

## CAPÍTULO 8.- BASES DE DATOS ORIENTADAS A OBJETOS.

### 8.1.- NUEVAS APLICACIONES DE LAS BASES DE DATOS.

- Las primeras bd se desarrollaron a partir de los sistemas de gestión de archivos. Evolucionaron hacia las bd en red o hacia las jerárquicas y posteriormente hacia las relacionales.
- **Características** comunes de estas **aplicaciones antiguas**:
  1. **Uniformidad** → gran nº de elementos de datos con estructura y tamaño similar.
  2. **Orientación a registros** → los elementos de datos consisten en registros de longitud fija.
  3. **Elementos de datos de pequeño tamaño** → cada registro es pequeño.
  4. **Campos atómicos** → no hay estructuras en el interior de los campos (cumplen la 1ª forma normal).
- Han aparecido **nuevas aplicaciones** de bd que **no cumplen alguna** de las características y para las cuales los **modelos relacionales** son **insuficientes**.

### 8.2.- EL MODELO ORIENTADO A OBJETOS.

#### 8.2.1.- ESTRUCTURA DE LOS OBJETOS.

- El paradigma orientado a objetos se basa en el **encapsulamiento** de datos y código relacionado.
- **Cada objeto** asociado con:
  1. Un conjunto de **variables** que contienen los datos.
  2. Un conjunto de **mensajes** a los que responde (única interfaz con el exterior).
  3. Un conjunto de **métodos**, cada uno es código que implementa un mensaje; el método devuelve un valor como respuesta al mensaje.
- **Mensaje** → hace referencia al **intercambio** de **solicitudes** entre los objetos.
- **“Invocar un método”** → para denotar el hecho de enviar un mensaje a un objeto y la ejecución del método correspondiente.
- **Métodos** (y mensajes, según el método que lo implementa) **clasificación**:
  1. Sólo **lectura**.
  2. De **actualización**.

#### 8.2.2.- CLASES DE OBJETOS.

- Todos los objetos de una **clase** comparten una **definición común** y se diferencian en los valores asignados a los atributos.
- Cada **objeto** es un **ejemplar** de su **clase**.
- Un **objeto clase** incluye:
  1. Una variable de tipo conjunto → conjunto de todos los objetos que son ejemplares de la clase.
  2. Un método para el mensaje **nuevo** → crea un nuevo ejemplar de la clase.

#### 8.2.3.- HERENCIA.

- La palabra clave **isa** (es una) se utiliza para indicar que una clase es una **especialización** de otra.
- Los objetos de una clase contienen (**heredan**) las **variables y métodos** definidos en sus **superclases**.
- Se representa como un árbol.
- **Posibilidad de sustitución** → Cualquier método de una clase dada puede ser llamado con un objeto perteneciente a una de sus subclases (reutilización del código).
- **Especialización parcial** → permite objetos que pertenezcan a una clase, pero que no pertenezcan a ninguna de las subclases de la misma.

### 8.2.4.- HERENCIA MÚLTIPLE.

- **Herencia múltiple** → heredar de varias superclases (grafo acíclico dirigido).
- **Ambigüedad potencial** si se puede heredar la misma variable o método de más de una superclase. Ninguna solución se ha considerado óptima y cada sistema lo trata de una forma diferente.
- Se puede utilizar **para modelar** el concepto de **papeles** → cada categoría es un papel y la herencia múltiple expresa que un objeto tiene varios papeles de modo simultáneo.

### 8.2.5.- IDENTIDAD DE LOS OBJETOS.

- Los **objetos conservan** su **identidad aunque** los **valores** de las variables o las definiciones de los métodos **cambien** total o parcialmente con el tiempo (Esto no se aplica a tuplas de bd relacionales).
- Cada objeto recibe del sistema de manera **automática** un **identificador único** en el momento en que se crea.
- Los identificadores pueden guardarse como campo de otro objeto.
- Los identificadores generados por el sistema suelen ser específicos del mismo, y si se desplazan los datos a un sistema diferente hay que traducirlos.
- Los identificadores generados por el sistema pueden resultar **redundantes si** las entidades que se moldean ya disponen de **identificadores unívocos externos** al sistema (DNI).

### 8.2.6.- CONTINENTES DE OBJETOS.

- **Objetos complejos** o **compuestos** → objetos que contienen a otros objetos.
- **Jerarquía de continentes** → cuando hay varios niveles de continentes (árbol).
- Permiten examinar los datos con diferente detalle.
- Un objeto puede estar incluido en varios objetos (grafo acíclico dirigido -GAD-)

## 8.3.- LENGUAJES ORIENTADOS A OBJETOS.

La **orientación a objetos** en b.d. puede expresarse de **2 maneras**:

1. Como **herramienta de diseño** y se **codifica** con un sistema **sin** orientación a objetos.
2. Se **incorporan en** un **lenguaje** que se utiliza para trabajar con la bd. **Lenguajes posibles**:
  - 2.1. **Sistemas relacionales orientados a objetos** → se extiende un lenguaje para el tratamiento de datos como SQL.
  - 2.2. **Lenguajes de programación persistentes** → tomar un lenguaje de programación orientado a objetos existente y extenderlo para que trabaje con bd.

