

XML schema: A special type of XML document that describes the ***elements*** that may be present in a specified class of XML documents.

XSLT (Extensible Stylesheet Language Transformations): A language for ***transforming*** XML documents into alternative formats, including non-XML formats such as HTML.

DOM (Document Object Model): a platform- and language-neutral interface for dynamically managing the content, structure and style of documents such as XML and XHTML. A document is handled as tree.

Example of XML Scheme Usage

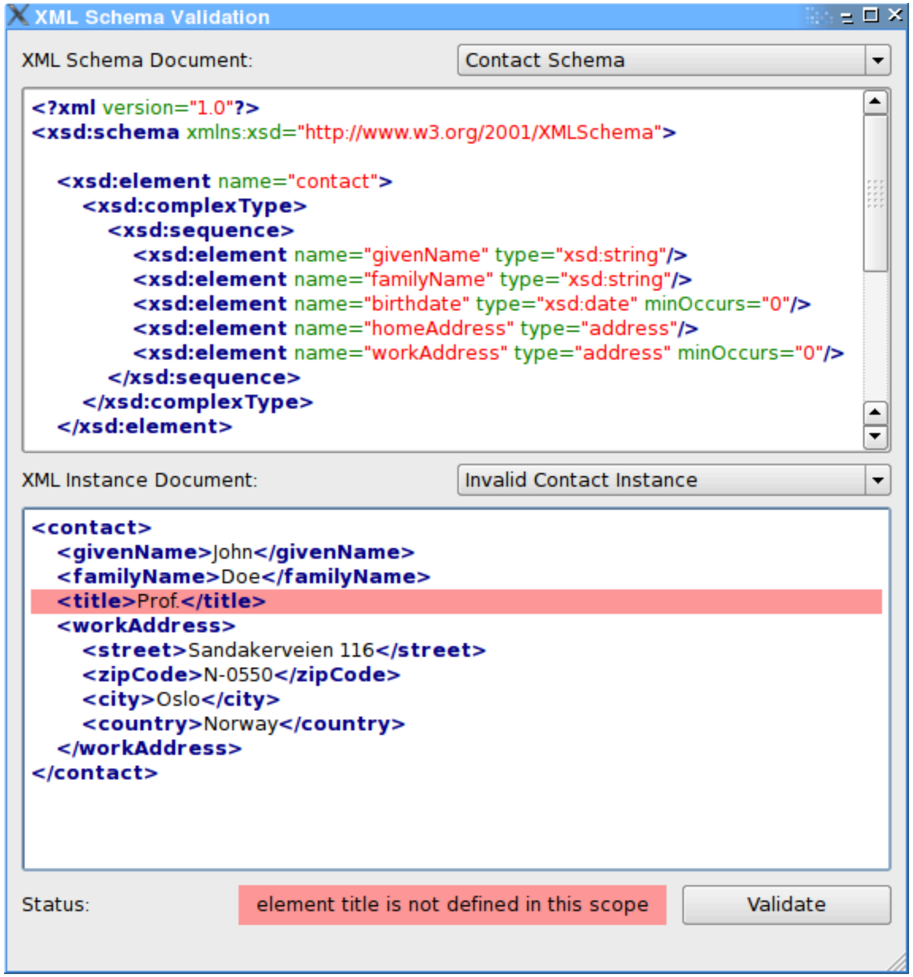
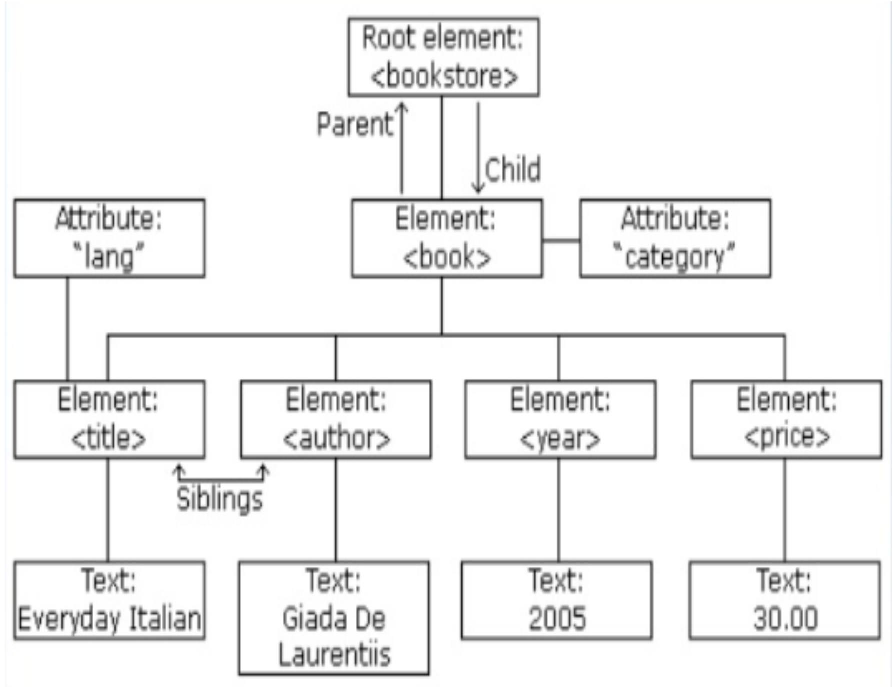


