2019-
2020

Sistemas Informáticos 1

PREPARACIÓN PARA EL NUEVO Y MEME DE LA CARRERA.

REALIZADO POR NICOLÁS SERRANO SALAS

Contenido

Tema 1: Introducción a los Sistemas Distribuidos

Sistema centralizado vs sistema distribuido

- Sistema centralizado: ordenador central y red de terminales sin capacidad de proceso
- Sistema distribuido:
    - Conjunto de elementos de proceso computacional autónomos.
    - No necesariamente homogéneos.
    - Interconectados por una red de comunicaciones.
    - Cooperan mediante envío de mensajes para realizar tareas que tienen asignadas.

Motivación y desventajas de los sistemas distribuidos

Motivación:

- Distribución inherente de algunas aplicaciones
- Compartición de recursos
- Acceso a recursos remotos
- Economía
- Aumento de potencia computacional y velocidad de cálculo (paralelización y escalabilidad)
- Flexibilidad y modularidad

Desventajas:

- Complejidad
- Problemas y errores de comunicación. (Perdida mensajes, saturación y latencia)
- Seguridad
- Confidencialidad

Tipos de sistemas distribuidos

Atendiendo a su grado de acoplamiento (HW)

- Fuertemente acoplados: Procesadores que comparten memoria o buses. Multiprocesador.
- Débilmente acoplados: Procesadores autónomos interconectados.

Atendiendo a su arquitectura software:

- P2P:
    - Sistema simétrico.
    - Todos los procesos desempeñan tareas semejantes
    - Interactúan para realizar una actividad distribuida
- Cliente-Servidor:
    - Sistema asimétrico
    - Cliente:
        - Activo
        - Envía peticiones
        - Espera hasta que llega la respuesta
    - Servidor:
        - Pasivo
        - Espera peticiones
        - Cuando recibe una petición, la procesa y envía respuesta
        - Puede conservar o no el estado de la comunicación.

Cliente pesado – Servidor pesado

2-Tier: Cliente = Lógica de aplicación + lógica presentación. Servidor = acceso a datos

3-Tier: Lógica de la aplicación separada de la de presentación

N-Tier Múltiples niveles de clientes y servidores. (Un servidor puede ser cliente de otro)

Componentes sw de un sistema cliente-servidor.

- Cliente: todo proceso que reclama servicios a otro
- Middleware: Enlace entre cliente y servidor o entre servidores. Se ejecuta tanto en el servidor y en el cliente.
- Servidor: todo proceso que proporciona un servicio a otros.

Computación en la nube

Implementación de un sistema distribuido con almacenamiento y acceso a través de intenet en los que todos los recursos HW y SW son ofrecido como un servicio.

Ventajas:

- Disminución de costes (pay-as-per-use)
- Permite desarrollos complejos sin necesidad de un conocimiento avanzado
- Administración

Desventajas:

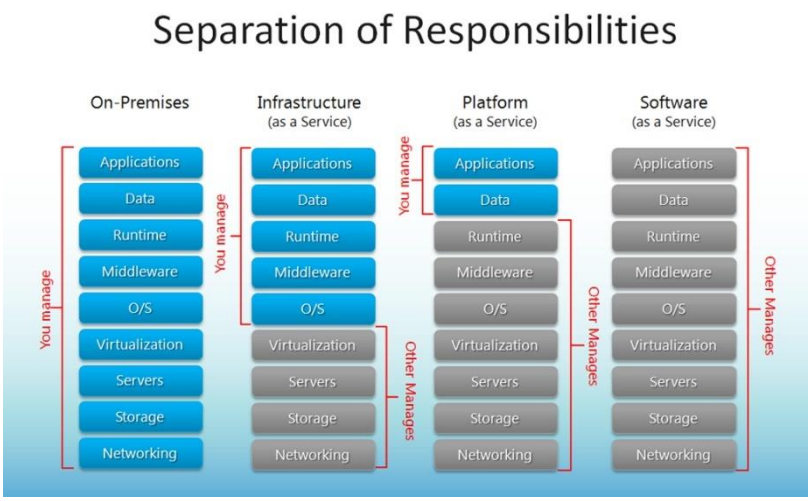
- Privacidad
- Falta de control y dependencia del proveedor

Características:

1. Servicios bajo demanda
2. Amplio acceso de red
3. Pooling de recursos
4. rápida elasticidad
5. Medición de servicios

Modelos de servicio

1. IaaS (Infrastructure as a Service)
2. Paas (Platform as a Service)
3. Saas (Service as a Service)



Tema 2: SSDD basados en la WWW

Introducción

Internet, Intranet, extranet:

- Internet: Red pública, externa, comunicación con el mercado. Business to customer (b2c)
- Intranet: Red privada., comunicaciones de un grupo de usuarios. Business to employee (b2e)
- Extranet: Acceso a una Intranet por usuarios externos: Business to business (b2b)

URI

- URI: (Uniform Resource Identifier) Extensión del nombre de un fichero que permite identificar recursos en Internet.
- URN: (Uniform Resource Name): Identifica un recurso.
- URL: (Uniform Resource Locator): Donde / Como encontrar el recurso

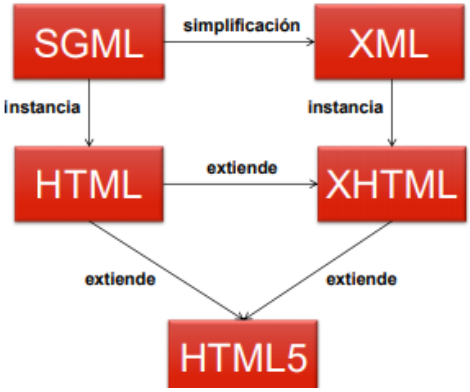
Web hipertexto

Funcionamiento

1. El usuario solicita un recurso mediante su URL en un navegador (cliente)
2. El navegador genera una petición HTTP y la envía al servidor Web
3. El servidor Web recibe la petición y envía el recurso solicitado codificado en HTML
4. El cliente interpreta y muestra el documento recibido

Familia de lenguajes de la WWW

Lenguajes de etiquetas y atributos.



