

Bases de datos

Objeto-Relacionales

José R.R. Viqueira

Centro Singular de Investigación en Tecnoloxías Intelixentes (CITIUS)
Rúa de Jenaro de la Fuente Domínguez,
15782 - Santiago de Compostela.

Despacho: 209

Telf: 881816463

Mail: jrr.viqueira@usc.es

Skype: jrviqueira

URL: <https://citius.gal/team/jose-ramon-rios-viqueira>

Curso 2023/2024

- **Introducción**
- **Estructuras de datos complejas**
- **Referencias**
- **Herencia**
- **Representaciones abiertas**
 - ▷ **XML**
 - ▷ **JSON**

Introducción

Estr. de datos
complejas

Referencias

Herencia

XML

JSON

■ Objetivo

- ▷ Extender el modelo relacional para incorporar las características deseables de la orientación a objetos
- ▷ Conservar el modelo compatible con el modelo relacional
 - _ No perder 35 años de investigación en el modelo relacional
 - Con un **fundamento teórico muy sólido**

■ Generaciones de SGBDs

- ▷ **Pre-relacionales**
 - _ Modelos jerárquicos y Modelos en red
- ▷ **Relacionales**
 - _ Modelo relacional y Lenguaje SQL
- ▷ **Objeto-relacionales** (finales de los 90)
 - _ Extensión del modelo relacional
 - Tipos de dato: Estructuras de datos más complejas y operaciones
 - Incorporación de objetos: tablas con tipo como clases de objetos (referencias)
 - _ Extensión del lenguaje SQL
 - A partir del SQL:1999 y SQL:2003
 - Incorpora otras características no O-R (Recursividad)

La persistencia en
Lenguajes de
Programación O.O. no
se está considerando
como un SGBD



■ Tipos de dato

▷ Tipos incorporados (Built-in)

- Boolean, Char, Varchar, Numeric, BLOB, CLOB, Date, Time, Timestamp, Interval, etc.
- 10 categorías de tipos con tipado fuerte en cada categoría

▷ Constructores de tipo

– Definición de datos

```
CREATE TABLE departamento (
  id_dep INTEGER PRIMARY KEY,
  nombre VARCHAR(50),
  direccion ROW(calle VARHCAR(100), num INTEGER, loc VARCHAR(10),
  presupuesto DECIMAL(12, 2),
);
```

```
CREATE TABLE empleado (
  id_emp INTEGER PRIMARY KEY,
  nombre VARCHAR(50),
  direccion ROW(calle VARHCAR(100), num INTEGER, loc VARCHAR(10),
  salario DECIMAL(9, 2),
  hijos VARCHAR(50) ARRAY,
  cursos ROW(nombre VARCHAR(50), nota DECIMAL(3, 1)) MULTISET
  dep INTEGER,
  FOREIGN KEY dep REFERENCES departamento(id_dep)
);
```

■ Tipos de dato

▷ Constructores de tipo

– Consultas

Desanidar

```
SELECT e.nombre, h.hijo
FROM empleado AS e, UNNEST (e.hijos) AS h(hijo)
```

```
SELECT e.nombre, AVG(c.nota)
FROM empleado AS e, UNNEST (e.cursos) AS c(nota)
WHERE e.direccion.loc = 'Santiago' and c.nota >= 5
GROUP BY e.nombre
```

Anidar

```
SELECT d.nombre, d.presupuesto,
       MULTISET(SELECT e2.nombre, e2.salario, count(h.hijo) AS Hijos
                FROM empleado AS e2, UNNEST (e2.hijos) AS h(hijo)
                WHERE d.id_dep = e2.dep
                GROUP BY e2.nombre, e2.salario),
       SUM(e.salario) AS SalarioTotal
FROM departamento AS d, empleado AS e
WHERE d.id_dep = e2.dep
GROUP BY d.nombre, d.presupuesto
HAVING d.presupuesto > SUM(e.salario)
```

```
CREATE TABLE departamento (
  id_dep INTEGER PRIMARY KEY,
  nombre VARCHAR(50),
  direccion ROW(calle VARCHAR(100), num INTEGER, loc VARCHAR(10),
  presupuesto DECIMAL(12, 2),
);
```

```
CREATE TABLE empleado (
  id_emp INTEGER PRIMARY KEY,
  nombre VARCHAR(50),
  direccion ROW(calle VARCHAR(100), num INTEGER, loc VARCHAR(10),
  salario DECIMAL(9, 2),
  hijos VARCHAR(50) ARRAY,
  cursos ROW(nombre VARCHAR(50), nota DECIMAL(3, 1)) MULTISSET
  dep INTEGER,
  FOREIGN KEY dep REFERENCES departamento(id_dep)
);
```



