

Development of Web Applications

Principles and Practice

Vincent Simonet, 2013-2014

Université Pierre et Marie Curie, Master Informatique, Spécialité STL

3

Server Technologies

Vincent Simonet, 2013-2014

Université Pierre et Marie Curie, Master Informatique, Spécialité STL

Today's agenda

- Tasks of the Web Server,
 - Java Servlets,
 - Java Server Pages,
 - Persistence and Data Storage.
-

Tasks of the Web Server

HTTP Server Basics

The web server is a site-independent piece of software that:

- Handles individual HTTP requests, and
- Generates HTTP responses.

Some common features:

- Virtual hosting,
 - Large file support,
 - Bandwidth throttling,
 - Server-side scripting.
-

How to manage concurrency?

The HTTP server generally handles HTTP requests separately. Concurrency / synchronization is handled at the level of the database:

- It is used to share information between all requests,
 - It ensures the consistency via the transaction model.
-

CGI Common Gateway Interface

The historic method for web server software to delegate the generation of web content to executable files.

Introduced in 1993 (www-talk), formalized in 1997 ([RFC 3875](#)).

Web Servers like Apache support CGI scripts in plenty of different languages (Perl, Python, C++, Java, etc.)

CGI: Example Script

```
#!/usr/bin/perl

print "Content-type: text/plain\r\n\r\n";

for my $var ( sort keys %ENV ) {
    my $value = $ENV{$var};
    $value =~ s/\n/\\n/g;
    $value =~ s/"/\\"/g;
    print qq[$var="$value"\n];
}
```

CGI: Example output

```
GATEWAY_INTERFACE="CGI/1.1"
HTTP_ACCEPT="text/html,application/xhtml+xml,application/xml;q=0.9,
*/*;q=0.8"
HTTP_ACCEPT_CHARSET="ISO-8859-1,utf-8;q=0.7,*;q=0.7"
HTTP_ACCEPT_ENCODING="gzip, deflate"
HTTP_ACCEPT_LANGUAGE="en-us,en;q=0.5"
HTTP_CONNECTION="keep-alive"
HTTP_HOST="example.com"
HTTP_USER_AGENT="Mozilla/5.0 (Windows NT 6.1; WOW64; rv:5.0)
Gecko/20100101 Firefox/5.0"
QUERY_STRING="var1=value1&var2=with%20percent%20encoding"
REMOTE_ADDR="127.0.0.1"
REMOTE_PORT="63555"
REQUEST_METHOD="GET"
REQUEST_URI="/cgi-bin/printenv.pl/foo/bar?var1=value1&var2=with%
20percent%20encoding"
SERVER_NAME="127.0.0.1"
SERVER_PORT="80"
SERVER_PROTOCOL="HTTP/1.1"
SERVER_SOFTWARE="Apache/2.2.19 (Win32) PHP/5.2.17"
```

CGI limitations and alternatives

In the original CGI approach, a new process is created for every HTTP request calling the CGI script. This results in a huge overhead, especially if the script needs to be interpreted or compiled.

Several approaches are possible for remedying this:

- Extension mechanisms that allows third-party software to run inside the server itself, such as [Apache modules](#),
 - [FastCGI](#) allows a single, long-running process to handle more than one user request while keeping close to the CGI programming model,
 - [Simple CGI](#) (SCGI), similar to FastCGI,
 - A more integrated model, like Java Servlets.
-

Java Servlets

Definition

The servlet is a Java class used to extend the capabilities of a server.

They can respond to any types of requests, but they are commonly used to extend HTTP servers.

Created by Sun Microsystems in 1997. Servlet specification v3.0 released in 2009.

Servlet Container

The component of a web server that interacts with the servlets. The web container is responsible for managing the lifecycle of servlets, mapping a URL to a particular servlet and ensuring that the requester has the correct access rights.

The Servlet API, contained in the Java package hierarchy `javax.servlet`, defines the expected interactions of the web container and a servlet.

Life cycle of a Servlet (1/4)

1. Assume that a user requests to visit a URL.
 - The browser then generates an HTTP request for this URL.
 - This request is then sent to the appropriate server.
 2. The HTTP request is received by the web server and forwarded to the servlet container.
 - The container maps this request to a particular servlet.
 - The servlet is dynamically retrieved and loaded into the address space of the container.
-

Life cycle of a Servlet (2/4)

3. The container invokes the `init()` method of the servlet.
 - This method is invoked only when the servlet is first loaded into memory.
 - It is possible to pass initialization parameters to the servlet so that it may configure itself.
 4. The container invokes the `service()` method of the servlet.
 - This method is called to process the HTTP request.
 - The servlet may read data that has been provided in the HTTP request.
 - The servlet may generate an HTTP response.
-

Life cycle of a Servlet (3/4)

5. The servlet remains in the container's address space and is available to process any other HTTP requests received from clients.
 - The service() method is called for each HTTP request.
 6. The container may, at some point, decide to unload the servlet from its memory.
 - The algorithms by which this decision is made are specific to each container.
-

Life cycle of a Servlet (4/4)

○

8. The container calls the servlet's `destroy()` method to relinquish any resources such as file handles that are allocated for the servlet; important data may be saved to a persistent store.
 9. The memory allocated for the servlet and its objects can then be garbage collected.
-