Image extracted from: https://doc.qt.io/qt-5/qtxmlpatterns-schema-example.html

DOM Example

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
  <bookstore>
      <book category="cooking">
          <title lang="en">Everyday Italian</title>
          <author>Giada De Laurentiis</author>
          <year>2005</year>
          <price>30.00</price>
      </book>
      <book category="web" cover="paperback">
          <title lang="en">Learning XML</title>
          <author>Erik T. Ray</author>
          <year>2003</year>
          <price>39.95</price>
      </book>
  </bookstore>
```



DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

UNIVERSITÀ DI PISA

CROSSLAB Innovation for industry 4.0

XML Databases

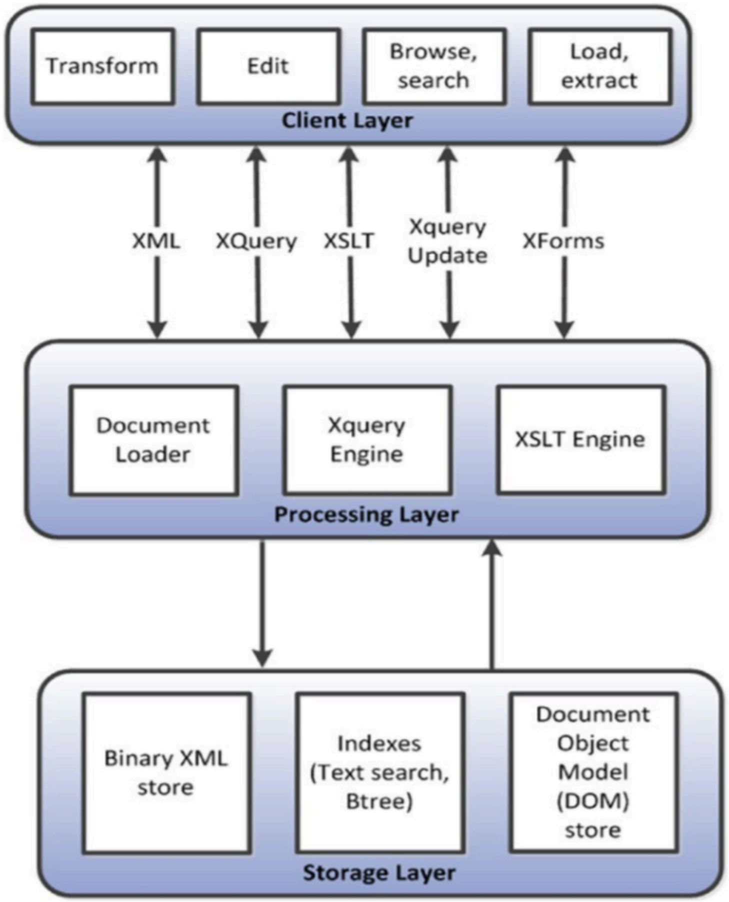


Image extracted from "Guy Harrison, Next Generation Databases, Apress, 2015"

XML databases: ***platforms*** that implement the various XML standards such as XQuery and XSLT,

They provide ***services*** for the ***storage***, ***indexing***, ***security***, and concurrent ***access*** of XML files.

XML databases ***did not represent an alternative*** for RDBMSs.

On the other hand, some RDBMSs introduced and XML, allowing the ***storage of XML*** documents within A BLOB (binary large object) columns.

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

IN SUPREMÆ DIGNITATIS 1343 UNIVERSITÀ DI PISA

CROSSLAB Innovation for industry 4.0

XML : Main Drawbacks

- XML tags are ***verbose*** and ***repetitious***, thus the amount of storage required increases.

- XML documents are ***wasteful of space*** and are also ***computationally expensive*** to parse.

- In general, XML databases are used as ***content-management systems***: collections of text files (such as academic papers and business documents) are organized and maintained in XML format.

- On the other hand, ***JSON-based*** document databases are more suitable to support ***web-based*** operational ***workloads***, such as storing and modifying dynamic contents.

JSON Documents

- JSON acronym of ***JavaScript Object Notation.***
- Used to format data.
- Thanks to its integration with JavaScript, a JSON document has been often preferred to a XML for data interchanging on the Internet.

JSON example

- "JSON" stands for "JavaScript Object Notation"
  - Despite the name, JSON is a (mostly) language-independent way of specifying objects as name-value pairs

```
{"skillz": {
          "web":[
                   {"name": "html",
                    "years": "5"
                   },
                   {"name": "css",
                    "years": "3"
                   }],
          "database":[
                   {"name": "sql",
                    "years": "7"
                   }]
}}
```

Image extracted from http://secretgeek.net/json_3mins

JSON syntax

- An ***object*** is an unordered set ***of name/value pairs***
  - The pairs are enclosed within braces, { }
  - There is a colon between the name and the value
  - Pairs are separated by commas
  - Example: { "name": "html", "years": 5 }
- An ***array*** is an ***ordered collection*** of values
  - The values are enclosed within brackets, [ ]
  - Values are separated by commas
  - Example: ["html", ”xml", "css"]

JSON syntax

- A *value* can be: A string, a number, true, false, null, an object, or an array
  - Values can be nested
- *Strings* are enclosed in double quotes, and can contain the usual assortment of escaped characters
- *Numbers* have the usual C/C++/Java syntax, including exponential (E) notation
  - All numbers are decimal--***no octal or hexadecimal***

Example of Nested Objects

```json
{
  "sammy" : {
    "username"  : "SammyShark",
    "location"  : "Indian Ocean",
    "online"    : true,
    "followers" : 987
  },
  "jesse" : {
    "username"  : "JesseOctopus",
    "location"  : "Pacific Ocean",
    "online"    : false,
    "followers" : 432
  },
  "drew" : {
    "username"  : "DrewSquid",
    "location"  : "Atlantic Ocean",
    "online"    : false,
    "followers" : 321
  },
  "jamie" : {
    "username"  : "JamieMantisShrimp",
    "location"  : "Pacific Ocean",
    "online"    : true,
    "followers" : 654
  }
}
```

Image extracted from: https://www.digitalocean.com/community/tutorials/an-introduction-to-json

Example of Nested Arrays