■ Tipos de dato

▷ Tipos estructurados

– Definición de las estructuras de datos

```
CREATE TYPE direccion AS (
  calle VARCHAR (50),
  num INTEGER,
  loc VARCHAR (15)
) NOT FINAL;
```

```
CREATE TYPE empleado AS (
  nombre VARCHAR (50),
  dir DIRECCION,
  salario_base DECIMAL (9, 2),
  complementos DECIMAL (9,2)
) NOT FINAL;
```

```
CREATE TABLE departamento (
  nombre VARCHAR (50),
  dir DIRECCION,
  presupuesto DECIMAL(10,2),
  secretario EMPLEADO
);
```



- Tipos de dato
 - ▷ Tipos estructurados
 - Definición de los métodos

```
CREATE TYPE empleado AS (
  nombre VARCHAR (50),
  dir DIRECCION,
  salario_base DECIMAL (9, 2),
  complementos DECIMAL (9,2))
INSTANTIABLE NOT FINAL
INSTANCE METHOD salario()
RETURNS DECIMAL(9, 2);

CREATE INSTANCE METHOD salario()
RETURNS DECIMAL (9, 2) FOR empleado
BEGIN
  RETURN SELF.salario_base +
         SELF.complementos
END
```

```
SELECT nombre, secretario.salario()
FROM Departamento
```



■ Tipos de dato

▷ Tipos estructurados

– Definición de los métodos (Constructores)

```
CREATE TYPE circulo AS (
    radio DECIMAL(5, 1))
INSTANTIABLE NOT FINAL
CONSTRUCTOR METHOD circulo(
    radio DECIMAL(5, 1))
RETURNS circulo
SELF AS RESULT;

CREATE METHOD circulo(
    radio DECIMAL(5, 1))
RETURNS circulo FOR circulo
BEGIN
    SET SELF.radio = radio
    RETURN SELF
END
```

```
INSERT INTO circulos
VALUES (3, NEW circulo(5));
```




■ Tipos de dato

▷ Tipos estructurados

– Definición de los métodos (Observers y Mutators)

```
CREATE TYPE empleado AS (
  nombre VARCHAR (50),
  dir DIRECCION,
  salario_base DECIMAL (9, 2),
  complementos DECIMAL (9,2))
INSTANTIABLE NOT FINAL
INSTANCE METHOD salario()
RETURNS DECIMAL(9, 2);
```

```
SELECT secretario.nombre()
FROM Departamento
WHERE secretario.salario_base > 34
```

```
UPDATE Departamento
SET secretario.nombre = 'Juan'
WHERE secretario.salario() < 123
```

```
INSERT INTO Departamento (nombre,
secretario)
VALUES (
'Ventas',
NEW empleado().nombre('Juan')
)
```



■ Tipos de dato

▷ Tipos estructurados

– Definición de columnas de tablas y atributos de otros tipos

Definición de datos

```
CREATE TYPE punto AS (  
  x FLOAT, y FLOAT)
```

```
CREATE TYPE rectangulo AS (  
  x1 FLOAT, y1 FLOAT,  
  x2 FLOAT, y2 FLOAT)
```

```
CREATE TYPE linea AS (  
  mbr rectangulo,  
  coords punto ARRAY)  
INSTANCE METHOD longitud()  
RETURNS FLOAT
```

```
CREATE TYPE poligono AS (  
  mbr rectangulo,  
  coords punto ARRAY)  
INSTANCE METHOD area()  
RETURNS FLOAT
```

```
CREATE FUNCTION distancia (g1 geo, g2 geo) RETURNS FLOAT  
BEGIN ... END
```

Consultas

```
SELECT sum(trazado.longitud)  
FROM carreteras  
WHERE propietario = 'Municipal' and estado = 'Malo'
```

```
SELECT sum(camas)  
FROM carreteras AS c, centros_salud AS cs  
WHERE c.estado = 'bueno' and c.longitud < 2000  
and distancia(c.trazado, cs.posicion) < 1000
```

