Desarrollo de Aplicaciones Web Empresariales

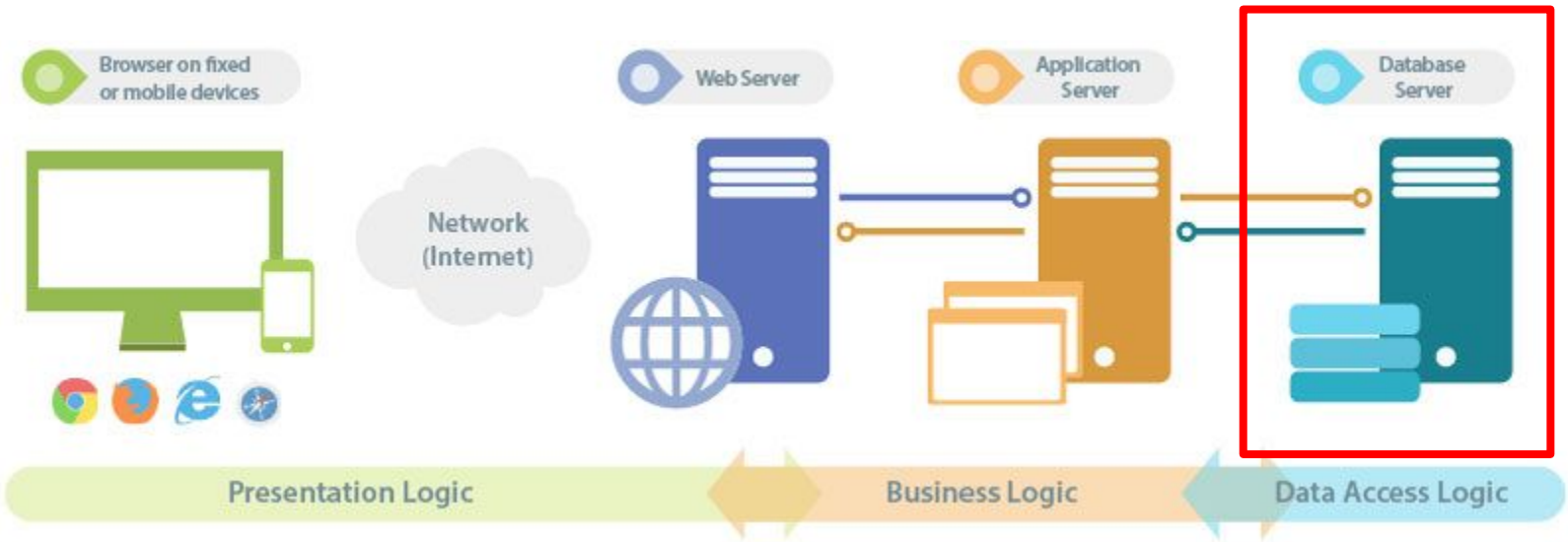
Exequiel Fuentes Lettura
exequiel.fuentes@ucn.cl

Información de contacto

- Exequiel Fuentes Lettura
  - Email: exequiel.fuentes@ucn.cl
  - Horario de Atención: Jueves y Viernes, bloque C

- Departamento de Ingeniería de Sistemas y Computación
  - Oficina: Y1 - 329
  - http://www.disc.ucn.cl

Arquitectura de aplicaciones Web: DB server



Almacenamiento en aplicaciones Web

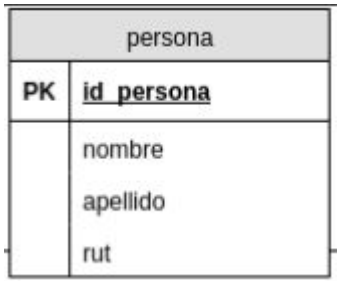
- Esto no es almacenamiento en el lado del cliente, para este tópico vea:
  - https://developer.mozilla.org/en-US/docs/Web/API/Web_Storage_API
- Las propiedades que debe tener son:
  - Siempre disponible.
  - Obtener los datos correctos.
  - Almacenar las actualizaciones.
  - Tolerante a fallos.
  - Escalable.
  - Proporcionar una buena organización de los datos.
  - Etc.