HttpServlet Interface

[Source file](#)

```
public abstract class HttpServlet extends
GenericServlet {
    public HttpServlet();

    protected void doGet(HttpServletRequest req,
                        HttpServletResponse resp)
        throws ServletException, IOException;

    // Similar methods doHead, doPost, doPut,
    // doDelete, doOptions and doTrace

    protected void service(HttpServletRequest req,
                        HttpServletResponse resp)
        throws ServletException, IOException;
}
```

HttpServletRequest Interface

[Source file](#)

```
public interface HttpServletRequest extends ServletRequest {  
    public Cookie[] getCookies();  
    public String getHeader(String name);  
    public String getParameter(String name);  
    public BufferedReader getReader() throws IOException;  
    public Collection<Part> getParts() throws IOException,  
        IllegalStateException, ServletException;  
    ...  
}
```

HttpServletResponse Interface

[Source file](#)

```
public interface HttpServletResponse extends HttpServletResponse {
    public void addCookie(Cookie cookie);
    public String encodeURL(String url);
    public void sendError(int sc, String msg) throws IOException;
    public void sendRedirect(String location) throws IOException;
    public void setHeader(String name, String value);
    public void setStatus(int sc);
    public void setContentType(String type);
    public ServletOutputStream getOutputStream()
        throws IOException;
    public PrintWriter getWriter() throws IOException;
    ...
}
```

Servlet Example

```
public class ExampServlet extends HttpServlet {
    public void doPost(HttpServletRequest request,
                        HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

        out.println("<title>Example</title><body>");
        out.println("<h2>Button Clicked</h2>");
        String DATA = request.getParameter("DATA");
        if(DATA != null){
            out.println(DATA);
        } else {
            out.println("No text entered.");
        }
        out.println("<p>Return to <a href='index.html'>home</a>");
        out.close();
    }
}
```

Asynchronous Processing

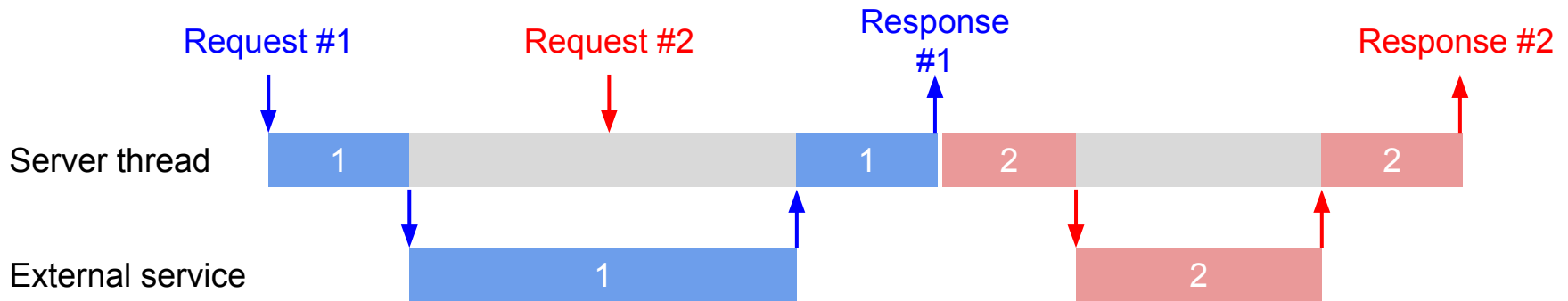
In regular Java Servlets, the processing of a request blocks the execution of the server thread. This is not efficient even if the server is multi-threaded.

Asynchronous Java Servlets allow to call external threads/services without blocking the server thread.

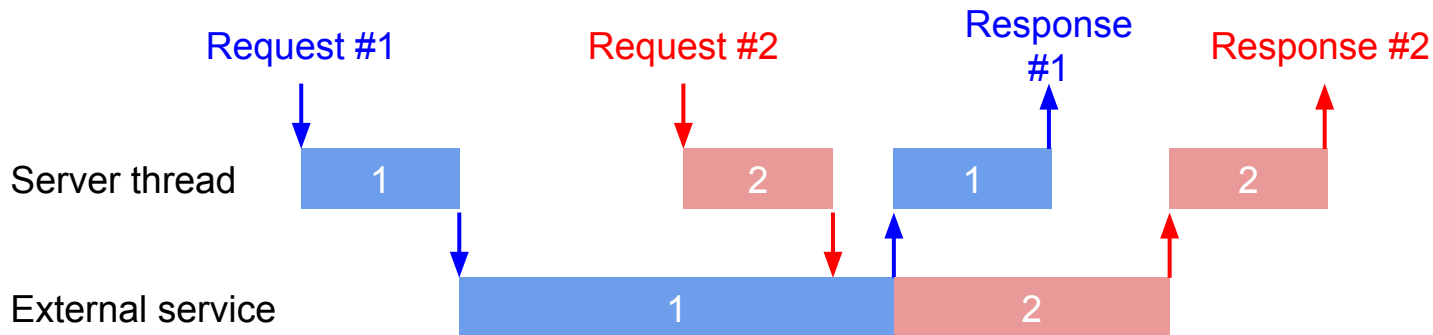
See [tutorial](#).