■ Tipos de dato

▷ Tipos estructurados

– Definición de los tipos de las filas de una tabla (Tablas con tipo)

```
CREATE TYPE direccion AS (
  calle VARCHAR (50),
  num INTEGER,
  loc VARCHAR (15)) NOT FINAL;
```

```
CREATE TYPE empleado AS (
  id_emp VARCHAR (8),
  nombre VARCHAR (50),
  salario DECIMAL (9, 2),
  dir DIRECCION)
NOT FINAL
REF IS SYSTEM GENERATED;
```

REF USING INTEGER
REF FROM (id_emp)

Opción por defecto

```
CREATE TABLE empleados
OF empleado
(REF IS oid SYSTEM GENERATED)
```

REF IS oid USER GENERATED
REF IS oid DERIVED

Introducción

Estr. de datos
complejas

Referencias

Herencia

XML

JSON

■ Tipos Referencia

▷ Definición de datos

```
CREATE TYPE Empleado AS (
  nombre VARCHAR (50),
  salario_base DECIMAL (9, 2),
  complementos DECIMAL(9, 2),
  dep REF(Departamento))
NOT FINAL
REF IS SYSTEM GENERATED
INSTANCE METHOD salario()
RETURNS DECIMAL(9, 2);
```

```
CREATE TABLE Empleados
OF Empleado
(REF IS oid SYSTEM GENERATED,
dept WITH OPTIONS SCOPE Departamentos);
```

```
CREATE TYPE Departamento AS (
  nombre VARCHAR (50),
  dir DIRECCION,
  emps REF(Empleado) MULTISSET,
  director REF(Empleado))
NOT FINAL
REF IS SYSTEM GENERATED;
```

```
CREATE TABLE Departamentos
OF Departamento
(REF IS oid SYSTEM GENERATED,
emps WITH OPTIONS SCOPE Empleados,
director WITH OPTIONS SCOPE Empleados);
```

Introducción

Estr. de datos
complejas

Referencias

Herencia

XML

JSON

■ Tipos Referencia

▷ Derreferenciación de atributos

```
SELECT e.nombre, e.dept->nombre
FROM Empleados e
WHERE e.dept->dir.loc = 'Santiago'
```

```
SELECT e.nombre, Deref(e.dept).nombre
FROM Empleados e
WHERE Deref(e.dept).dir.loc = 'Santiago'
```

```
SELECT e.nombre, (SELECT d.nombre FROM departamentos d WHERE d.oid = e.dept)
FROM Empleados e
WHERE (SELECT dir.loc FROM departamentos d WHERE d.oid = e.dept) = 'Santiago'
```

```
SELECT e.nombre, d.nombre
FROM Empleados e LEFT JOIN Departamentos d ON (e.dept = d.oid)
WHERE d.dir = 'Santiago'
```

▷ Derreferenciación de métodos

```
SELECT d.nombre, SUM(e.ptr->salario)
FROM Departamentos d, UNNEST(d.emps) AS e(ptr)
GROUP BY d.nombre
```

```
SELECT e.nombre, e.salario
FROM Empleados e
```

```
SELECT e.nombre, Deref(e.oid).salario
FROM Empleados e
```

Introducción

Estr. de datos
complejas

Referencias

Herencia

XML

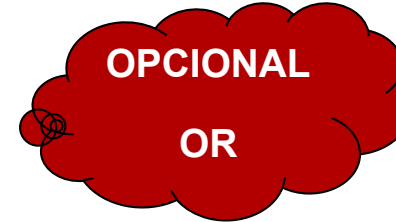
JSON

■ Herencia entre tipos estructurados

▷ Características

– Herencia simple

- Un valor solo tiene un tipo más específico (subtipos disjuntos)



```
CREATE TYPE rectangulo AS (  
  x1 FLOAT, y1 FLOAT, x2 FLOAT, y2 FLOAT)  
NOT FINAL  
INSTANCE METHOD area() RETURNS FLOAT;
```

```
CREATE TYPE cuadrado  
UNDER rectangulo (tamano FLOAT)  
NOT FINAL  
OVERRIDING METHOD area() RETURNS FLOAT;
```

```
CREATE TABLE rectangulos AS (  
  id_rect INTEGER PRIMARY KEY,  
  rec rectangulo)
```