CSS

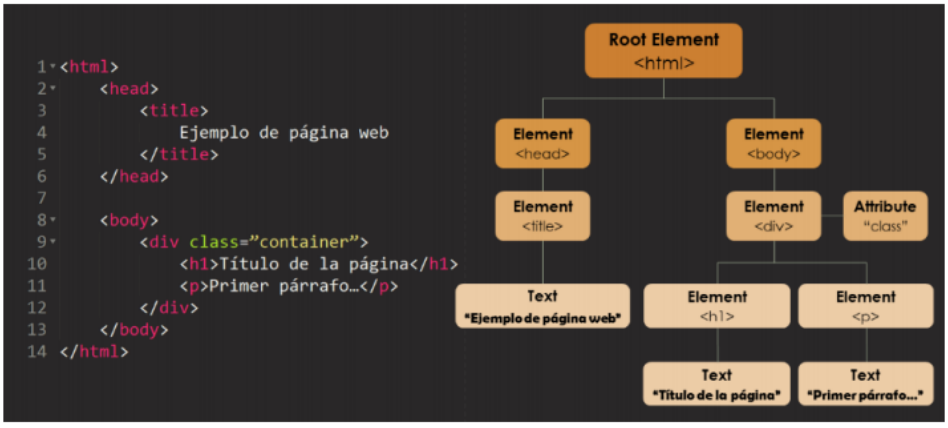
Usado para describir la semántica de presentación de un documento escrito con un lenguaje de marcado.

Tipos de selectores:

- Selector de elemento. Elemento { }
- Selector id. #id { }
- Selector class .class { }
- Combionación de selectores:
  - Descendientes #id elemento { }
  - Hijos elemento > elemento { }
  - Hermanos .class ~ elemento { }
  - Hermanos adyacentes #id + .class {}

Precedencia:

1. Inline
2. Id
3. Class
4. Elemento



Ejercicio CSS

XML

Validación con DTD

Parseo con documentos

- SAX: Simple API for XML

- DOM: Document Object Model (Árbol DOM)
    - API DOM
    - XPATH

XSL

Documento XML que funciona como CSS de otro documento xml

JSON

Sustituto actual de XML, Javascript Object Notation

Web interactiva

Sesión

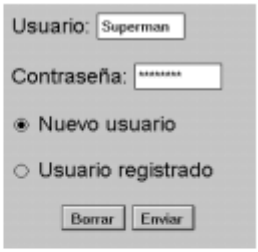
Contexto persistente en el que almacenar/recuperar datos mientras la relación entre cliente y servidor se mantenga activa

Cookies

- Pequeño fragmento de información que el servidor (con el permiso del navegador) almacena en el cliente ya que HTTP es un protocolo "sin memoria"
- Cada vez que el navegador solicite una nueva página al servidor envía también la cookie

Formularios HTML

```
<FORM ACTION="http://host/cgi-bin/logon" METHOD="GET">
   <p>
      Usuario:
      <INPUT NAME="usuario" TYPE="text" SIZE="10" MAXLENGTH="8">
   </p>
   <p>
      Contrase&ntilde;a:
      <INPUT NAME="clave" TYPE="password" SIZE="10" MAXLENGTH="8">
   </p>
   <p>
      <INPUT NAME="nuevo" TYPE="radio" VALUE="si" CHECKED>
      Nuevo usuario
   </p>
   <p>
      <INPUT NAME="nuevo" TYPE="radio" VALUE="no">
      Usuario registrado
   </p>
   <center><p>
      <INPUT TYPE="RESET" VALUE="Borrar">
      <INPUT TYPE="SUBMIT" VALUE="Enviar">
   </p></center>
</FORM>
```



Diferencia entre GET Y POST

- Get los atributos se ven en la url. Ex: /logon?usuario=Superman&clave=loislane&nuevo=si
  Primero url?primer_input=valor_primer_input&segundo_input=valor_segundo_input&…
- POST los atributos no se ven en la url.

CGI Commont Gateway Interface

Método "estándar" para que un servidor WWW pueda ejecutar programas externos y recoger información de ellos. Extensión del protocolo HTTP

Ventajas

- Sencillez de programación
- Uso de cualquier lenguaje de programación
- No afecta al funcionamiento del servidor. Se ejecuta como proceso independiente
- Estándar. Portable entre distintos servidores

Desventajas

- Lentitud, crea un proceso por cada petición

- Sin estado de la comunicación entre peticiones. (Gestión de sesiones por los programas mediante campos de formulario ocultos)

Web API

Surgen para tratar de evitar los problemas de bajo rendimiento de la interfaz CGI.

- Los nuevos programas se enlazan junto con el servidor en una librería dinámica
- El servidor llama a las funciones de librería como tareas dentro del propio proceso servidor
- El proceso servidor no finaliza: Se mantienen ficheros abiertos, conexiones a bases de datos, etc. entre llamadas a funciones
- Se proporciona una API de acceso a los datos y estado del servidor

Inconvenientes

- Un fallo en una rutina puede hacer caer el servidor completo
- Lenguajes de programación normalmente limitados.
- Difícil de programar.

Interfaces hibridas

- Intentan conseguir las ventajas de CGIs y Web APIs evitando sus inconvenientes
- Los programas se desarrollan de modo independiente al servidor Web, y en cualquier lenguaje
- El servidor Web, durante su inicialización, puede arrancar los programas en procesos independientes
- Los programas arrancados, tras inicializarse, quedan a la espera de recibir peticiones
- Mecanismo de comunicación entre procesos más rápido
- Tras atender una petición, el programa no finaliza: vuelve a esperar la siguiente petición

Páginas dinámicas

- Client side scripting
    - Inclusión de código en el documento que el cliente interpretará para variar dinámicamente la presentación de la página
- Server side scripting
    - Inclusión de código funcional en el fichero HTML que contiene la descripción de la página
    - El servidor lo interpretará para generar dinámicamente la página antes de su envío al cliente

Modelos de implementación en el servidor

NodeJS

PHP

Flask

Mapeo/Enrutamiento de URLS

```
@app.route('/', methods=['GET', 'POST'])
@app.route('/index', methods=['GET', 'POST'])
def index():
    catalogue_data = open(os.path.join(app.root_path,'catalogue/catalogue.json'), encoding="utf-8").read()
    catalogue = json.loads(catalogue_data)

    categories = []

    for pelicula in catalogue['peliculas']:
        for category in pelicula['genre']:
            if(category not in categories):
                categories.append(category)

    searchResult = []
    if request.method == "POST":
        search = request.form['search']
        genre = request.form['genre']
        for pelicula in catalogue['peliculas']:
            if pelicula['title'].lower().__contains__(search.lower()):
                if genre == "0":
                    searchResult.append(pelicula)
                else:
                    for movieGenre in pelicula['genre']:
                        if genre == movieGenre:
                            searchResult.append(pelicula)
    else:
        searchResult = catalogue['peliculas']
    return render_template('index.html', searchResult = searchResult, categories = categories)
```