Synchronous vs Asynchronous

Synchronous Servlet



Asynchronous Servlet



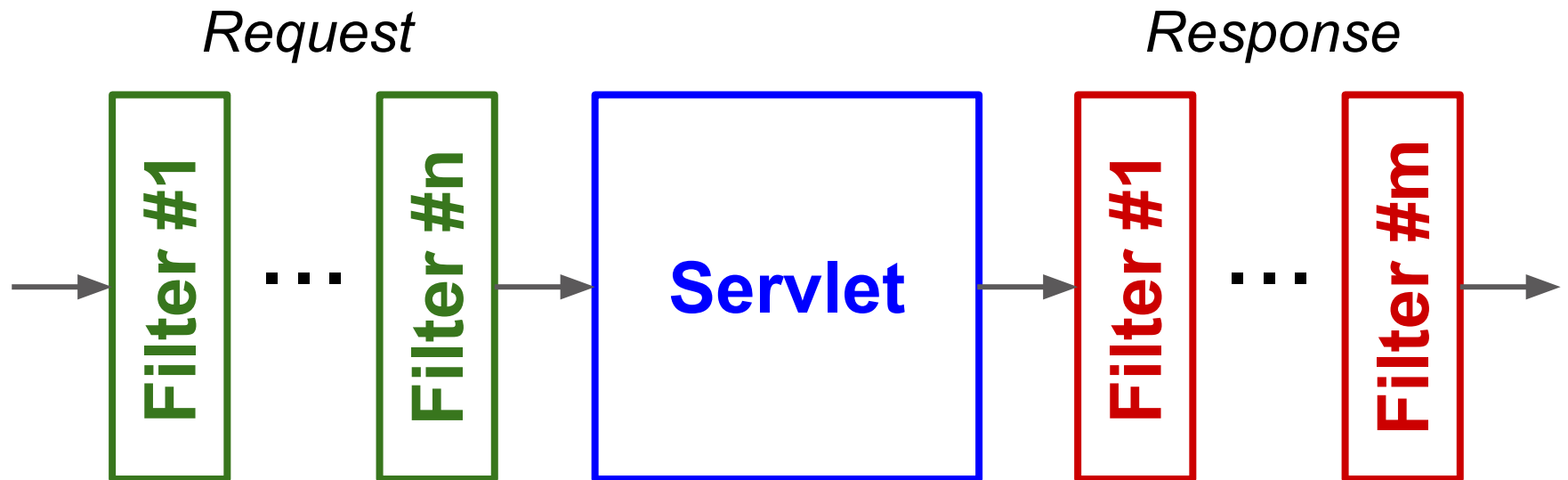
Filters

A filter is a reusable piece of code that modifies or adapt requests or responses of a servlet.

Examples:

- Authentication,
 - Logging and auditing,
 - Image conversion,
 - Data compression,
 - Encryption,
 - Caching, etc.
-

Filters (cont.)



Sessions

Sessions allow storing information on server side between several requests (despite the stateless nature of the HTTP protocol).

A session can be tracked either:

- by a cookie (sent by the server in a response, and returned by the browser at every request), or
 - by URL rewriting (adding a session ID at the end of every URL).
-

HttpSession

The class `HttpSession` provides a high-level interface built on top of cookies or URL-rewriting.

You can get the `HttpSession` object associated with a request using `request.getSession(true)`

Example of session

```
public class SessionCount extends HttpServlet {
    public void doGet(HttpServletRequest request,
                      HttpServletResponse response)
        throws ServletException, IOException {
        HttpSession session = request.getSession(true);
        response.setContentType("text/text");
        PrintWriter out = response.getWriter();
        Integer count = new Integer(0);
        if (session.isNew()) {
            out.println("Welcome, Newcomer");
        } else {
            out.println("Welcome Back");
            Integer previousCount =
                (Integer) session.getValue("count");
            if (oldAccessCount != null) {
                count =
                    new Integer(previousCount.intValue() + 1);
            }
        }
        session.putValue("count", count);
        out.println("Counter: " + count.toString());
    }
}
```

Common issues with sessions

- Threading,
 - Distribution,
 - Sharing on client side.
-

Apache Tomcat

Open Source Web Server and Servlet Container, implementing the Java Servlet and the JSP specifications.

Main components:

- **Catalina:** Servlet Container,
- **Coyote:** HTTP connector,
- **Jasper:** JSP engine,
- **Cluster:** load balancing.

Version 3 (first) in 1999, version 7 (last) in 2011.

Web application

A web application is a set of Servlet classes and possibly other resources which are packed into a .war file, and served by a servlet container.

The overall definition of the web application is contained in a web.xml file.

web.xml

```
<web-app xmlns="..."
          xmlns:xsi="..."
          xsi:schemaLocation="..."
          version="3.0"
          metadata-complete="true">
  <servlet>
    <servlet-name>MyServlet</servlet-name>
    <servlet-class>example.MyServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>MyServlet</servlet-name>
    <url-pattern>/my</url-pattern>
  </servlet-mapping>
</web-app>
```

JSP

Java Server Pages

Definition

A technology that helps software developers create dynamically generated web pages based on HTML, XML, or other document types.