```
{
  "first_name" : "Sammy",
  "last_name" : "Shark",
  "location" : "Ocean",
  "websites" : [
    {
      "description" : "work",
      "URL" : "https://www.digitalocean.com/"
    },
    {
      "desciption" : "tutorials",
      "URL" : "https://www.digitalocean.com/community/tutorials"
    }
  ],
  "social_media" : [
    {
      "description" : "twitter",
      "link" : "https://twitter.com/digitalocean"
    },
    {
      "description" : "facebook",
      "link" : "https://www.facebook.com/DigitalOceanCloudHosting"
    },
    {
      "description" : "github",
      "link" : "https://github.com/digitalocean"
    }
  ]
}
```

Image extracted from: https://www.digitalocean.com/community/tutorials/an-introduction-to-json

Comparison of JSON and XML

- ***Similarities***:
  - Both are human readable
  - Both have very simple syntax
  - Both are hierarchical
  - Both are language independent
  - Both supported in APIs of many programming languages
- ***Differences***:
  - Syntax is different
  - JSON is less verbose
  - JSON includes arrays
  - Names in JSON must not be JavaScript reserved words

JSON vs XML

users.xml

```xml
<users>
    <user>
        <username>SammyShark</username> <location>Indian Ocean</location>
    </user>
    <user>
        <username>JesseOctopus</username> <location>Pacific Ocean</location>
    </user>
    <user>
        <username>DrewSquir</username> <location>Atlantic Ocean</location>
    </user>
    <user>
        <username>JamieMantisShrimp</username> <location>Pacific Ocean</location>
    </user>
</users>
```

users.json

```json
{"users": [
  {"username" : "SammyShark", "location" : "Indian Ocean"},
  {"username" : "JesseOctopus", "location" : "Pacific Ocean"},
  {"username" : "DrewSquid", "location" : "Atlantic Ocean"},
  {"username" : "JamieMantisShrimp", "location" : "Pacific Ocean"}
] }
```

Image extracted from: https://www.digitalocean.com/community/tutorials/an-introduction-to-json

Main Feature of JSON Databases

- Data is stored in JSON format.
- A ***document*** is the basic ***unit*** of storage. It includes one or more more ***key-value pairs***, and may also contain ***nested documents*** and ***arrays***.
- ***Arrays*** may also ***contain documents*** allowing for a complex hierarchical structure.
- A ***collection*** or ***data bucket*** is a set of documents sharing some ***common purpose***.
- ***Schema less***: predefined document elements must not be defined.
- ***Polymorphic Scheme***: the documents in a collection may be ***different***.

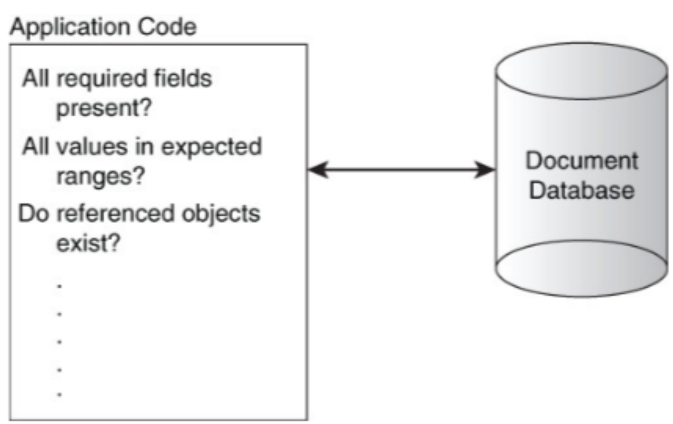
Schema-less pros and cons

Pros: High flexibility in handling the structure of the objects to store

```
{  'employeeName' :  'Janice Collins',
   'department' : 'Software engineering'
   'startDate' : '10-Feb-2010',
   'pastProjectCodes' : [ 189847, 187731, 176533, 154812]
}
```

```
{   'employeeName'  :   'Robert Lucas,
    'department' : 'Finance'
    'startDate' : '21-May-2009',
    'certifications' : 'CPA'
}
```

Cons: the DBMS may be not allowed to enforce rules based on the structure of the data.



Some considerations

A *document database* could ***theoretically*** implement a ***third normal form schema***.

Tables, as in relational databases, may be "***simulated***" considering collections with ***JSON*** documents with an identical ***pre-defined structure***.

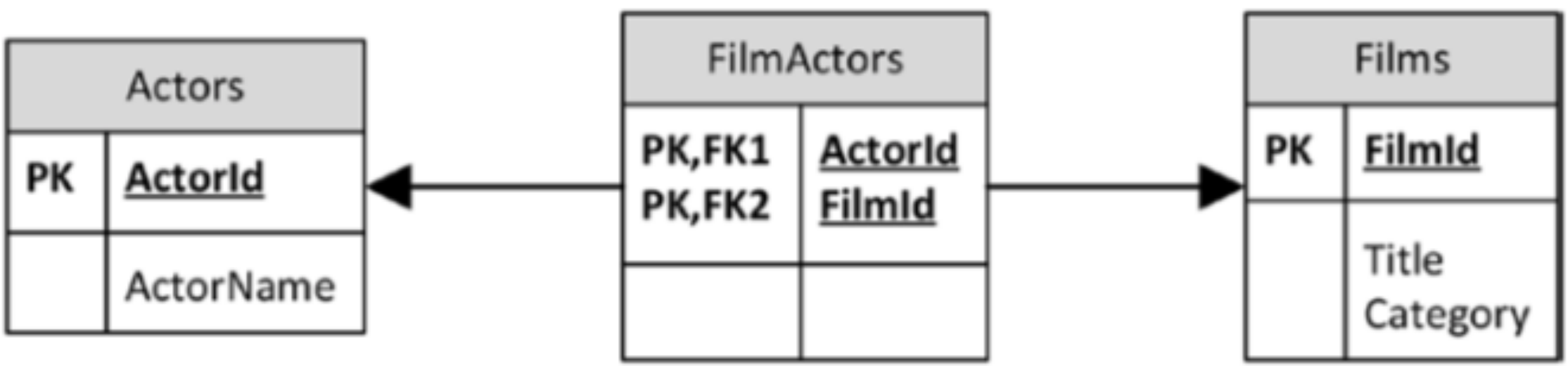


Image extracted from "Guy Harrison, Next Generation Databases, Apress, 2015"

JSON Databases: An example

Document databases usually adopts a ***reduced number*** of collections for modeling data.

Nested documents are used for representing ***relationships*** among the different entities.

Document databases ***do not*** generally provide ***join operations***.

Programmers like to have the JSON structure map ***closely to the object*** structure of their code!!!

Image extracted from "Guy Harrison, Next Generation Databases, Apress, 2015"

Data Modeling: Document Embedding

Film Document

Actor document

Array of actors