Cookies

```
from flask import make_response

@app.route("/setcookie/<user>")
def setcookie(user):
    msg = "user cookie set to: " + user
    response = make_response(render_template('mensaje.html', mensaje=msg))
    response.set_cookie('helloflask_user',user)
    return response
```

```
@app.route("/getcookie")
def getcookie():
    user_id = request.cookies.get('helloflask_user')
    if user_id:
        msg = "user is: " + user_id
    else:
        msg = "no user cookie"
    return render_template('mensaje.html', mensaje=msg)
```

Sesiones

```
from flask import session

@app.route("/setsession/<data>")
def setsession(data):
    msg = "session data set to: " + data
    session['data'] = data
    session.modified = True
    return render_template('mensaje.html', mensaje=msg)

@app.route("/getsession")
def getsession():
    if 'data' in session:
        msg = "session data: " + session['data']
    else:
        msg = "no session data"
    return render_template('mensaje.html', mensaje=msg)
```

Django

Ejecución de código en el cliente

Objetivos

- Aumentar la capacidad de interacción del usuario con la aplicación
- Disminuir el trasiego de datos entre el cliente y el servidor

Tecnologías asociadas al uso de Javascript en el cliente:

- DHTML = HTML + JavaScript + CSS
- AJAX = Asynchronous JavaScript and XML

Javascript

- Inline: : <script type = "text/javascript"> function nombre() { … }</script>
- Documento aparte: <script type = "text/javascript" src=""></script>

Obtener elementos del árbol DOM

- document.getElementById("id")
- document.getElementsByName("tag")
- document.getElementsByClassName("clase")

Métodos para crear, eliminar, sustituir, concatener… nodos:

- createElement, createTextNode, createAttribute
- insertBefore
- removeChild, remove Attribute
- appendChild

Recorrer el árbol

- childNodes, parentNode, nextSibling, previousSibling

Atributos para acceder/modificar el contenido HTML/Texto del nodo:

- innetHTML, outerHTML
- innerText, outerText

Atributos style

- .style.color, .style.backgroundColor

Localstorage y sessionstorage

Ejercicio validación de formulario

AJAX

Conjunto de técnicas para intercambiar información con el servidor en background y actualizar partes de un documento HTML sin recargar la página completa.

Se usa para realizar peticiones HTTP al servidor

Si tenemos que mandar parámetros:

- GET: concatenarlos en la url de la dirección

- POST: Usar función *setRequestHeader(header, value)*

Jquery

Trabaja con estructuras similares a un array que permiten aplicar acciones sobre varios elementos en una misma sentencia

Sintaxis abreviada:
$(selector).accion1().accion2()…

Iteración sobre conjuntos de elementos:
Ejemplo *$('form :input').each(function(index, element) {})*

- Selector de elemento:
```
$('td');
```
- Selector *id*:
```
$('#nombre');
```
- Selector *class*:
```
$('.nombre-clase');
```
- Combinaciones de selectores:
```
$('div ul#nombre');
```
- Selector atributo:
```
$('[atributoN]');
$('tipo[nombre-atributo]');
$('tipo[nombre-atributo=valor]');
$('tipo[nombre-atributo!=valor]');
```
- Pseudo-selectores:
```
$('tbody tr:even');
$('section:first');
$('tr:gt(0)');
```

Manipulación dinámica del documento HTML

- Texto contenido de un elemento:
```
$('#contactDetails h2').text('CONTACT DETAILS');
```
- HTML de un elemento:
```
$('#contactDetails h2').html('<span>Contact Details</span>');
```
- Valor de un campo de formulario:
```
$('[name="contactName"]').val('testing 123');
```
- Valor de un atributo:
```
$('textarea').attr('maxlength', 200);
```
- Quitar valor de un atributo:
```
$('textarea').removeAttr('maxlength');
```
- Ocultar un elemento:
```
$('#loading').hide();
```

```
// todos los tr dentro de contactScreen
$('#contactScreen').find('tr');

// padres de los párrafos
$('p').parent();

// form antecesor de todos los input
$(':input').parents('form');

// labels hermanas de inputs con atributo required
$(':input[required]').siblings('label');

// todos los inputs + todos los labels del documento
$(':input').add('label');
```

AJAX con jquery (visto en PSI)

```
$.ajax({
        url:,
        type:,
       data: {
       },
       success: function(data) {
        }
});
```

Frameworks

- Boostrap
- ZURB Foundation
- HTML5 Boilerplate
- AngularJS
- ReactJS
- Meteor

HTML5

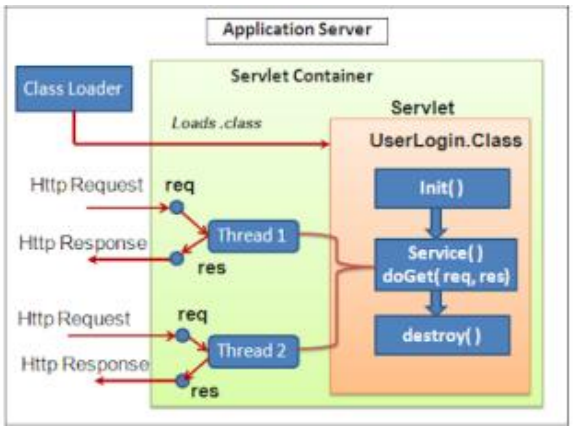
- Nuevos elementos específicos sobre estructura del documento: header, nav, article, section, aside, footer.
- Nuevos controles para los formularios date, time, email, url, search
- Tag para video y audio. <video></video>
- Canvas. <canvas></canvas>
- SVG <svg> </svg>
- Nuevos campos de formulario. datalist, keygen, output

Alternativas para el backend

Java Enterprise Edition

Extensión de la Java SE con especificaciones y APIs para:

- Gestión de transacciones
- Acceso a base de datos
- Desarrollo de aplicaciones web



Servlets

Aplicación Java que se ejecuta en el servidor gestionando y procesando peticiones HTTS

Ventajas:

- Portabilidad y flexibilidad
- Seguridad
- Rendimiento

JSP. Javaserver pages

Tecnología Java de server side scripting para la generación dinámica de contenidos

.NET

Arquitectura de Microsoft para aplicaciones Web. Consta de tres componentes principales:

- Un entorno de aplicaciones independiente del lenguaje y optimizado para el entorno distribuido: .NET Framework
- Un entorno de desarrollo para la programación de aplicaciones: Visual Studio .NET
- Un sistema operativo que soporta entornos distribuidos y el framework .NET: Windows Server