Similar to PHP, but using Java and a servlet container.

JSP are converted into Servlet at runtime (and hence require a servlet container to run).

Example of JSP code

```
<html>
  <head></head>
  <body>
    <jsp:include page="header.jsp" >
      <jsp:param name="title"
        value="Example" />
    </jsp:include>
    <p>Counting to three:</p>
    <% for (int i=1; i<4; i++) { %>
      <p>This number is <%= i %>.</p>
    <% } %>
    <p>OK.</p>
  </body>
</html>
```

JSP Tag Extension API

JSP:

```
<%@ taglib uri="/WEB-INF/taglib.tld"
        prefix="mytaglib" %>
<mytaglib:hello name="Bob">
    You're welcome :)
</mytaglib:hello>
```

JSP Tag Extension API

TLD:

```
<tag>
  <name>hello</name>
  <tagclass>HelloTag</tagclass>
  <bodycontent>JSP</bodycontent>
  <attribute>
    <name>name</name>
  </attribute>
</tag>
```

JSP Tag Extension API

JAVA:

```
public class HelloTag extends TagSupport {
    private String name = null;
    public void setName (String string) {
        this.name = string;
    }

    public int doStartTag() throws JspException {
        pageContext.getOut().println(
            "Hello " + this.name + " !");
        return EVAL_BODY_INCLUDE;
    }
}
```

Persistence and Data Storage: Relational Databases

What is persistence?

Most web applications need to store information between sessions. This information should be stored on server side, so that it can be retrieved from different clients.

The database is the common memory of a web application. This is also the main synchronization point!

The most common solution: Relational databases

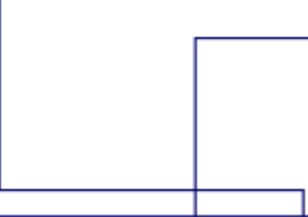
A relational database is a set of tables, consisting of fixed columns (the fields) and an arbitrary number of rows (the entries).

Hypothetical Relational Database Model

PubID	Publisher	PubAddress
03-4472822	Random House	123 4th Street, New York
04-7733903	Wiley and Sons	45 Lincoln Blvd, Chicago
03-4859223	O'Reilly Press	77 Boston Ave, Cambridge
03-3920886	City Lights Books	99 Market, San Francisco

AuthorID	AuthorName	AuthorBDay
345-28-2938	Haile Selassie	14-Aug-92
392-48-9965	Joe Blow	14-Mar-15
454-22-4012	Sally Hemmings	12-Sept-70
663-59-1254	Hannah Arendt	12-Mar-06

ISBN	AuthorID	PubID	Date	Title
1-34532-482-1	345-28-2938	03-4472822	1990	Cold Fusion for Dummies
1-38482-995-1	392-48-9965	04-7733903	1985	Macrame and Straw Tying
2-35921-499-4	454-22-4012	03-4859223	1952	Fluid Dynamics of Aquaducts
1-38278-293-4	663-59-1254	03-3920886	1967	Beads, Baskets & Revolution



Relational database (cont.)

The database typically brings:

- Tables,
- Primary and foreign keys, indexing,
- Query language (SQL),
- Transactions,

Examples of relational database management systems (RDBMS): MySQL, HSQLDB, etc.

When using a relational database in a web application, one has to be careful about:

- The indexing of fields used as keys,
 - The complexity of queries.
-

The problem...



**Relational
database**



**Object
model**

The solution?



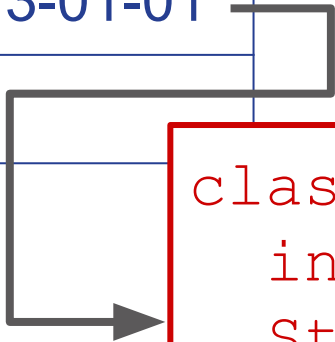
**Object-relational
mapping**

Object-relational mapping

First idea: one row of the table == one object

Table Persons

int id	string name	date birthdate
42	Bob	2013-01-01



```
class Person {  
    int id;  
    String name;  
    Date birthdate;  
}
```

The diagram illustrates the mapping from a database table row to a class object. A grey arrow originates from the row containing '42', 'Bob', and '2013-01-01' in the 'Table Persons' and points to a red-bordered box containing the 'Person' class definition. The class definition includes attributes 'int id', 'String name', and 'Date birthdate', which correspond to the columns in the table row.

Typical issues

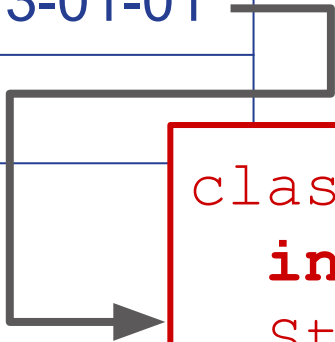
- Type mismatches between programming languages,
 - How to find the row from the object, and vice-versa?
 - How to keep the object and the row in sync?
 - How to represent collections?
 - How to share sub-objects?
 - How to represent inheritance?
-

How to find the row from the object, and vice-versa?