```
{ "_id": 115,"Title": "CAMPUS REMEMBER",
              "Category": "Action",
    "Actors" :
        [
          { "actorId": 8,"Name": "MATTHEW JOHANSSON" },
          { "actorId": 45,"Name": "REESE KILMER" },
          { "actorId": 168,"Name": "WILL WILSON" }
        ]
}
```

The solution above allows the user to ***retrieve*** a film and all its actors in a ***single operation***.

However, “actors” result to be ***duplicated*** across multiple documents.

In a complex design this could lead to ***issues*** and possibly ***inconsistencies*** if any of the “actor” attributes need to be changed.

Moreover, some JSON databases have some ***limitations*** of the maximum ***dimension*** of a single document.

Data Modeling: Document Linking

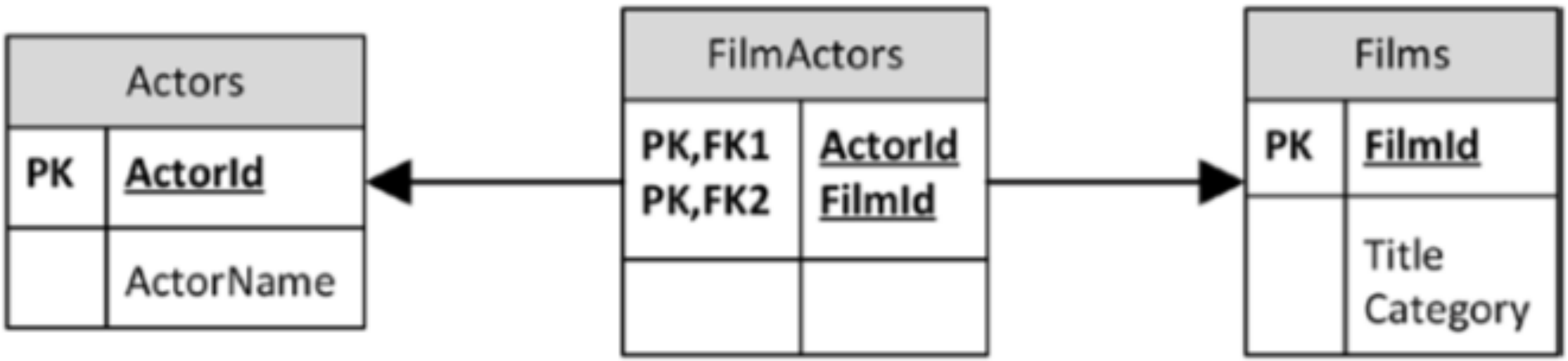
In the solution above, an array of actor ***IDs*** has been embedded into the film document.

The IDs can be used to ***retrieve*** the documents of the actors (in on other collection) who appear in a film.

We are ***rolling back*** to a relational model!!! Now, at least two collections of document must be defined.

Data Modeling: Document Linking

We are ***rolling back*** to a the third normal form!!!!!



Data Modeling: Some Discussions

Document linking approaches are usually somewhat an ***unnatural*** style for a document database.

However, for ***some workloads*** it may provide the best ***balance*** between performance and maintainability.

When modeling data for document databases, there is no equivalent of third normal form that defines a "***correct***" model.

In this context, the ***nature of the queries*** to be executed ***drives*** the approach to ***model*** data.

Data Modeling: An Example (I)

Scenario: Trucks in a company fleet have to transmit location, fuel consumption and other metrics every three minutes to a fleet management data base (one-to-many relationship between a truck and the transmitted details)

We may consider to generate a new document to add to the DB for each data transmission.

Let consider a document as follows:

```
{
    truck_id: 'T87V12',
    time: '08:10:00',
    date :  '27-May-2015',
    driver_name: 'Jane Washington',
    fuel_consumption_rate: '14.8 mpg',
    …
}
```

Repeated information in each new document

At the end of the day, the DB will include 200 new documents for each truck (we consider 20 transmissions per hour, 10 working hours)

Data Modeling: An Example (II)

An alternative solution may be to use embedded documents as follows:

```
{
   truck_id: 'T87V12',
   date :  '27-May-2015',
   driver_name: 'Jane Washington',
   operational_data:
                [
                   {time : '00:01',
                    fuel_consumption_rate: '14.8 mpg',
                    …},
                     {time : '00:04',
                    fuel_consumption_rate: '12.2 mpg',
                    …},
                     {time : '00:07',
                    fuel_consumption_rate: '15.1 mpg',
                    …},
                 …]
}
```

Pay attention: we can have a potential ***performance problem***!

Data Modelling:
Many to Many Relationships

Let consider an example of application in which:

- A ***student*** can be enrolled in many courses
- A ***course*** can have many students enrolled to it

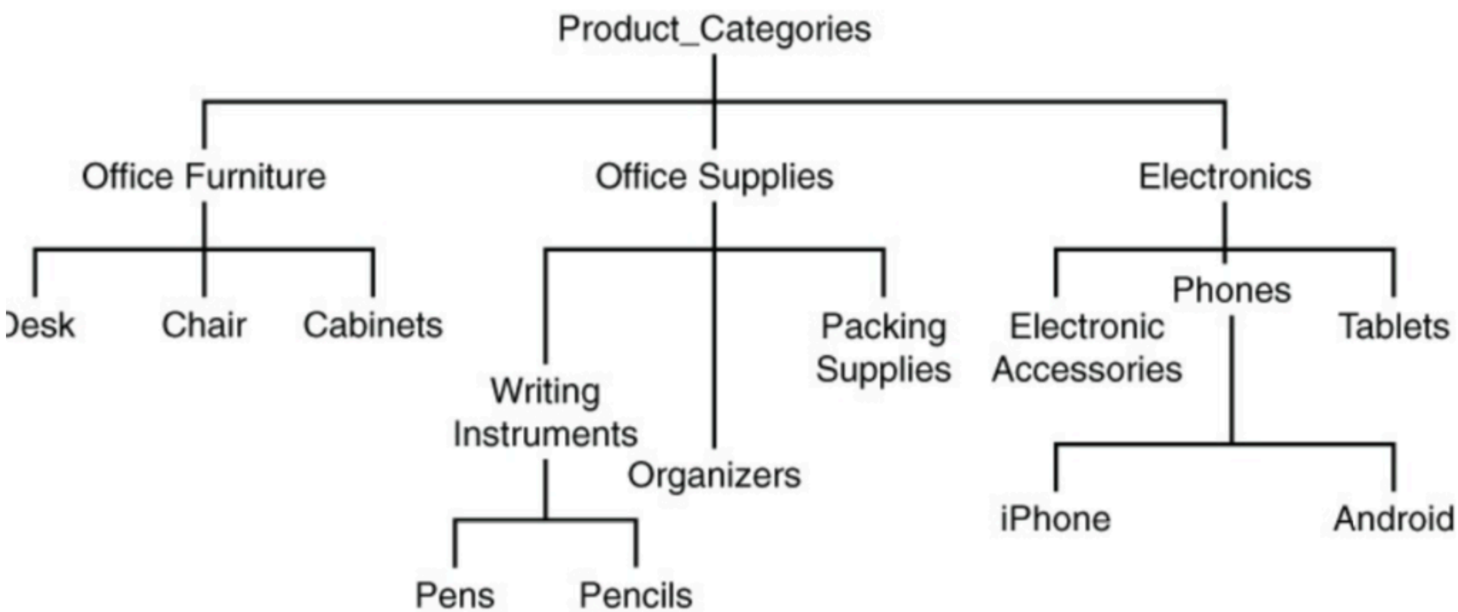
We can model this situation considering the following two collections:

```
{
  { courseID: 'C1667',
     title: 'Introduction to Anthropology',
     instructor: 'Dr. Margret Austin',
     credits: 3,
     enrolledStudents: ['S1837', 'S3737', 'S9825' …
       'S1847'] },
  { courseID: 'C2873',
     title: 'Algorithms and Data Structures',
     instructor: 'Dr. Susan Johnson',
     credits: 3,
     enrolledStudents: ['S1837','S3737', 'S4321', 'S9825'
       … 'S1847'] },
  { courseID: C3876,
     title: 'Macroeconomics',
     instructor: 'Dr. James Schulen',
     credits: 3,
     enrolledStudents: ['S1837', 'S4321', 'S1470', 'S9825'
       … 'S1847'] },
```

```
{
 {studentID:'S1837',
   name: 'Brian Nelson',
   gradYear: 2018,
   courses: ['C1667', C2873,'C3876']},
 {studentID: 'S3737',
   name: 'Yolanda Deltor',
         gradYear: 2017,
         courses: [ 'C1667','C2873']},
      …
}
```

We have to take care when updating data in this kind of relationship. Indeed, the DBMS will ***not*** control the ***referential integrity*** as in relational DBMSs.

Modeling Hierarchies (I)

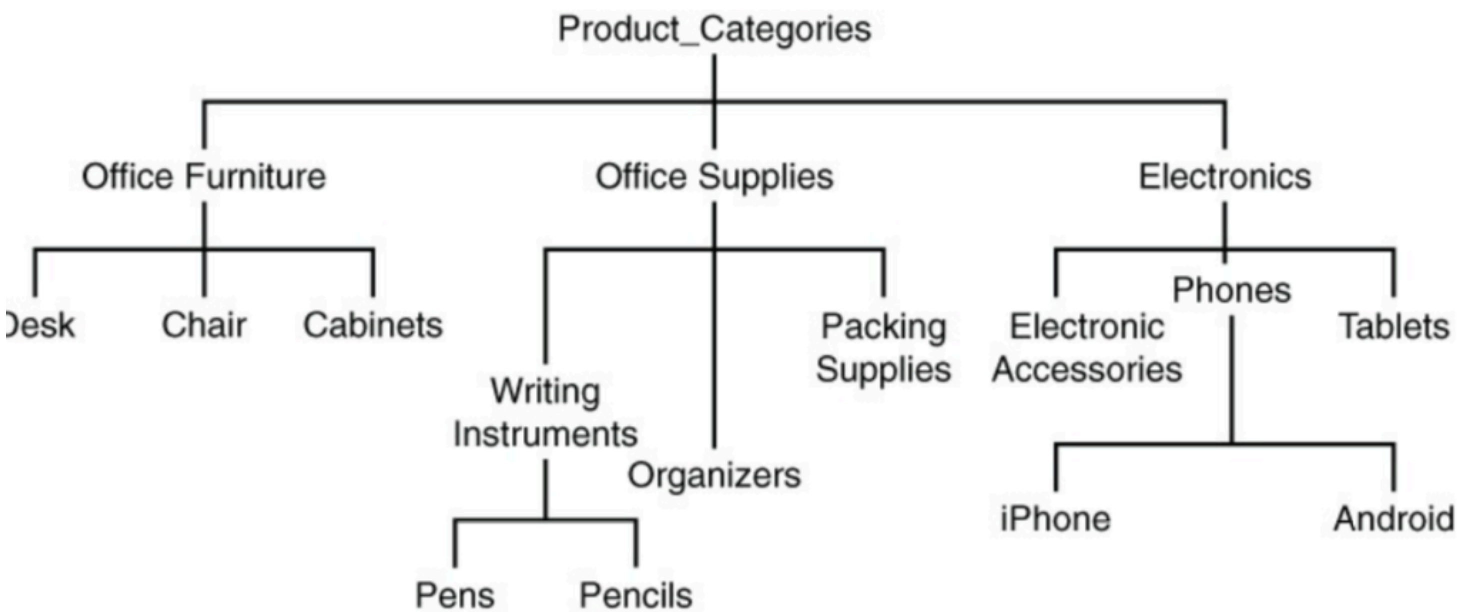


Parent reference solution:

```
{
    {productCategoryID: 'PC233', name:'Pencils',
      parentID:'PC72'},
    {productCategoryID: 'PC72', name:'Writing Instruments',
      parentID: 'PC37"},
    {productCategoryID: 'PC37', name:'Office Supplies',
      parentID: 'P01'},
    {productCategoryID: 'P01', name:'Product Categories' }
}
```

This solution is useful if we have frequently show a specific instance of an object and then show the more general type of that category.

Modeling Hierarchies (II)

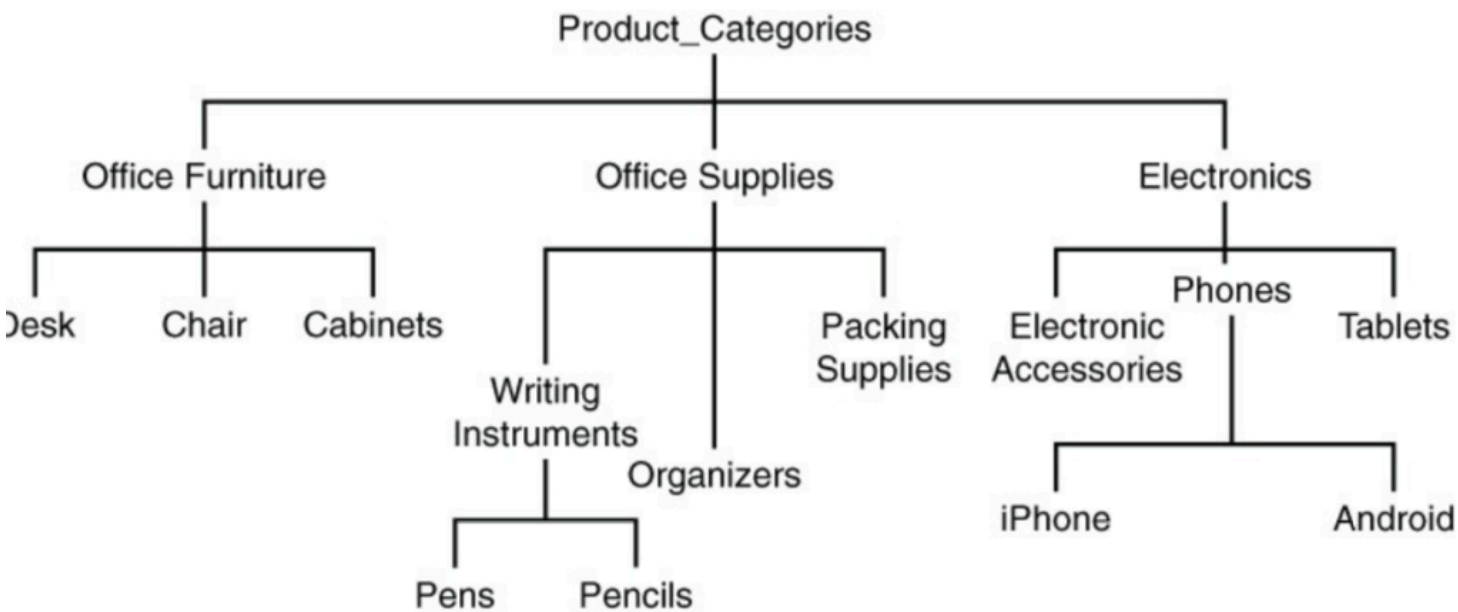


Child reference solution:

```
{
  {productCategoryID: 'P01', name:'Product Categories',
       childrenIDs: ['P37','P39','P41']},
      {productCategoryID: 'PC37', name:'Office Supplies',
        childrenIDs: ['PC72','PC73','PC74"]},
       {productCategoryID: 'PC72', name:'Writing
         Instruments', childrenIDs: ['PC233','PC234']'},
        {productCategoryID: 'PC233', name:'Pencils'}
   }
```

This solution is useful if we have frequently retrieve the children (or sub parts) of a specific instance of an object.

Modeling Hierarchies (III)



List of ancestor solutions:

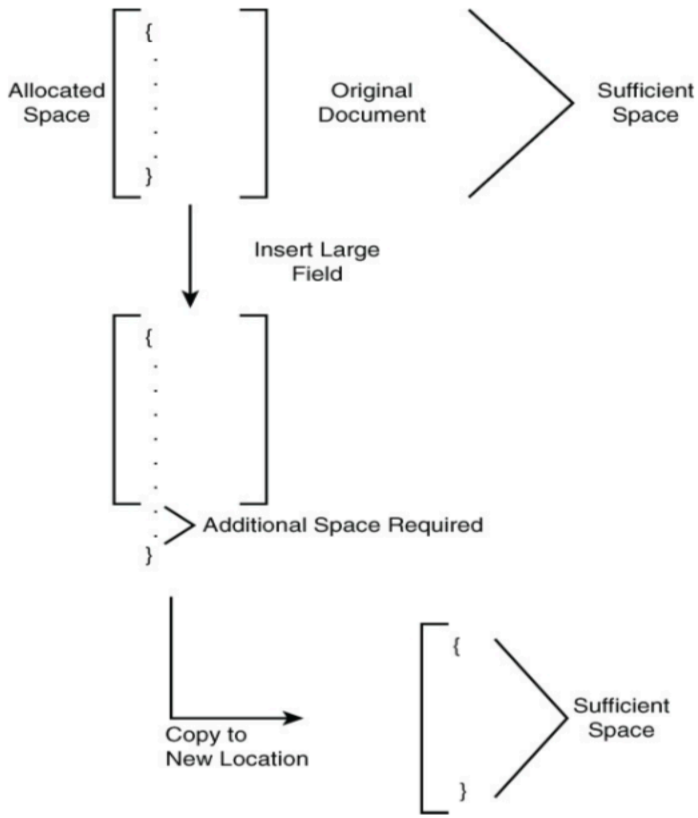
```
{productCategoryID: 'PC233', name:'Pencils',
  ancestors:['PC72', 'PC37', 'P01']}
```

This solution allows to retrieve the full path of the ancestors with one read operation.

A change to the hierarchy may require many write operations, depending on the level at which the change occurred.

Planning for Mutable Documents (I)

When a document is created, the DBMS allocates a certain amount of spaces for the document.



If the document ***grows more*** than the allocated space, the DBMS has to ***relocate*** it to another location.

Moreover, the DBMS ***must free*** the previously allocated space.

The previous steps can adversely affect the system performance.

Planning for Mutable Documents (II)

A solution for avoiding to move oversized document is to ***allocate sufficient space*** at the moment in which the document is ***created***.

Regarding the previous problem, the following solution may be adopted:

```
{truck_id: 'T8V12'
 date: '27-May-2015'
 operational_data:
    [{time: '00 : 00',
      fuel_consumption_rate: 0.0}
     {time: '00 : 00',
      fuel_consumption_rate: 0.0}
      .
      .
      .
      .
     {time: '00 : 00',
      fuel_consumption_rate: 0.0}
      ]
```

200 Embedded Documents with Default Values

In conclusion, we have to consider the ***life cycle*** of a document and planning, if possible, the strategies for handling its growing.

Indexing Document Database

In order to avoid the entire scan of the overall database, DBMSs for document databases (for example MongoDB) allow the definition of ***indexes***.

Indexes, like in book indexes, are a ***structured set of information*** that maps from one attribute to related information.

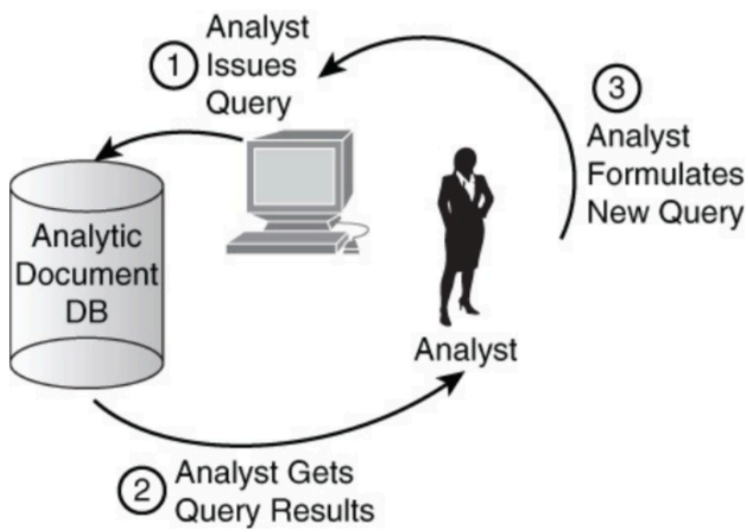
In general, indexes are ***special data structures*** that store a small portion of the collection's data set in an ***easy to traverse*** form.

The index stores the value of a specific field or set of fields***, ordered by the value of the field***.

The ordering of the index entries supports ***efficient equality matches*** and ***range-based query operations***.

Read-Heavy Applications

In the figure we show the classical scheme of a read-heavy application (***business intelligence*** and ***analytics applications***):



In this kind of applications, the use ***of several indexes*** allows the uses to quickly access to the database. For example, indexes can be defined for easily retrieve documents describing objects related to a specific ***geographic region*** or to a specific ***type***.

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

UNIVERSITÀ DI PISA

CROSSLAB Innovation for industry 4.0

Write-Heavy Applications

The example of the truck information transmission (each three minutes) is a typical write-heavy application.

The ***higher*** the number of ***indexes*** adopted the ***higher*** the amount of ***time*** required for closing a write operation.

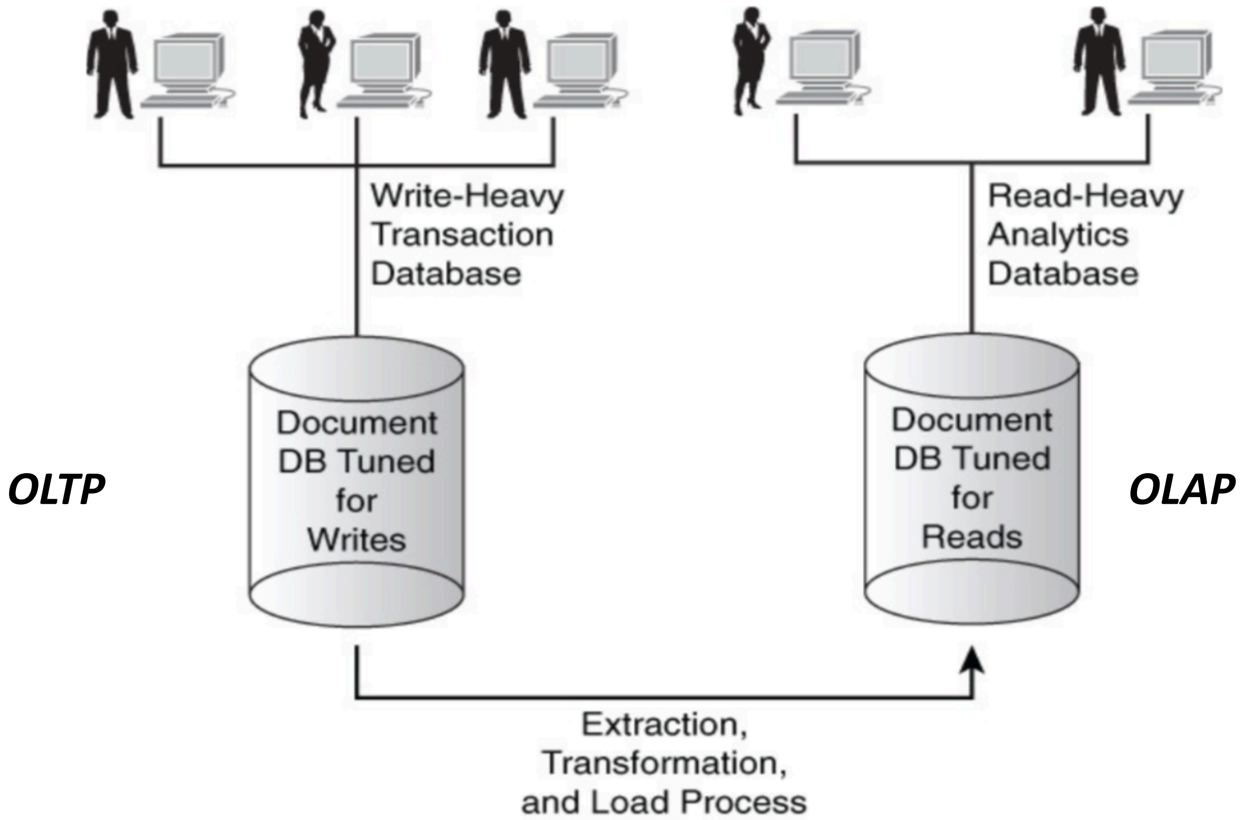
Indeed, all the ***indexes*** must be ***updated*** (and created at the beginning).

Reducing the number of indexes, allow us to obtain systems with ***fast write*** operation responses. On the other hand, we have to accept to deal with ***slow read operations***.

In conclusion, the number and the type of indexes to adopt must be identified as a ***trade-off*** solution.