Base de datos relacionales

- Los datos están organizados como una serie de tablas.
- Una tabla está constituida por filas (registros, tuplas) y columnas.
- Una fila tiene un número fijo de columnas.
- Una columna tiene diferentes tipos:
  - String
  - Integer
  - Floating
  - Date
  - Otros (dependiendo qué tipos de datos soporte la base de datos)

Esquema de base de datos

- Esquema: la estructura de la base de datos
  - Nombre de las tablas (usuario, comentarios, etc)
  - Nombre y tipos de las columnas de la tabla.
  - Información adicional opcional (constraints, stored procedures, etc)




```
                                        Table "p
  Column    |          Type          |
------------+------------------------+----------
 id_persona | integer                | not null
 nombre     | text                   | not null
 apellido   | text                   | not null
 rut        | character varying(9)   | not null
```

```
 id_persona |  nombre  | apellido  |    rut
------------+----------+-----------+-----------
          1 | Exequiel | Fuentes   | 1
          2 | Brian    | Keith     | 1
          3 | Mario    | Camacho   | 1
          4 | Rodrigo  | Chacon    | 1
          5 | Pablo    | Tapia     | 1
          6 | Osvaldo  | Hernandez | 1
```

Structured Query Language (SQL)

- Es el estándar para acceder a los datos relacionales
  - Teoría detrás: álgebra relacional
- Consultas:
  - Hay muchas maneras para extraer información.
  - Se puede especificar lo que se quiere.
  - El sistema de la base de datos encuentra la manera de obtener la información eficientemente.
  - Busca los datos por el contenido, no sólo por el nombre.

Ejemplos de comandos SQL

```
% Elimina la base de datos colegio si existe
DROP DATABASE IF EXISTS universidad;

% Crea la base de datos colegio
CREATE DATABASE universidad;

% Crea tabla persona
CREATE TABLE persona (
  id_persona SERIAL PRIMARY KEY,
  nombre TEXT NOT NULL CHECK (nombre <> ''),
  apellido TEXT NOT NULL CHECK (apellido <> ''),
  rut varchar(9) NOT NULL CHECK (rut <> '')
);
```

```
% Insertamos una persona
INSERT INTO persona (nombre, apellido, rut) VALUES
  ('Exequiel', 'Fuentes', '144338553');

% Borramos datos
DELETE FROM persona WHERE nombre=’Exequiel’;

% Lista los datos
SELECT * FROM persona;
```

Índices y claves

- Considera la consulta: SELECT * FROM persona WHERE id = 1;
- La base de datos puede obtener los datos usando un índice si este fue definido.
  - CREATE INDEX ON persona (rut);
- El uso de clave le dice a la base de datos que se debe crear un índice:
  - Primary key.
  - Foreign key.
- Notas adicionales: http://stackoverflow.com/a/1130/1347377

Object Relational Mapping (ORM)

- Modelo de relación y SQL puro en lenguajes de alto nivel no coinciden muy bien
  - Objetos versus tablas?
- En la segunda generación de framework para aplicaciones web se introdujo una forma de mapear objetos y base de datos.
- Maneja la creación de los esquemas y los comando SQL utilizando una interfaz.
- Ejemplo: Active Record en Rails, Django ORM, etc.

Object Relational Mapping (ORM)

```ruby
def authenticate
  params.require(:password)
  @cuenta = Cuenta.find_or_create(params[:password])
  redirect_to action: "show", direccion: @cuenta.direccion
end
```

```ruby
class CreateSessions < ActiveRecord::Migration
  def change
    create_table :sessions do |t|
      t.column :session_id, :string, :null => false
      t.column :data, :text, :null => true
      t.column :person_fk, :integer, :null => true, :default => nil
      t.column :force_close, :boolean, :default => false

      # Create created_at and updated_at columns
      t.timestamps
    end
  end
end
```

NoSQL - MongoDB

- Base de datos tradicionales proporcionaban una manera confiable de almacenamiento para las primeras generaciones de aplicaciones web.
- Un nuevo tipo de base de datos es necesario para modelos basados en aplicaciones web. Este nuevo tipo es conocido por NoSQL (no relacionales).
  - Datos no estructurados.
  - Consultas hechas a colecciones de documentos.
  - Velocidad y escalabilidad es imperativa.
- Ejemplo: MongoDB, BigTable, Redis, RavenDb, Cassandra, Hbase, Neo4j y CouchDb.