Introducción

Estr. de datos
complejas

Referencias

Herencia

XML

JSON

■ Herencia entre tipos estructurados

▷ Sobrecarga de operadores

```
CREATE TYPE geom AS (  
  mbr rectangulo)  
NOT INSTANTIABLE NOT FINAL;
```

```
CREATE TYPE poligono UNDER geom (  
  coords FLOAT ARRAY)  
INSTANTIABLE NOT FINAL  
INSTANCE METHOD area() RETURNS FLOAT;
```

```
CREATE TYPE rectangulo UNDER geom (  
  x1 FLOAT, y1 FLOAT, x2 FLOAT, y2 FLOAT)  
INSTANTIABLE NOT FINAL  
OVERRIDING METHOD area() RETURNS FLOAT;
```

```
CREATE METHOD area() FOR rectangulo  
BEGIN  
  RETURN (SELF.x2 - SELF.x1) *  
         (SELF.y2 - SELF.y1);  
END
```

```
CREATE TYPE cuadrado  
UNDER rectangulo (tamano FLOAT)  
INSTANTIABLE NOT FINAL  
OVERRIDING METHOD area() RETURNS FLOAT;
```

```
CREATE METHOD area()  
FOR cuadrado  
BEGIN  
  RETURN tamano * tamano;  
END
```

Introducción

Estr. de datos
complejas

Referencias

Herencia

XML

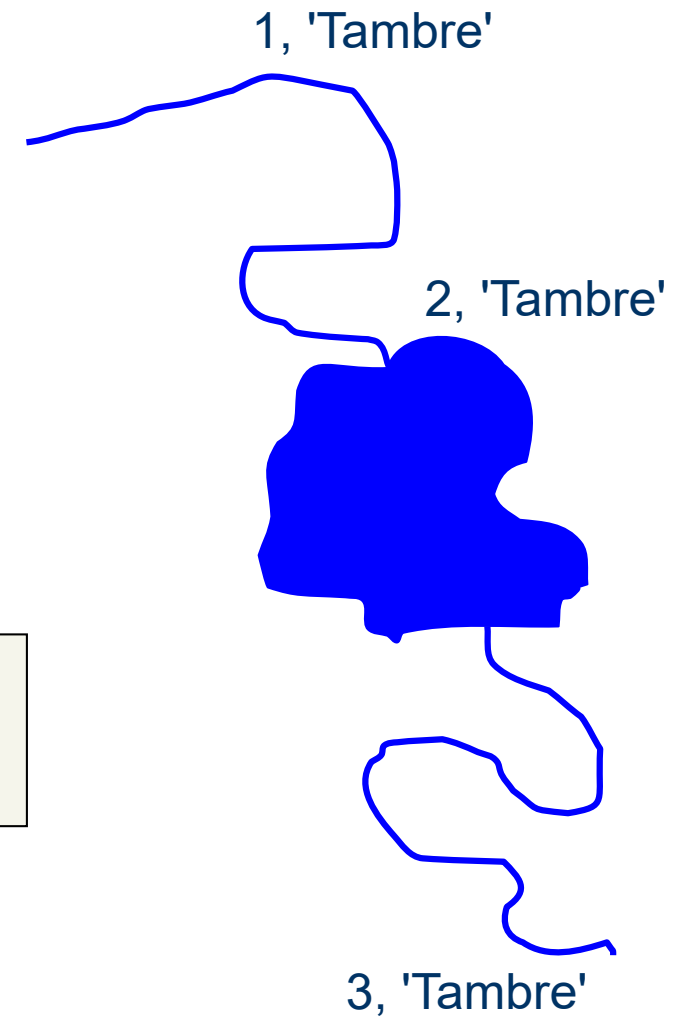
JSON

■ Herencia entre tipos estructurados

▷ Uso de un tipo concreto

```
CREATE TABLE rios AS (
  id_rio INTEGER PRIMARY KEY,
  nombre VARCHAR(50),
  geom geo)
```

```
SELECT nombre, SUM(TREAT(geom AS linea).longitud)
FROM rios
WHERE geom IS OF (linea)
GROUP BY nombre
```