Modelo-Vista-Controlador MVC

- Modelo. Permite tener la funcionalidad de la aplicación.
- Vista. Realiza la presentación de los datos del modelo
- Controlador. Recibe las interacciones del usuario y las traslada en acciones sobre el modelo

Tema 3: Bases de Datos Distribuidas

Introducción

Gestor de base de datos (SGBD)

Sistema que se encarga de la organización, almacenamiento, gestión y recuperación (eficiente) de la información

Incluye:

- Un lenguaje para modelar la información de acuerdo con un determinado modelo (DDL, Data Definition Language)
- Estructuras de almacenamiento de la información optimizadas para trabajar con un gran volumen de datos
- Un lenguaje para recuperar/manipular la información almacenada mediante búsquedas dirigidas (DML, Data Manipulation Language)
- Los mecanismos adecuados que le permitan integrarse en un sistema de acceso con control transaccional

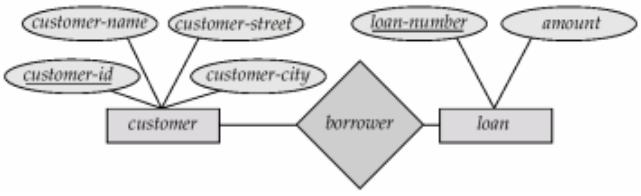
El modelo más habitual de gestores de bases de datos es el que sigue el modelo relacional cuyo l lenguaje más utilizado actualmente para modelar y recuperar la información es SQL (Structured Query Language)

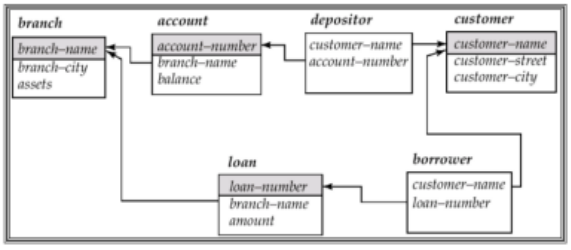
Acceso a los datos

Originalmente las aplicaciones relacionadas con el manejo de datos se construían sobre un conjunto de ficheros, pero esto originaba varios problemas:

- Volumetría: Muchos datos = tamaño del fichero inmanejable
- Redundancia e Inconsistencia en los datos
- Acceso a los datos ineficiente: información duplicada
- Integridad debida restricciones
- Atomicidad de las modificaciones difícil de asegurar
- Simultaneidad de acceso por varios usuarios
- Seguridad. Restringiendo el acceso parcial a los datos
- Abstracción de los datos

Modelo Entidad-Relación





Modelo Relacional

Informalmente, una relación puede considerarse una tabla (conceptualmente)

- Una base de datos que se ajusta al modelo relacional puede representarse como un conjunto de tablas
- Convertir un diagrama E-R a tablas es el primer paso para obtener una base de datos relacional
- Normalmente cada entidad y cada relación muchos a muchos da lugar a una tabla

SQL

Lenguajes de SQL

- DDL: Esquemas de relación, índices, vistas, restricciones, derechos de acceso, tablas
- DML: Lenguaje de consulta basado en álgebra relacional, CRUD (insert, delete, update, select)

Tipos de datos

- Varchar
- Integer
- Date

Se pueden extraer valores independientes de date/time/timestamp. Ej: *extract (year from r.comienzo)*

Creación, alteración y destrucción de tablas

Restricciones

CHECK

NOT NULL

UNIQUE

PRIMARY KEY

REFERENCES

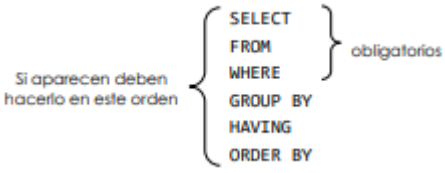
Agregaciones y agrupaciones

Comparaciones de caracteres (LIKE 'Exam%')

Cruce de tablas (Join)

Combinación de relaciones (UNION, INTERSECT, EXCEPT)

Subconsultas (S IN R / EXIST R)



SQL en las aplicaciones

Las aplicaciones no acceden a datos almacenados en bases de datos relacionales a través de SQL interactivo, sino a través de un middleware, el driver de la base de datos (ODBC, JDBC)

- SQL embebido
  - Ej: PHP (PDO), Java (java.sql)
- Sentencias preparadas
  - Sentencia SQL precompilada que acepta parámetros
  - Mejora el tiempo de respuesta y/o la seguridad
  - Útil cuando una misma sentencia se utiliza muchas veces
- DataSources lógicos
  - Como patrón de diseño mejora la reusabilidad/mantenibilidad
  - Como herramienta de acceso a datos mejora de los tiempos de acceso
- Separación de responsabilidades
  - Mejora de la legibilidad y mantenibilidad del código
  - Frameworks y bibliotecas que permiten que las sentencias SQL no aparezcan inmersas en el código funcional
  - Ej: MyBatis (elimina casi todo el código JDBC)
- ORM
  - Abstracción del acceso a datos
  - Frameworks de persistencia que almacenan y recuperan objetos de una base de datos relacional
  - Ej: Hibernate, django

Procedimientos almacenados y Triggers

Funcionalidad de usuario dentro de una base de datos SQL

Funciones y procedimientos almacenados

Se ejecutan a petición del usuario.

- Mejoran de rendimiento frente al SQL interactivo
- Acceso está controlado por mecanismos de seguridad
- Acepta parámetros de entrada
- Los procedimientos se invocan, las funciones se incluyen dentro de una sentencia SQL
- La sintaxis se valida en tiempo de ejecución, no durante la creación

```
CREATE FUNCTION nombre_funcion (tipos-argumentos)
   RETURNS integer AS $$

DECLARE
    -- declaraciones

BEGIN
    -- cuerpo
END;
$$ LANGUAGE lenguaje;
```

```
CREATE OR REPLACE FUNCTION trae_pelicula (integer) RETURNS text AS $$
DECLARE
  pelicula_id ALIAS FOR $1;
  encontrada_pelicula pelicula%ROWTYPE;
BEGIN
  SELECT INTO encontrada_pelicula * FROM pelicula WHERE id = pelicula_id;
  RETURN encontrada_pelicula.titulo || "  (" || encontrada_pelicula.agno || ")";
END;
$$ LANGUAGE plpgsql;
```

Retonrno de tuplas:

```
CREATE OR REPLACE FUNCTION get_film (p_pattern VARCHAR)
 RETURNS TABLE (
 film_title VARCHAR,
 film_release_year INT
)
AS $$
BEGIN
 RETURN QUERY SELECT
 title,
 cast( release_year as integer)
 FROM
 film
 WHERE
 title LIKE p_pattern ;
END; $$

LANGUAGE 'plpgsql';
```

Triggers

Se ejecutan cuando ocurre un evento asociado a una tabla. Se pueden considerar un tipo especial de procedimiento almacenado.