Transactions Processing Systems

These systems are designed for fast write operation and targeted reads, as shown in the figure below:



Sharding

- ***Sharding*** or ***horizontal partitioning*** is the process of dividing data into blocks or ***chunks***.
- Each block, labeled as shard, is deployed on a ***specific node*** (server) of a ***cluster***.
- Each node can contain ***only one*** shard.
- In case of ***data replication***, a shard can be hosted by more than one node.

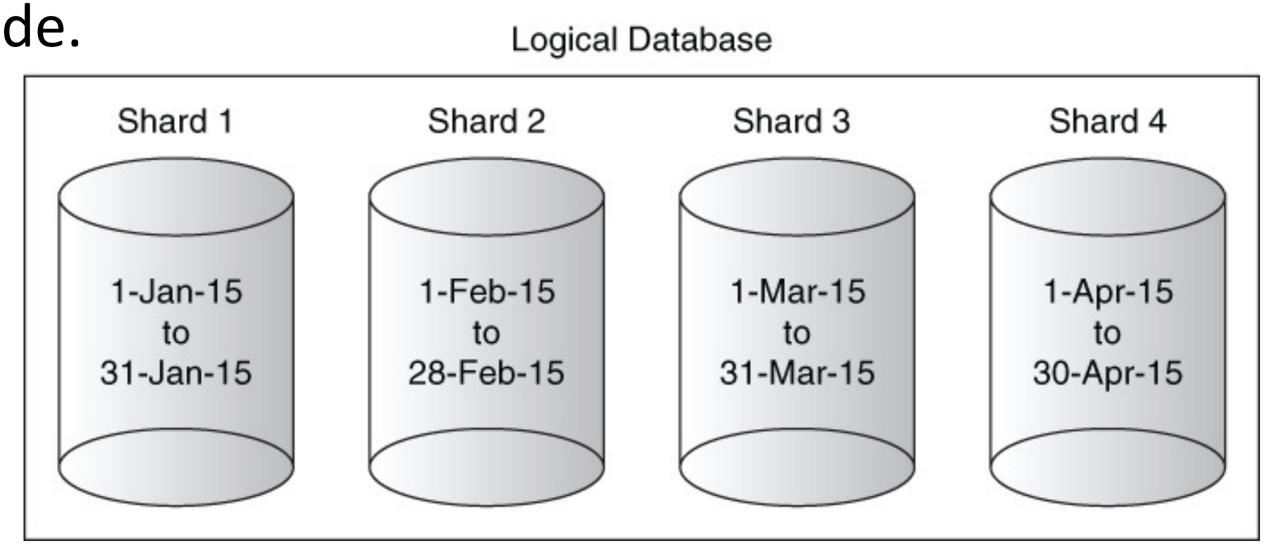


Image extracted from: "Dan Sullivan, NoSQL For Mere Mortals, Addison-Wesley, 2015"

Advantages of Sharding

- Allows handling ***heavy loads*** and the ***increase*** of system users.
- Data may be ***easily distributed*** on a variable number of servers that may be ***added*** or ***removed*** by request.
- Cheaper than ***vertical scaling*** (adding ram and disks, upgrading CPUs to a single server).
- Combined with ***replications***, ensures a ***high availability*** of the system and ***fast*** responses.

Shard Keys

To implement sharding, document database designers have to select a ***shard key*** and a ***partitioning method***.

A shard key is ***one or more*** fields that exist ***in all documents*** in a collection that is used to separate documents.

Examples of shard keys may be: Unique ***document ID***, ***Name***, ***Date***, such as creation date, ***Category*** or type, Geographical region.

Actually, ***any atomic field*** in a document may be chosen as a shard key.

Partition Algorithms

There are three main categories of partition algorithms, based on:

- ***Range***: for example, if all documents in a collection had a creation date field, it could be used to partition documents into monthly shards.
- ***Hashing***: a hash function can be used to determine where to place a document. ***Consistent hashing*** may be also used.
- ***List***: for example, let imagine a product database with several types (electronics, appliances, household goods, books, and clothes). These product types could be used as a shard key to allocate documents across five different servers.

Suggested Readings

Chapter 4 of the book "*Guy Harrison, Next Generation Databases, Apress, 2015*".

Chapters 6,7,8 of the book "*Dan Sullivan, NoSQL For Mere Mortals, Addison-Wesley, 2015*"