Use primary keys

Table Persons

int id	string name	date birthdate
42	Bob	2013-01-01



```
class Person {  
    int id;  
    String name;  
    Date birthdate;  
}
```

How to keep the object and the row in sync?

Use set/get methods
update method

How to represent collections?

```
class Album {  
    String title;  
    Collection<Track> tracks;  
}
```

```
class Track {  
    String title;  
}
```

Table Album

int id	string title
42	Album 1
43	Album 2

Table Track

int id	string title	int album
101	Track 1	42
102	Track 2	42
103	Track 3	42
104	Track 1	43



How to re

Table Album2Track

int album	int track
42	101
42	102
42	103
43	101

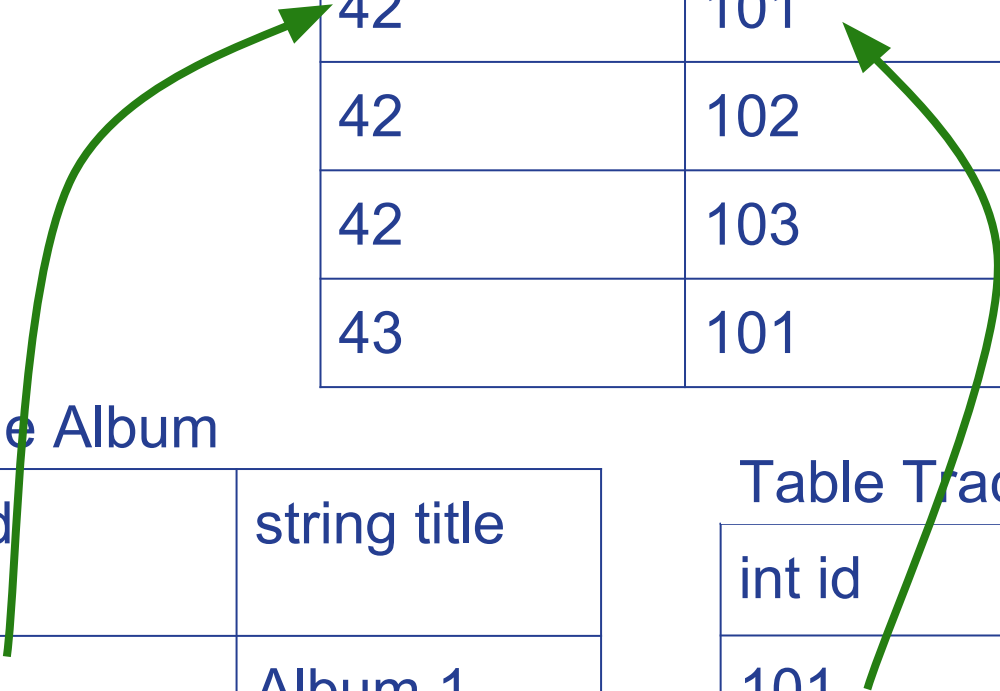
ons? (cont)

Table Album

int id	string title
42	Album 1
43	Album 2

Table Track

int id	string title
101	Track 1
102	Track 2
103	Track 3



How to represent inheritance?

Several techniques:

- Separate tables,
 - Table inheritance,
 - Unique table with discriminant.
-

Relational database: Java

- JDBC (Java DataBase Connectivity):
 - The Servlet code uses the JDBC API to access the contents of the database,
 - The JDBC driver takes care of translating the API calls into SQL requests for the RDBMS.
 - Hibernate:
 - Mapping from Java classes to database tables,
 - Data query and retrieval facilities (HQL).
-

Hibernate: example of Java class

```
public class Person {
    private int id, age;
    private String name;

    public Person() {}
    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }
    public int getId() { return id; }
    public void setId(int id) { this.id = id; }
    public String getName() { return firstName; }
    public void setName(String name) { ... }
    public int getAge() { return age; }
    public int setAge(int age) { this.age = age; }
}
```

Hibernate: create table

```
create table PERSON (  
    id INT NOT NULL auto_increment,  
    name VARCHAR(20) default NULL,  
    age INT default NULL,  
    PRIMARY KEY (id)  
);
```

Hibernate: example of mapping file

```
<?xml version="1.0" encoding="utf-8"?>  
<!DOCTYPE hibernate-mapping PUBLIC ...>
```

```
<hibernate-mapping>
```

```
  <class name="Person" table="PERSON">
```

```
    <meta attribute="class-description">...</meta>
```

```
    <id name="id" type="int" column="id">
```

```
      <generator class="native"/>
```

```
    </id>
```

```
    <property name="name" column="name"  
              type="string"/>
```

```
    <property name="age" column="age" type="int"/>
```

```
  </class>
```

```
</hibernate-mapping>
```

Persistence and Data Storage: NoSQL Databases

NoSQL

A cover name for database technologies which use less constrained consistency models than traditional relational databases.

NoSQL databases are often highly optimized key–value stores.

Example: MongoDB.

MongoDB



Document-oriented database system.

Free and open source software.

First developed in 2007, shift to open source in 2009.

Adopted by a number of major websites including eBay, Foursquare and SourceForge.