CREATE TRIGGER name { BEFORE | AFTER } { INSERT | UPDATE | DELETE }

ON table

FOR EACH ROW

EXECUTE PROCEDURE function_name (arguments)

```
CREATE OR REPLACE FUNCTION tr_function()
RETURNS TRIGGER
AS $$
BEGIN
  NEW.c3 = NEW.c1 + NEW.c2
  RETURN NEW
END;
$$
LANGUAGE 'plpgsql';

CREATE TRIGGER tr BEFORE INSERT ON tableName
FOR EACH ROW EXECUTE
PROCEDURE tr_function();
```

Si queremos realizar distinas operaciones en el mismo trigger con distintos varios eventos(insert, update, delete) usar:

IF (TG_OP = 'EVENT') THEN __ ELSE __ END IF;

Optimización de consultas

Explain plan

Forma en que el SGBD busca o inserta los datos

- Formas de cruzar tablas
- Uso de índices
- Orden de ejecución de subconsultas

Estrategias de acceso

- Directo/Constante (CONST). Tablas con un solo registro o por valor en índice
- Cruce por clave única (EQ_REF)
- Clave no única (REF)
- Merge de índices (INDEX_MERGE)
- Clave única en subconsulta (UNIQUE_SUBQUERY)
- Clave no única en subconsulta (INDEX_SUBQUERY)
- Rango en índice (RANGE). =, <>, <, >, <=, >=, IS NULL, BETWEEN, LIKE, IN
- Full index scan (INDEX)
- Full table scan, secuencial (ALL)

Big Data y NoSQL

Big Data

"Big Data" es similar a "Small Data", solo que con mayores volúmenes de datos. La diferencia de tamaño requiere de soluciones diferentes: Técnicas, Herramientas y Arquitecturas

Las 3 V

Volumen, Velocidad y Variedad

Las 5 V

Volumen, Velocidad, Variedad, Veracidad, Valor

Escalabilidad

Vertical

Incrementar la potencia de la máquina en la que se ejecuta el software

Horizontal

Distribuir la carga de trabajo entre varios ordenadores conectados entre sí

Teorema de CAP

No se pueden cumplir las 3 a la vez.

- Sistemas CA: SGBDR
- Sistemas CP: Mayoría NoSQL (MongoDB)
- Sistemas AP: Apache Cassandra

Consistency

Todos los nodos contengan valores consistentes entre sí en todo momento

Availability

Garantía de que cada petición a un nodo reciba una confirmación de si ha sido o no resuelta satisfactoriamente

Partition Tolerance

Pueda fallar un nodo o conexión y el sistema siga funcionando

Nueva problemática

- Problema de caída de servidores cuando tengo tantos
- Las BBDD se pueden distribuir, pero a costa de sus propiedades ACID (atomicidad, consistencia, aislamiento (isolation) y durabilidad)
- El procesamiento distribuido en un contexto big data requiere el uso de modelos computacionales no estándar:

Apache Hadoop

Las tareas se ejecutan en una red (cluster hadoop) de ordenadores conectados entre sí (nodos) que se reparten la tarea

HDFS

Permite aprovechar y trabajar con la capacidad total de almacenamiento de todos los ordenadores a la vez, mostrándonosla como si fuera uno solo.

Es un sistema de almacenamiento tolerante a fallos (mediante replicación).

MapReduce

Un problema objetivo debe ser paralelizable.

- Etapa Map: Se divide el problema en problemas menores que son resueltos paralelamente
- Etapa Reduce: el conjunto de soluciones a los problemas menores es sintetizado en una solución al problema original

NoSQL

- Bases de datos sin esquemas (no relacionales)
- Mayormente utilizan interfaces distintas al SQL
- Soportan almacenamiento de grandes cantidades de datos mediante escalabilidad horizontal
- Operan sobre infraestructuras distribuidas, como Hadoop
- Flexibles

Tipos:

- Basados en pares clave-valor. Ej: Redis
- Basados en grafos. Ej: Neo4J
- Basados en columnas. Ej: Apache Cassandra
- Basados en documentos. Ej: MongoDB

Bases de datos documentales

- Se basan en el modelo clave-valor, pero permitiendo el uso de meta-datos para aportar mayor expresividad.
- La unidad organizativa de la información es el documento, que goza de alta flexibilidad. Cada uno consta de un ID único.
- Los datos se agrupan en colecciones y documentos, que serían como las tablas y las filas, respectivamente, en las BBDD Relacionales.
- Suelen basarse en el formato JSON preferentemente, si bien pueden usar también XML.
- La principal ventaja es la flexibilidad, ya que no tienen estructuras predefinidas. Podemos tener documentos diferentes entre sí
- La utilización de esquemas flexibles también las hace propensas a errores de introducción de datos, por lo que es necesario implementar métodos de saneado y limpieza de datos

MongoDB

- Documento: unidad básica de almacenamiento. La información se guarda en formato BSON (Binary JSON). Se permiten documentos embebidos en otros.
- Colección: grupos de documentos.
- Base de Datos: contenedores físicos para almacenar colecciones
- Cluster: almacena varias bases de datos.

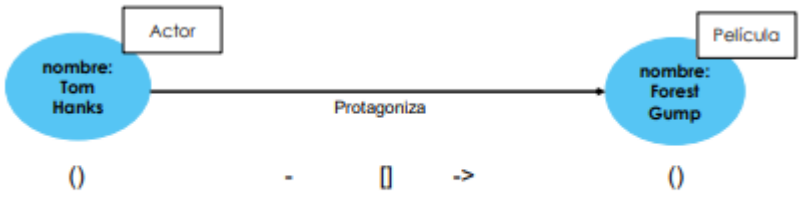
Comandos CRUD

- Mostrar bases de datos:
  - `show dbs`
- Creación y utilización de bases de datos:
  - `use nombre_base_de_datos`
- Borrado de bases de datos en uso:
  - `db.dropdatabase`
- Mostrar colecciones:
  - `show collections`
- Crear colección en una base de datos en uso:
  - `db.createCollection(nombre, opciones)`
    - Se crea automáticamente cuando se inserta un documento.
- Insertar un documento en una colección:
  - `db.nombre_coleccion.insert(nombre_documento)`
    - `db.nombre_coleccion.insertOne(documento)`
    - `db.nombre_coleccion.insertMany(documentos)`
- Eliminar colección:
  - `db.nombre_coleccion.drop()`
- Búsqueda:
  - `db.nombre_coleccion.find()`
- Actualizar datos de documentos:
  - `db.nombre_coleccion.update(criterio_seleccion, datos)`
- Actualizar documento completo:
  - `db.nombre_coleccion.save({_id:ObjectId(), datos})`
- Eliminar documentos de una colección:
  - `db.nombre_coleccion.remove(criterio_seleccion)`
- Ordenar documentos en una colección:
  - `db.nombre_coleccion.find().sort({KEY:i})`
    - Donde i puede ser 1 (ascendente) o -1 (descendente)
    - Se pueden indicar multiples campos a indexar
- Creación de índices:
  - `db.nombre_coleccion.ensureIndex({KEY:i})`
    - Donde i puede ser 1 (ascendente) o -1 (descendente)
- Agregación en la búsqueda:
  - `db.nombre_colección.aggregate(operacion_de_agregacion)`