Esquema en NoSQL - MongoDB

- JSON proporciona una manera flexible para representar documentos.
- Aunque pueden haber algunos problemas, pero se resuelven con validaciones.
  - Considera: <h1>Hello {{person.informalName}}</h1>

    Bueno: typeof person.informalName == 'string' and length < unnumero

    Malo: El tipo puede ser un objeto de 1 GB o undefined o null, etc.
- Existen varios drivers para conectarse a una base de datos MongoDB, entre ellos: Mongoose (http://mongoosejs.com/index.html)
  - Utiliza un familiar ORM y lo mapea en Mongoose.
  - Ayuda a encapsular interfaces de bajo nivel de MongoDB para que sea más fácil de utilizar.

Conectarse a una BD en MongoDB

1. Conectar a una instancia MongoDB

```
mongoose.connect('mongodb://localhost/dawedb');
```

2. Esperar que la conexión se haya completado. Mongoose exporta un EventEmitter

```
mongoose.connection.on('open', function () {
    // Can start processing model fetch requests
});
mongoose.connection.on('error', function (err) { });
```

También espera eventos de tipo: connecting, connected, disconnecting, disconnected, etc.

Mongoose: Definición del esquema

```
const mongoose = require('mongoose');
const Schema = mongoose.Schema;

const blogSchema = new Schema({
  title:  String,
  author: String,
  body:   String,
  comments: [{ body: String, date: Date }],
  date: { type: Date, default: Date.now },
  hidden: Boolean,
  meta: {
    votes: Number,
    favs:  Number
  }
});
```

Mongoose: Definición del esquema

- SchemaTypes permitidos:
  - String
  - Number
  - Date
  - Buffer
  - Boolean
  - Mixed
  - ObjectId
  - Array
- Leer más acá: http://mongoosejs.com/docs/schematypes.html

Esquema permite índices y valores por defecto

- Índice simple
  - first_name: {type: 'String', index: true}
- Índice con valor único
  - user_name: {type: 'String', index:{unique: true} }
- Valor por defecto
  - date: {type: Date, default: Date.now }

Siguiendo con índices

- Hay un trace-off: Desempeño versus espacio:
  - Consultas más rápidas: Elimina escanear, la base de datos retorna la coincidencia a partir del índice.
  - Operaciones que hacen cambios en los valores son más lentas: agregar, borrar, actualizar deben actualizar el índice también.
  - Usa más espacio: Se necesita almacenar los índices.
- Cuando utilizar:
  - Consulta gasta demasiado tiempo en ejecutarse.
  - Se necesita que los valores sean únicos.

Mongoose: Crear un modelo desde un esquema

- Un modelo en Mongoose es un constructor de objetos de una colección. Puede o no corresponder a un modelo MVC.
  - const User = mongoose.model('User', userSchema);
- Crear objetos desde un modelo:
  - User.create({ first_name: 'Ian', last_name: 'Malcolm'}, doneCallback);

    function doneCallback(err, newUser) {

        assert (!err);

        console.log('Created object with ID', newUser._id);

    }

Modelo usado para hacer consultas

- Retornar la colección User:
  - User.find(function (err, users) {/*users is an array of objects*/ });
- Retornar un objeto user dado su user_id
  - User.findOne({_id: user_id}, function (err, user) { /* ... */ });
- Actualizar un objeto user dado su user_id
  - User.findOne({_id: user_id}, function (err, user) {

    // Update user object

    user.save();

    });
- Para más información: http://mongoosejs.com/docs/queries.html

Otras operaciones en Mongoose

const query = User.find({});

- Proyección:
  - query.select("first_name last_name").exec(doneCallback);
  - https://docs.mongodb.com/v3.2/tutorial/project-fields-from-query-results/
- Ordenar:
  - query.sort("first_name").exec(doneCallback);
- Límite:
  - query.limit(50).exec(doneCallback);

Eliminando objetos de una colección

- Eliminar un sólo objeto dado user_id:
  - User.remove({_id: user_id}, function (err) { });
- Eliminar todos los objetos User:
  - User.remove({}, function (err) { });

- Para hacer más pruebas ver archivo notas.txt

¿Preguntas?