MongoDB: Document-Oriented (1/2)

A MongoDB database is a set of **collections** (which stand for tables in relational DB), consisting of **documents** (which stand for records).

The data schema is **flexible**:

- documents in the same collection do not need to have the same set of fields or structure, and
- common fields in a collection's documents may hold different types of data.

Atomicity is guaranteed at the level of the document.

MongoDB: Document-Oriented (2/2)

MongoDB documents are JSON objects, which are represented in a binary form called "BSON".

```
{  
  name: "sue",  
  age: 26,  
  status: "A",  
  groups: [ "news", "sports" ]  
}
```



← field: value
← field: value
← field: value
← field: value

The advantages of using documents are:

- Documents correspond to native data types in many programming language.
 - Embedded documents reduce need for joins.
 - Dynamic schema supports polymorphism.
-

MongoDB: (De-)normalization

Document-oriented database encourage storing information in a de-normalized way, in order to avoid multiple lookups.

De-normalized:

```
{id: 42,  
  title: "album 1",  
  tracks: [  
    {id: 101,  
      title: "track 1"},  
    {id: 102,  
      title: "track 2"},  
    {id: 103,  
      title: "track 3"}  
  ]  
}
```

Normalized:

```
{id: 42,  
  title: "album 1",  
  tracks: [101, 102, 103]}  
-----  
{id: 101,  
  title: "track 1"},  
-----  
{id: 102,  
  title: "track 2"}  
-----  
{id: 103,  
  title: "track 3"}
```

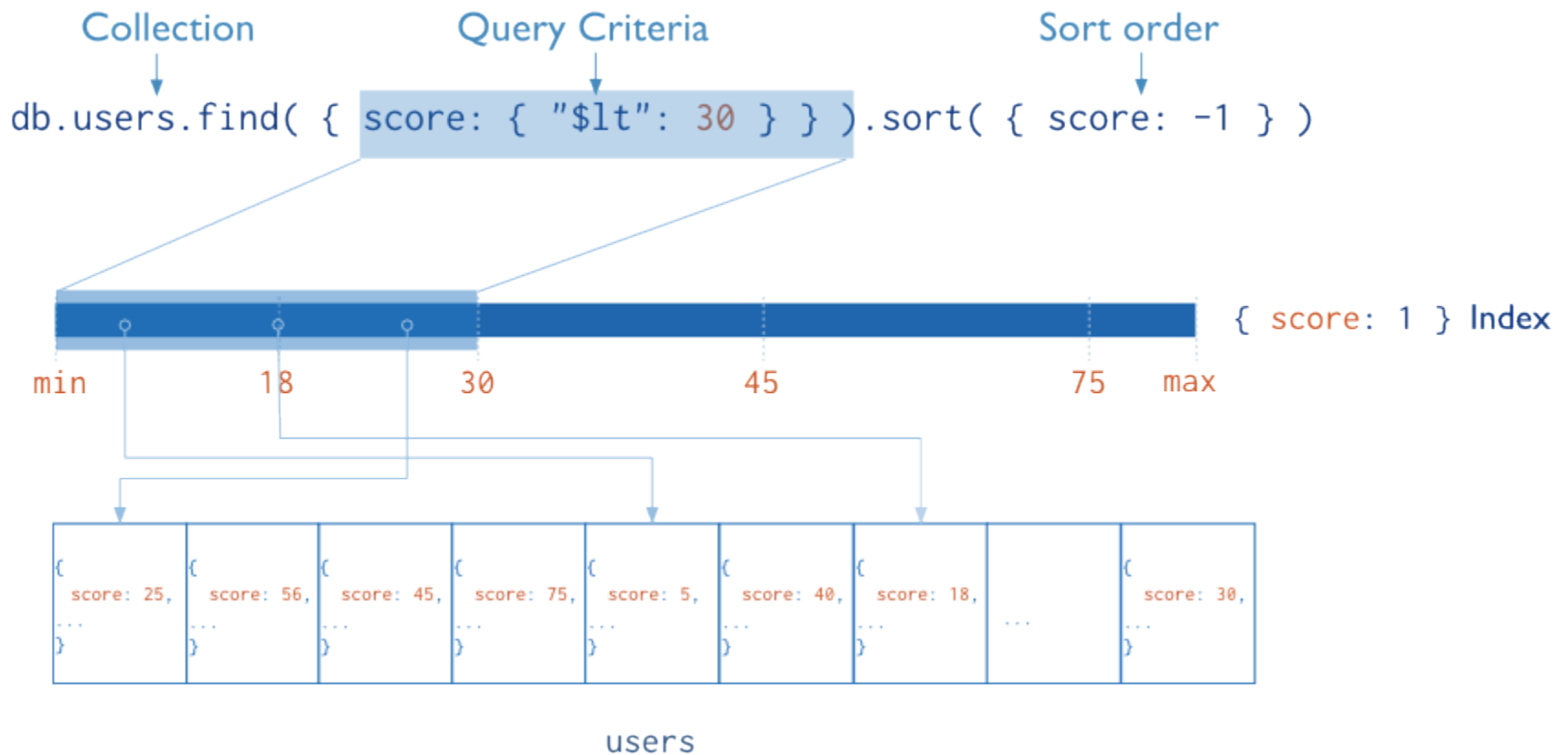
MongoDB: Index (1/2)

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.