Ejercicio MongoDB

Bases de datos basadas en grafos (Neo4J)

- Neo4J se centra más en las relaciones entre los datos que en los aspectos comunes entre conjuntos de datos (tales como tablas de filas o colecciones de documentos).
- Define un lenguaje propio (Cypher) para manipulación de los datos, pero existen varios lenguajes capaces de interactuar con Neo4J



(:Actor {nombre: 'Tom Hanks'}) - [:Protagoniza] -> (:Pelicula {nombre: 'Forest Gump'})

Tema 4: Servicios de Back-End

Introducción al proceso de transacciones

- Concepto orientado a proveer tolerancia a fallos y permitir la concurrencia en sistemas distribuidos
- Una transacción es una colección de operaciones de lectura y escritura de datos que están relacionadas a nivel lógico que:
    - Deben ocurrir en su totalidad o no ocurrir en absoluto (atomicidad)
    - Si la transacción se ejecuta, los efectos de las operaciones de escritura deben persistir; y si no se completa, la transacción no debe producir ningún efecto
    - Debe implementarse de forma que estos efectos se garanticen incluso si se produce un fallo del sistema

Procesos transaccionales

- El cliente solicita la ejecución de un procedimiento remoto especial en el servidor: una transacción
- El servidor garantiza la atomicidad de la transacción es decir, el sistema nunca queda en un estado inconsistente aunque haya un fallo
- El servidor garantiza la correcta ejecución concurrente de las transacciones.

Estados consistentes

El sistema se encuentra en un estado consistente si satisface todas sus invariantes.

Las transacciones pasan al sistema de un estado consistente a otro estado consistente pudiendo pasar por un estado inconsistente.

Si la transacción falla, el sistema debe quedar en un estado consistente.

Propiedades de las transacciones

Cumplen las propiedades ACID:

- Atomicidad:
    - La transacción es una unidad indivisible de trabajo (Se ejecuta entera o no se ejecuta)
- Consistencia:
    - Al finalizar su ejecución, el sistema debe quedar en estado estable consistente
    - Si falla la ejecución se debe devolver el sistema a su estado inicial (rollback)
- Aislamiento:
    - Una transacción no se ve afectada por otras que se ejecuten concurrentemente, aunque utilicen los mismos recursos
    - • Necesario que la transacción “bloquee” los recursos que tiene que actualizar
- Durabilidad:
    - Sus efectos son permanentes una vez que ha finalizado correctamente (commit)

Modelos de transacciones

- Transacciones Planas (no hay llamadas a otras transacciones)
- Transacciones Planas Distribuidas.
    - Trabajan con recursos ubicados en diferentes sistemas
    - Two-Phase Commit
- Transacciones anidadas. Llamada a una transacción desde otra (modelo subrutinas)
    - Un commit de una transacción hace sus cambios visibles a todas las transacciones precedentes en la jerarquía de llamadas

- Un rollback de una transacción realiza un rollback de todas las subtransacciones que haya ejecutado, aunque estas ya hayan realizado un commit
- Savepoints para evitar el efecto de "todo o nada" en restados consistentes dentro de la transacción

Aislamiento

Tipos de violación de aislamiento

Actualización perdida

La escritura de una transacción es ignorada por otra, que realiza una nueva escritura basada en la versión previa del objeto. (No se tiene en cuenta la información que se escribió en primer lugar)

Lectura sucia

Una transacción lee un objeto escrito por otra en un estado intermedio.

Lectura no repetible

Una transacción lee un objeto dos veces con valores distintos, por haber una actualización intermedia realizada por otra transacción.

Control de la concurrencia

Locks

- Bloqueo compartido (SLOCK): No se requiere el uso exclusivo del objeto.
- Bloqueo exclusivo (XLOCK): Se requiere el uso exclusivo del objeto.

Acciones dentro de una transacción

- Acciones sobre objetos:
  - READ: Lectura de un objeto.
  - WRITE: Escritura de un objeto.
  - XLOCK (eXclusive LOCK): Solicitud uso exclusivo de un objeto.
  - SLOCK (Shared LOCK): Solicitud de uso compartido de un objeto.
  - UNLOCK: Liberación de una solicitud de uso.
- Acciones genéricas:
  - BEGIN
  - COMMIT: Equivale a la liberación de todos los bloqueos realizados por la transacción.
  - ROLLBACK: Equivale a deshacer todas las escrituras realizadas por la transacción y liberar todos sus bloqueos

Transacción bien formada

Todas sus lecturas y escrituras están cubiertas por bloqueos (precedidas por un lock del tipo adecuado y no realizado un unlock).

- READ: cubierta (al menos) por SLOCK.
- WRITE: cubierta por XLOCK.

Transacción en dos fases

Todos los bloqueos preceden a todos los desbloqueos.

1. Fase de crecimiento: adquiere los bloqueos.
2. Fase de contracción: libera los bloqueos.

Grados de aislamiento

Aun con un aislamiento completo no evita las lecturas fantasma asociadas a inserciones y borrado de objetos que deberían estar bloqueados.

Grado 0

No está bien formada ni en dos fases.

Violaciones de aislamiento:

- Actualización perdida
- Lectura sucia
- Lectura no repetible

Grado 1

Bien formada respecto a escritura y escritura en dos fases

Lecturas no comprometidas:

- Lectura sucia
- Lectura no repetible

Grado 2

Bien formado y escritura en dos fases

Lecturas no comprometidas:

- Lectura no repetible

Grado 3

Bien formado y en dos fases

Lectura repetible: Única violación derivada de las lecturas fantasma.