Introducción

Estr. de datos
complejas

Referencias

Herencia

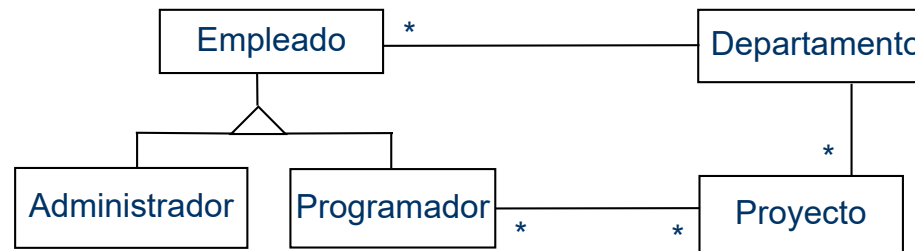
XML

JSON

■ Herencia entre tipos estructurados

▷ Herencia entre tablas

– Motivación



RELACIONAL

EMPLEADOS

id_emp	nombre	salario	dep
1	Juan	1500	3
2	Elisa	2100	5

DEPARTAMENTOS

id_dep	nombre	presup
3	Ventas	345466
5	Producción	216500

ADMINISTRADORES

id_emp	Sistema
2	Linux

PROG-PROY

id_emp	id_proy
1	23

PROGRAMADORES

id_emp	Lenguaje
1	Pascal

PROYECTOS

id_proy	duracion	presup	dep
23	2	4643	3
34	3	2234	3

Introducción

Estr. de datos
complejas

Referencias

Herencia

XML

JSON

■ Herencia entre tipos estructurados

▷ Herencia entre tablas

– Definición de datos

```
CREATE TYPE empleado AS (
  nombre VARCHAR(50),
  salario DECIMAL(6, 2),
  dep REF(departamento))
INSTANTIABLE NOT FINAL
```

```
CREATE TYPE administrador UNDER
empleado (
  sistema VARCHAR(20))
INSTANTIABLE NOT FINAL
```

```
CREATE TYPE programador
UNDER empleado (
  lenguaje VARCHAR(20),
  proys REF(proyecto) MULTISSET)
INSTANTIABLE NOT FINAL
```

```
CREATE TYPE departamento AS (
  nombre VARCHAR(20),
  presup DECIMAL(9, 2),
  emps REF(empleado) MULTISSET,
  proys REF(proyecto) MULTISSET)
INSTANTIABLE NOT FINAL
```

```
CREATE TYPE proyecto AS (
  duracion INTEGER,
  presup DECIMAL(9, 2),
  dep REF(departamento),
  progs REF(programador) MULTISSET)
INSTANTIABLE NOT FINAL
```

```
CREATE TABLE EMPLEADOS OF empleado
(REF IS id_emp SYSTEM GENERATED,
dep WITH OPTIONS SCOPE DEPARTAMENTOS);
```

```
CREATE TABLE ADMINISTRADORES OF administrador
UNDER EMPLEADOS;
```

```
CREATE TABLE PROGRAMADORES OF programador
UNDER EMPLEADOS
(proys WITH OPTIONS SCOPE PROYECTOS);
```

```
CREATE TABLE DEPARTAMENTOS OF departamento
(REF IS id_dep SYSTEM GENERATED,
emps WITH OPTIONS SCOPE EMPLEADOS,
proys WITH OPTIONS SCOPE PROYECTOS);
```

```
CREATE TABLE PROYECTOS OF proyecto
(REF IS id_proy SYSTEM GENERATED,
dep WITH OPTIONS SCOPE DEPARTAMENTOS,
progs WITH OPTIONS SCOPE PROGRAMADORES);
```

Introducción

Estr. de datos
complejas

Referencias

Herencia

XML

JSON



■ Herencia entre tipos estructurados

▷ Herencia entre tablas

– Consultas

- Al seleccionar en subtablas, se realiza automáticamente el **join** con la supertabla.

```
SELECT *  
FROM Empleados
```

```
SELECT p.nombre, p.lenguaje  
FROM Programadores
```

```
SELECT *  
FROM ONLY (Empleados)
```

```
SELECT *  
FROM Empleados  
WHERE Deref(oid) IS OF (Empleado)
```

Introducción

Estr. de datos
complejas

Referencias

Herencia



XML

JSON

■ Herencia entre tipos estructurados

▷ Herencia entre tablas

– Inserciones, modificaciones y borrados

■ Inserción

- Se permiten inserciones en las supertablas (**herencia opcional**)
- Inserción en la subtabla inserta automáticamente en la supertabla

```
INSERT INTO PROGRAMADORES
(nombre, salario, lenguaje)
VALUES ('Felipe', 2450, 'Java')
```