All MongoDB collections have an index on the `_id` field that exists by default.

MongoDB: Index (2/2)

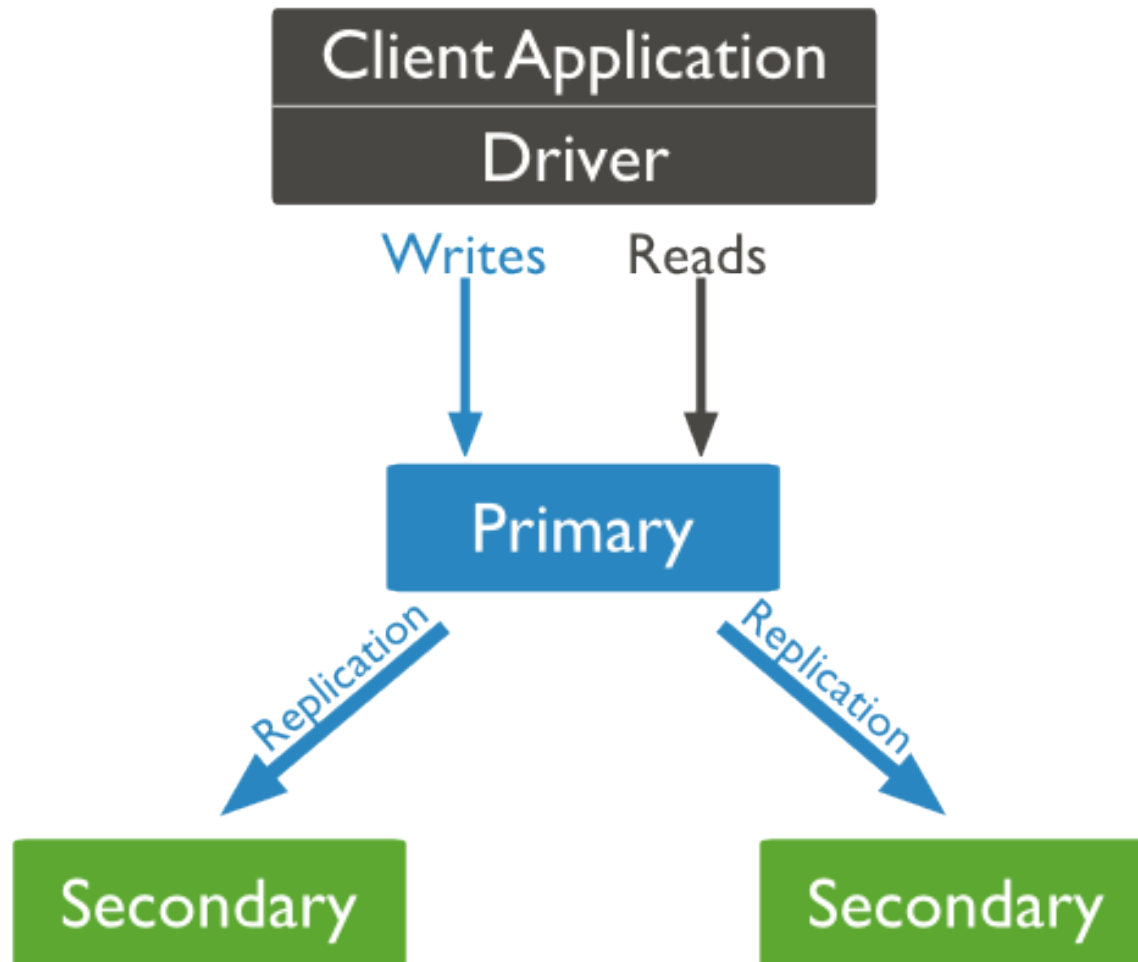


MongoDB: Replication (1/2)

Replication provides redundancy and increases data availability. With multiple copies of data on different database servers, replication protects a database from the loss of a single server.

In some cases, replication can be used to increase read capacity. Clients have the ability to send read and write operations to different servers. Copies can also be maintained at different locations.

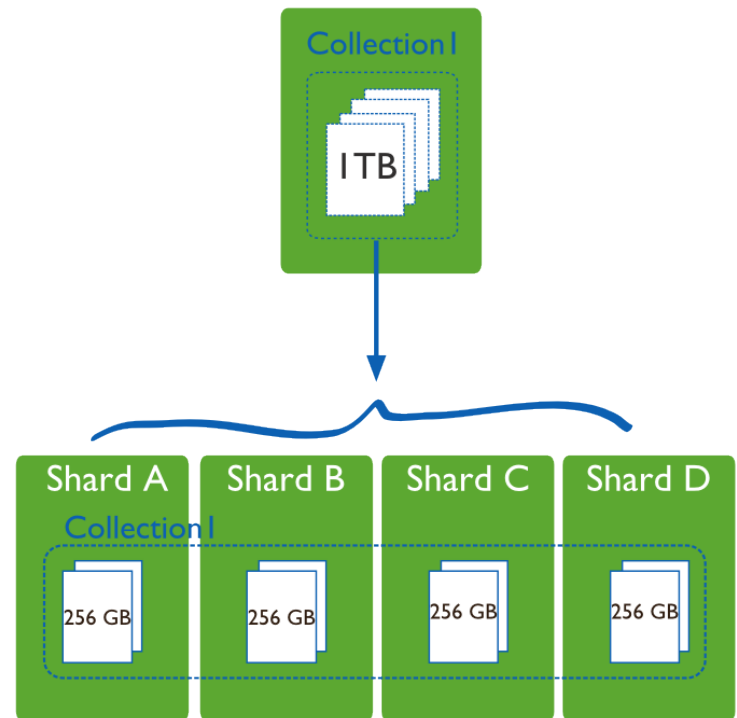
MongoDB: Replication (2/2)