Grados de aislamiento en SQL

Se utiliza en sentencias con cursores

Mantiene el bloqueo compartido sobre el registro al que accede el cursor hasta que se pase al siguiente registro

Update realiza un bloqueo exclusivo

Interbloqueo (Deadlock)

Situación de bloqueo recíproco de recursos, que producen espera ilimitada

Solución más simple: nunca parar. En caso de que se deniegue un bloqueo, ejecutar rollback. Intentar de nuevo la transacción

Detección de interbloqueos por grafos de esperas. Si existe un ciclo hay un interbloqueo.

Tema 5: Seguridad en los SSDD basados en la WWW

Introducción

La seguridad informática es la protección provista a un sistema de información para alcanzar los objetivos de preservar la **integridad, disponibilidad y confidencialidad** de los recursos del sistema, incluyendo software, hardware, firmware, datos/información y comunicaciones.

Seguridad física

- Limitar el acceso al espacio físico para prevenir robo de bienes y entradas no autorizadas
- Protección contra filtrado de información y robo de documentos (Dumpster Diving [Basura])

Seguridad tecnológica

- Seguridad de la aplicación
  - Proceso de verificación de identidad
  - Correcta configuración del servidor
- Seguridad del sistema operativo
- Seguridad de la red
  - Mitigar el tráfico malicioso (Firewall y sistemas de detección de intrusiones)

Política y procedimientos

- Ingeniería social (Phishing)
- Custodiar la información empresarial sensible
- Los empleados deben estar prevenidos, ser entrenados para ser un poco paranoicos y estar vigilantes

Autenticación

Verificar identidad. Existen 3 factores:

Algo que sabes

- Claves (Contraseñas): Simples de implementar y entender, pero fáciles de romper.
- Claves de usar y tirar (one time passwords – OTP, [Autentificadores del móvil])

Algo que tienes

La fortaleza de la autenticación depende de la dificultad de imitar el producto

- Tarjetas OTP
- Tarjeta inteligente (Smart Card):
- Token/Llave: por ejemplo iButton
- Tarjeta de cajero automático

Algo que eres

- Biometría

Autenticación en dos fases vs Autentificación en dos factores

La autenticación en dos fases utiliza 2 métodos del mismo factor (contraseña + OTP) mientras que en dos factores son de distintos factores (contraseña + huella dactilar).

Autorización

Verificar si un usuario (una vez autenticado) tiene permiso para llevar a cabo una acción determinada

- Lista de control de acceso (Access control list – ACL):
  - Mecanismos utilizados por muchos sistemas operativos para determinar si los usuarios están autorizados para llevar a cabo diferentes acciones

- Conjunto de triplas
  - <usuario, recurso, privilegio>
- Privilegios en base a roles
- Mecanismo de control de acceso:
  - • Obligatorio (Mandatory Access Control, MAC): el sistema informático decide quién accede a qué recursos en función de la política MAC del sistema
  - Discrecional (Discretionary Access Control, DAC): los mismos usuarios tiene autorización para determinar qué otros usuarios tienen derecho a acceder a recursos que ellos han creado, usan o poseen.
  - Basado en roles (Role Base Access Control, RBAC): sistema jerárquico en el que los privilegios de un usuario se asignan de acuerdo con su rol (no discrecional).

Conficencialidad

- Preservar las restricciones de autorización para **acceso y revelado de información**, incluyendo medios para proteger la privacidad personal e información propietaria
- La pérdida de confidencialidad implica el acceso o revelado no autorizado de información
- Incluye: Confidencialidad de datos y Privacidad
- Algunas veces se consigue con:
  - Criptografía
  - Esteganografía
  - Control de acceso
  - Vista en BBDD

Integridad

- **Evitar modificaciones** o destrucción no apropiada de los datos, incluyendo asegurar el no-rechazo (non-repudiation) y **autenticidad de los datos**
- Incluye:
  - Integridad de datos: asegurar que la información y los programas son modificados sólo de forma específica y autorizada
  - Integridad del sistema: asegurar que un sistema ejecuta la funcionalidad prevista sin menoscabo, sin manipulación no autorizada (sea con o sin intención)
- Ejemplo: Man in the middle attack
- Verificación de integridad: agregar redundancia a los datos/mensajes:
  - Técnicas: Hashing, Checksums
  - Códigos de Autenticación de mensajes (MACs)

Disponibilidad

Asegurar que la información pueda ser **accedida** y utilizada de forma confiable y en tiempo por **usuarios legítimos**

- Agregar redundancia para evitar un punto único de falla
- Imponer límites a lo que los usuarios legítimos pueden hace

Ejemplo: (D)DOS, Se utiliza malware para enviar un tráfico excesivo al servidor víctima. Servidores saturados no pueden atender peticiones legítimas.

Seguridad desde el diseño

- Diseñar el sistema con la seguridad en mente desde los requisitos no funcionales.
- Definir conceptos de seguridad concretos y medibles
- Conceptos fundamentales

    - Pentesting. Atacar un sistema informático para identificar fallos, vulnerabilidades y demás errores de seguridad existentes, para así poder prevenir los ataques externos.
    - Análisis forense: Conjunto de técnicas científicas y analíticas especializadas en infraestructuras tecnológicas que permiten identificar, preservar, analizar y presentar datos que sean válidos en un proceso legal. Persigue un ciberataque para recopilar datos que sirvan como pruebas judiciales.
- Tipo de ataque en sistemas en los que la validación de usuarios le dice al atacante si el nombre de usuario provisto es correcto (existe) o no

SQL Injection

- Permite recuperar (o manipular) información vital de la base de datos
- No todos los SGBD son igualmente vulnerables
- Contramedidad:
    - Evitar conectarse como root o como propietario
    - Sanitizar entrada de datos: validación de entradas y escape de caracteres
    - Usar prepared statements
    - Mitigar impacto
        - Prevenir el filtrado de meta-información
        - Limitar los privilegios
        - Encriptar datos sensibles
        - Endurecer las defensas del SGBD y del S.O. anfitrión
        - Validar las entradas
        - Tipo de ataque que explota la característica del protocolo HTTP/S de ser stateless. (Usar POST en vez de GET y HTTPS)

Cookies

OWASP (Open Web Application Security Project)

Ejercicio CSS

- Hacer una página web personal estática, recomendablemente añadiendo hojas de estilo CSS, que al menos incluya los siguientes elementos:
    - Un enlace
    - Una imagen (no es necesario que sea de una altísima calidad)
    - Una tabla