■ Borrado

- Siempre se realizan borrados en cascada de forma automática

```
DELETE FROM EMPLEADOS
WHERE nombre = 'Felipe'
```

```
DELETE FROM PROGRAMADORES
WHERE nombre = 'Felipe'
```

■ Modificación

- Modificación en la supertabla se puede hacer a través de la subtabla.

```
UPDATE ADMINISTRADORES
SET salario = salario + salario*0.25
WHERE dept->nombre = Redes'
```

Introducción

Estr. de datos
complejas

Referencias

Herencia

XML

JSON

■ Extensible Markup Language (XML)

- ▷ Metalenguaje: Reglas para definir lenguajes (cada uno con su vocabulario)
- ▷ Combina datos con etiquetas (Metadatos)
- ▷ Para humanos y máquinas

**Declaración
(Opcional)**

Comentario

```
<?xml version="1.0"?>
<!-- Esta es una representación XML de la tabla de empleados -->
<Empleados>
  <Empleado>
    <Nombre>Alberto</Nombre>
    <DNI>34233456-D</DNI>
    <Edad>35</Edad>
    <Sueldo Moneda ="Euro">1200</Sueldo>
  </Empleado>
  <Empleado>
    <Nombre>Inés</Nombre>
    <DNI>31245659-D</DNI>
    <Edad>29</Edad>
    <Sueldo Moneda ="Peseta">180000</Sueldo>
  </Empleado>
</Empleados>
```

Atributo

Elemento

Introducción

Estr. de datos
complejas

Referencias

Herencia

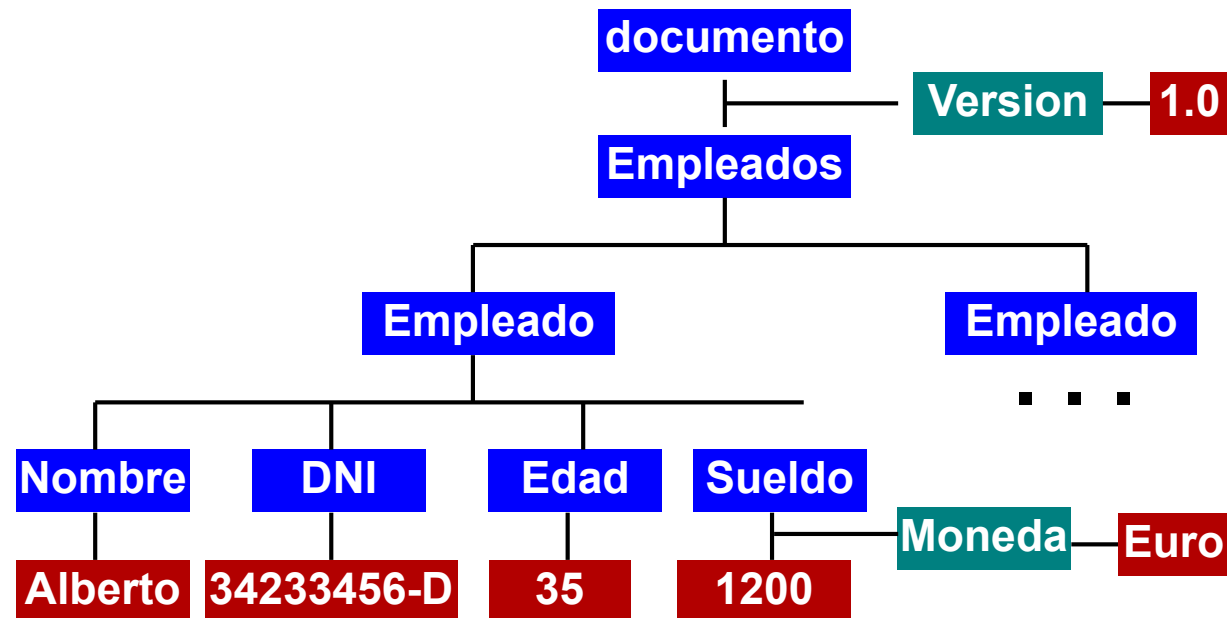
XML

JSON

■ Extensible Markup Language (XML)

▷ Estructura **Jerárquica**

- Raíz o nodo documento
- Elemento raíz
- Secuencia de hijos
- Hojas
 - caracteres, atributos, comentarios, etc.