MongoDB: Sharding (1/2)

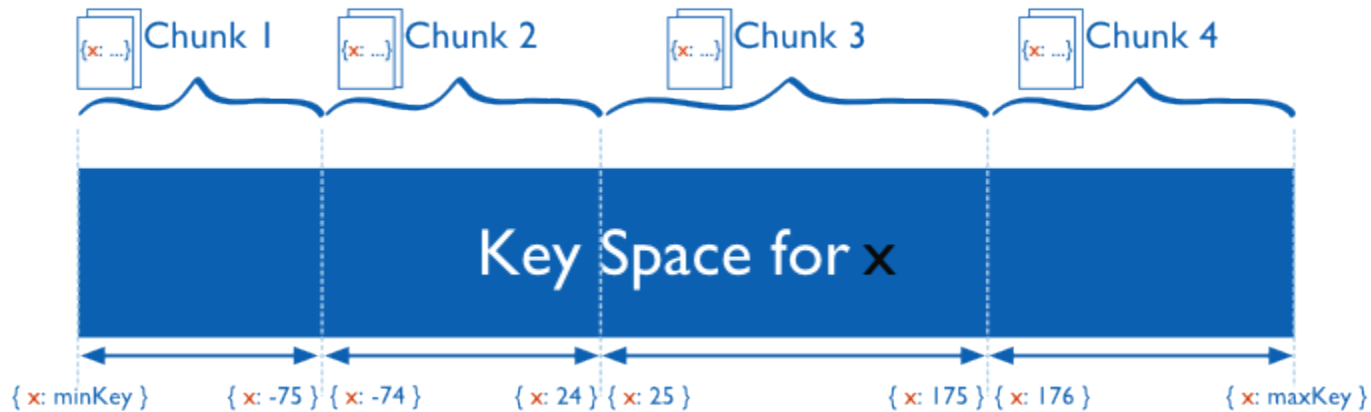
Sharding allows users to partition a collection within a database to distribute the collection's documents across a number of MongoDB instances or shards.

The shard key determines how data is distributed among shards. Selecting the proper shard key has significant implications for performance.

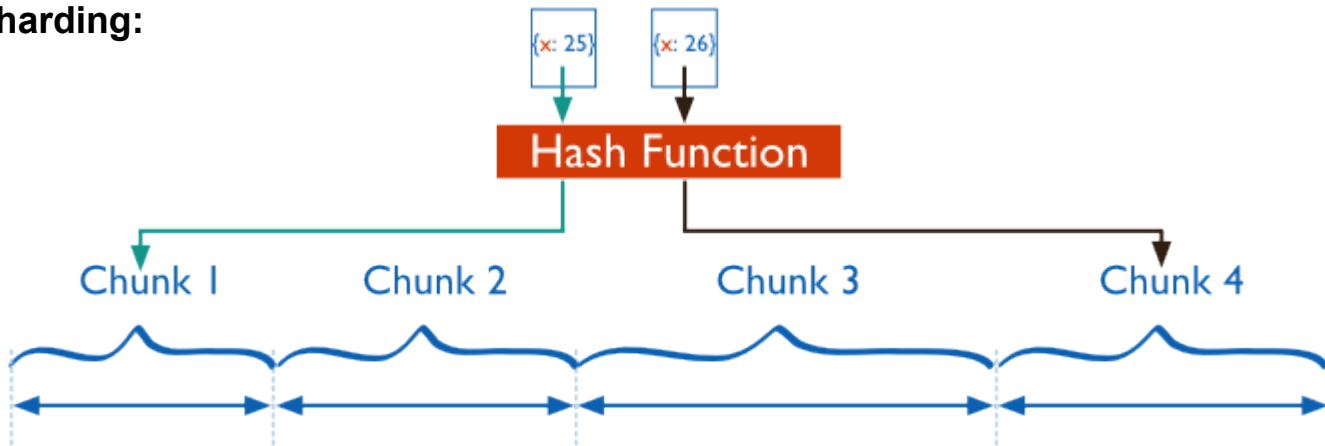


MongoDB: Sharding (2/2)

Range-based sharding:



Hash-based sharding:



MongoDB: Map-Reduce

Collection
↓
db.orders.mapReduce(
 map → function() { emit(this.cust_id, this.amount); },
 reduce → function(key, values) { return Array.sum(values) },
 query → { query: { status: "A" },
 output → out: "order_totals"
 }
)