CSS

```css
* {
    margin: 0%;
}

header {
  text-align: center;
}

hr {
    margin: 0%;
    border: 2px solid black;
}

h1 {
    background-color: honeydew;
    margin: 0%;
    padding: 0%;
}

#info {
    display: grid;
    grid-template-columns: repeat(auto-fit, minmax(500px, 1fr));
    grid-row-gap: 10px;
    background-color: whitesmoke;
}

#info > div{
    display: grid;
    grid-template-columns: auto minmax(300px, 20%);
    grid-row-gap: 10px;
}

#asignaturas {
    width: 100%;
}

#asignaturas th {
    background-color: honeydew;
}

#asignaturas td {
    background-color: whitesmoke;
}
```

HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <link rel="stylesheet" href="../css/style.css">
    <title>Expediente UAM</title>
</head>
<body>
    <header>
        <a href="http://www.uam.es"><img src="../images/uam.png"></a>
    </header>
    <hr>
    <h1>Expediente académico</h1>
    <hr>
    <div id =  info>
        <div>
            <p>Apellidos y nombre: </p>
            <p>Serrano Salas, Nicolás</p>
        </div>
        <div>
            <p>Dirección de correo:</p>
            <a href="mailto:nicolas.serranos@estudiante.uam.es"><p>nicolas.serranos@estudiante.uam.es</p></a>
        </div>
        <div>
            <p>Centro:</p>
            <p>Escuela Politénica Superior</p>
        </div>
        <div>
            <p>Tipo de estudio:</p>
            <p>Grado</p>
        </div>
        <div>
            <p>Estudios:</p>
            <p>2028 - Grado en Ingeniría Informática</p>
        </div>
    </div>
    <hr>
    <article>
            <table id = asignaturas>
                <tr>
                        <th>Año académ.</th>
                        <th>Descripción</th>
                        <th>Cr.</th>
                        <th>Convocatoria</th>
                        <th>Calificación</th>
                        <th>C.N.</th>
                        <th>Curso</th>
                </tr>
                <tr>
                        <td>2017/18</td>
                        <td>ALG</td>
                        <td>6.0</td>
                        <td>Mayo</td>
                        <td> </td>
                        <td> </td>
                        <td>3</td>
                </tr>
        </table>
    <article>
    <hr>
</body>
</html>
```

Ejercicio validación de formulario

- Realizar un documento HTML con un formulario como el de la figura:
- No se enviarán datos al servidor hasta que:
    - El campo para el nombre tenga un “nombre” válido de al menos 2 caracteres, no permitiéndose caracteres en blanco ni al principio ni al final
    - El campo edad tenga una edad válida
    - El campo e-mail tenga una dirección válida (al menos contenga un carácter “@”)
    - Las validaciones se tienen que hacer de forma programática con JavaScript, no se darán por válidas validaciones en HTML 5

HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <!-- Importamos nuestro javascript-->
    <script type = "text/javascript" src="scripts.js"></script>
</head>

<body>
    <div>
        <form action="https://postman-echo.com/post" onsubmit="return(validateForm(this));" method="post">
            Nombre:
            <input name="username" type="text">
            <br>
            Edad:
            <input name="edad" type="text">
            <br>
            Email:
            <input name="email" type="text">
            <br>
            <input type="submit" value="Enviar consulta">
        </form>
    </div>
</body>
```

JavaScript

```
function validateForm(s) {
    var name = s["username"].value;
    var edad = s["edad"].value;
    var email = s["email"].value;

    if (name.length < 2 || /^\s/.test(name) || /\s$/.test(name)) {
        alert("Nombre invalido");
        return false;
    } else if(/\D/.test(edad) || edad == ""){
        alert("Edad invalida");
        return false;
    } else if(!(/@/.test(email))){
        alert("Email invalido");
        return false;
    }
}
```

Ejercicio MongoDB

Crear una colección de universidades públicas de la ciudad de Madrid que contenga los siguientes documentos:

- Un documento para cada una de las universidades, que contenga: el nombre, la dirección postal, el teléfono, el e-mail de contacto y la página web.
- Se definirán también el nombre y número total de estudiantes de los centros de cada una de las universidades.

```
db.createCollection("universidades")

db.universidades.insertOne({
    nombre: "UAM",
    direccion_postal: "Ciudad Universitaria de Cantoblanco. 28049 Madrid",
    email: "informacion.general@uam.es",
    pagina_web: "http://www.uam.es",
    telefono: "914975000",
    centros: [
        {nombre:"Escuela Politecnica Superior", estudiantes:1198},
        {nombre:"Facultad de Derecho", estudiantes:3193},
        {nombre:"Facultad de Filosofia y Letras", estudiantes:3783},
        {nombre:"Facultad de Ciencias", estudiantes:4285},
        {nombre:"Facultad de Formacion de Profesorado y Educacion", estudiantes:2433},
        {nombre:"Facultad de Medicina", estudiantes:2071},
        {nombre:"Facultad de Ciencias Economicas y Empresariales", estudiantes:2886},
        {nombre:"Facultad de Psicologia", estudiantes:1505}
    ]
})

db.universidades.insertOne({
    nombre: "UCM",
    direccion_postal: "Avenida Seneca 2. 28040 Madrid",
    email: "infocom@ucm.es",
    pagina_web: "http://www.ucm.es",
    telefono: "914520400",
    centros: [
        {nombre:"Facultad de Derecho", estudiantes:5133},
        {nombre:"Facultad de Informatica", estudiantes:1736},
        {nombre:"Facultad de Matematicas", estudiantes:1240},
        {nombre:"Facultad de Farmacia", estudiantes:2350}
    ]
})

db.universidades.insertOne({
    nombre: "UPM",
    direccion_postal: "Paseo Juan XXIII 11. 28040 Madrid",
    email: "informacion@upm.es",
    pagina_web: "http://www.upm.es",
    telefono: "913366000",
    centros: [
        {nombre:"E.T.S. de Ingenieros Informaticos", estudiantes:800},
        {nombre:"E.T.S. de Ingenieros de Telecomunicacion", estudiantes:700}
    ]
})
```

Definir consultas que permitan:

- Mostrar todos los centros de la UAM.

```
db.universidades.find({nombre:"UAM"},{
    _id:0,
    nombre:1,
    "centros.nombre":1
})
```

- Mostrar el total de estudiantes de cada una de las universidades.

```
db.universidades.aggregate([{
    $project: {
        _id:0,
        nombre:1,
        "Total de Estudiantes": {$sum: "$centros.estudiantes"}
    }
}])
```

- Mostrar sólo aquellas universidades que tengan facultades de Derecho.

```
db.universidades.aggregate([
    {$match:{"centros.nombre":{$regex:"derecho", $options: "$i"}}},
    {$group: {
        _id:"$nombre"

     }}
])
```