Introducción

Estr. de datos
complejas

Referencias

Herencia

XML

JSON

■ Extensible Markup Language (XML)

▷ Document Type Definition (DTD)

- Documentos bien formados: 1 raíz, anidamiento correcto, etc.
- Documentos válidos: bien formados + cumplen un DTD

DTD define el
vocabulario del
lenguaje

Orden importa

```
<?xml version="1.0"?>
<!-- Esta es una representación
      XML de la tabla de empleados
-->
<Empleados>
  <Empleado>
    <Nombre>Alberto</Nombre>
    <DNI>34233456-D</DNI>
    <Edad>35</Edad>
    <Sueldo Moneda ="Euro">
      1200
    </Sueldo>
  </Empleado>
  <Empleado>
    <Nombre>Inés</Nombre>
    <DNI>31245659-D</DNI>
    <Edad>29</Edad>
    <Sueldo Moneda ="Peseta">
      180000
    </Sueldo>
  </Empleado>
</Empleados>
```

```
<!ELEMENT Empleados (Nota?, Empleado*)>
<!ELEMENT Empleado (Nombre, DNI, Edad, Sueldo)>
<!ELEMENT Nombre (#PCDATA)>
<!ELEMENT DNI (#PCDATA)>
<!ELEMENT Edad (#PCDATA)>
<!ELEMENT Sueldo (#PCDATA)>
<!ATTLIST Sueldo
  Moneda (Euro | Peseta) #REQUIRED>
```

Parsed Character Data

Cardinalidad :

- ? Opcional
- * Cero o más
- + uno o más
- Por defecto, Uno

Atributos:

Por defecto opcionales

```
<!ATTLIST Sueldo Moneda>
```

Valor por defecto

```
<!ATTLIST Sueldo
  Moneda
  (Euro | Peseta) Euro>
```

Introducción

Estr. de datos
complejas

Referencias

Herencia

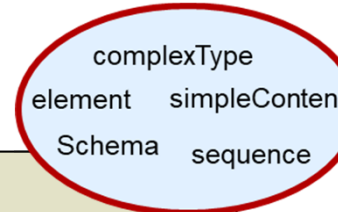
XML

JSON

■ XML Schema

- ▷ Es un lenguaje XML (no como el DTD)
- ▷ Mucho más expresivo

<http://www.w3.org/2001/XMLSchema>



<http://www.empleados.es>



```

<?xml version="1.0" encoding="UTF-8"?>
<!-- Schema para Empleados-->
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.usc.es/ribdd/empleados"
  xmlns="http://www.usc.es/ribdd/empleados"
  elementFormDefault="qualified">
  <xsd:element name="Empleados">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="Empleado"
          type="tipoEmpleado"
          minOccurs="0"
          maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:complexType name="tipoEmpleado">
    <xsd:sequence>
      <xsd:element name="Nombre" type="xsd:string"/>
      <xsd:element name="DNI" type="tipoDNI"/>
      <xsd:element name="Edad" type="xsd:integer"/>
      <xsd:element name="Sueldo" type="tipoSueldo"/>
    </xsd:sequence>
  </xsd:complexType>

```

```

<xsd:simpleType name="tipoDNI">
  <xsd:restriction base="xsd:string">
    <xsd:pattern value="\d{8}-[a-z]{1}" />
  </xsd:restriction>
</xsd:simpleType>
<xsd:complexType name="tipoSueldo">
  <xsd:simpleContent>
    <xsd:extension base="xsd:decimal">
      <xsd:attribute name="Moneda"
        type="tipoMoneda"
        use="required"/>
    </xsd:extension>
  </xsd:simpleContent>
</xsd:complexType>
<xsd:simpleType name="tipoMoneda">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="Euro"/>
    <xsd:enumeration value="Peseta"/>
  </xsd:restriction>
</xsd:simpleType>

```


Introducción

Estr. de datos
complejas

Referencias

Herencia

XML



JSON

■ Lenguajes de búsqueda y consulta

▷ XPath

- _ Expresiones formadas por conjuntos de pasos
 - /paso/paso/.../paso
- _ Cada paso tres componentes: **axisName::nodetest[predicate]**
 - axisName: dirección de navegación (child, parent, descendant, attribute, etc.)
 - child es la opción por defecto
 - nodetest: tipo de nodo y etiqueta del nodo
 - child::* (cualquier hijo), attribute::* (cualquier atributo), child::text() (hijos de tipo texto)
 - / (nodo raíz), // (cualquier nodo), . (nodo actual), .. (nodo padre), @ (atributo)
 - Predicate: Número de posición o expresión de tipo booleano
- _ Ejemplos
 - /Empleados/Empleado[1]/Nombre/text()
 - /Empleados/Empleado[Edad>34]/Nombre
 - //Empleado[@Sueldo>120]

Introducción

Estr. de datos
complejas

Referencias

Herencia

XML



JSON

■ Lenguajes de búsqueda y consulta

▷ XQuery

```
for $variable1 at $pos in doc("localizacion del documento") /xpath
let $variable2 := valor
where condición
order by expresión, expresión descending, expresión ascending
return expresión
```

```
<ul>
{
for $e in doc("empleados.xml") /Empleados/Empleado
where $e/Edad/data() > 58
order by $e/@Sueldo descending
return <li>{$e/Nombre/text()}</li>
}
</ul>
```

Introducción

Estr. de datos
complejas

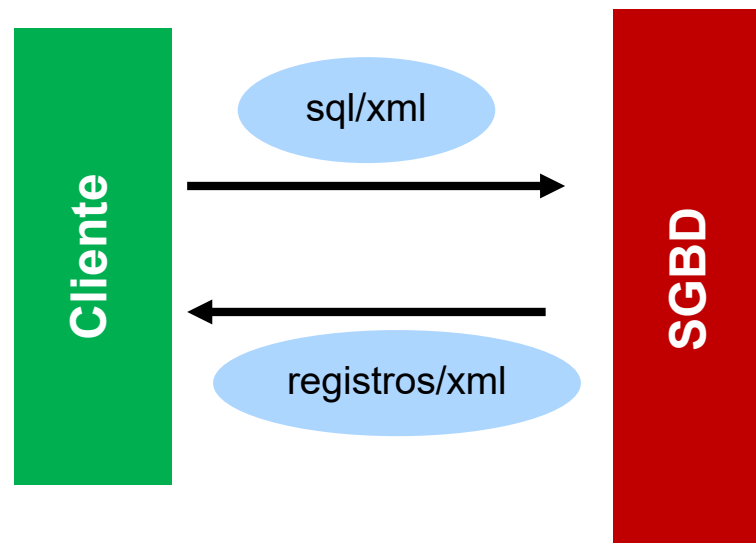
Referencias

Herencia

XML

JSON

- Soporte en sistemas de bases de datos
 - ▷ Bases de datos XML nativas (XPath / XQuery)
 - ▷ Soporte XML en ISO SQL
 - _ Tipo de datos XML
 - _ Métodos para crear XML a partir de datos relacionales
 - Creación de elementos
 - Concatenación de partes de documentos
 - Agregación de elementos
 - Etc.
 - _ Métodos para acceder a partes de un documento XML (XPath, XQuery)



Departamento

id	Nombre	Empleados
d1	Ventas	<Empleados...
d2	Publicidad	<Empleados...
d3	Infraestructuras	<Empleados...

Introducción

Estr. de datos
complejas

Referencias

Herencia

XML

JSON



■ JavaScript Object Notation (JSON)

- ▷ Formato abierto (texto) de intercambio de datos
- ▷ Para humanos y máquinas
- ▷ Subconjunto de JavaScript Programming Language Standard ECMA-262
- ▷ Estructuras
 - _ Colección de pares nombre/valor (**Objeto**)
 - _ Lista ordenada de valores (**Array**)
- ▷ Sintaxis muy simple
 - _ <https://www.json.org/json-es.html>

■ JSON Schema

- ▷ <https://json-schema.org/>

■ Soporte en sistemas de bases de datos

- ▷ Representación y modelo de datos para las bases de datos NoSQL de tipo documental más conocidas (MongoDB, CouchDB, ...)
- ▷ Soporte en ISO SQL similar al proporcionado para XML

Bases de datos

Objeto-Relacionales

José R.R. Viqueira

Centro Singular de Investigación en Tecnoloxías Intelixentes (CITIUS)
Rúa de Jenaro de la Fuente Domínguez,
15782 - Santiago de Compostela.

Despacho: 209

Telf: 881816463

Mail: jrr.viqueira@usc.es

Skype: jrviqueira

URL: <https://citius.gal/team/jose-ramon-rios-viqueira>

Curso 2023/2024