## 8.4.- LENGUAJES DE PROGRAMACIÓN PERSISTENTES.

- **Datos persistentes** → siguen existiendo una vez el programa que los generó ha concluido.
- **Lenguajes de las bd** → trabajan directamente con datos persistentes.
- En los lenguajes programación los únicos datos persistentes son los ficheros.
- Manera **tradicional** de crear las **interfaces** de las **bd** con los **lenguajes** de programación → **incorporar SQL** dentro del lenguaje de programación.
- **Lenguajes de programación persistentes** → lenguajes de programación extendidos con constructoras para el tratamiento de datos persistentes.

- Distinción entre lenguajes de programación persistentes y lenguajes con SQL incorporado:

LENGUAJES DE CONSULTA INCORPORADOS	LENGUAJES DE PROGRAMACIÓN PERSISTENTES
<ul style="list-style-type: none"> <li>• <b>Tipos</b> lenguaje anfitrión <b>diferentes</b> del lenguaje para tratamiento de datos.</li> <li>• Requiere <b>cambio</b> de <b>formato explícito</b> a cargo del programador (fácil introducir errores, requiere gran cantidad de código)</li> <li>• <b>Código</b> explícito para <b>búsqueda en memoria</b>.</li> <li>• Si se <b>actualizan</b>, <b>código</b> para <b>guardar</b> en disco.</li> </ul>	<ul style="list-style-type: none"> <li>• Lenguaje consulta totalmente integrado con el lenguaje anfitrión → <b>comparten</b> sistema de <b>tipos</b>.</li> <li>• No se requieren <b>cambios</b> de <b>formato</b> (se realizan de modo <b>transparente</b>)</li> <li>• No requiere código explícito del programador para buscar en memoria ni guardar en disco.</li> </ul>

- **Inconvenientes** de los lenguajes de programación persistentes:
  1. Al ser **potentes** es más sencillo cometer **errores** que dañen las **bd**.
  2. La **complejidad dificulta** la **optimización** automática de alto nivel (como reducción E/S).
  3. **No** permiten (actualmente) las **consultas declarativas** sin que aparezcan problemas.

#### 8.4.1.- PERSISTENCIA DE LOS OBJETOS.

Maneras de hacer persistentes los objetos:

- 1.- **Persistencia por clases** → **declarar** que una **clase** es **persistente** (todos los objetos de esa clase son persistentes).
  - Más sencillo. Menos conveniente. No es flexible.
  - **En muchos sistemas** el declarar que una clase es persistente lo que **significa** es que los objetos de esa clase **pueden** hacerse persistentes.
- 2.- **Persistencia por creación** → objetos persistentes o transitorios **según** la **manera** de **crearlos**.
- 3.- **Persistencia por marcas** → Los objetos **se crean** como **transitorios y después** si un objeto tiene que persistir **se marca** de manera explícita antes de que concluya el programa.
- 4.- **Persistencia por referencia** → Uno o varios objetos (**objetos raíz**) se declaran como **persistentes** de manera **explícita**. Todos los demás objetos serán **persistentes** si y sólo **si** se hace **referencia** a ellos de manera directa o indirecta **desde** un **objeto persistente** (objeto raíz).
  - **Sencillo** hacer **persistentes estructuras complejas** con declarar persistente la raíz.
  - **SGBD** sufre **carga** de seguir las cadenas lo que puede resultar costoso.

#### 8.4.2.- LA IDENTIDAD DE LOS OBJETOS Y LOS PUNTEROS.

**Grados permanencia** de la **identidad**:

1. Dentro de los procedimientos.
2. Dentro de los programas.
3. Entre programas.
4. **Persistente** → persiste entre las ejecuciones del programa y entre las reorganizaciones estructurales de los datos (es el tipo necesario para los sistemas orientados a objetos).

**Punteros persistentes** → siguen siendo válidos después del final de la ejecución del programa y después de algunas modalidades de reorganización de los datos (conceptualmente: punteros a objetos de la bd).

En C++ persistente los identificadores de los objetos persistentes se implementan como “punteros persistentes”.

### 8.4.3.- EL ALMACENAMIENTO Y EL ACCESO A LOS OBJETOS PERSISTENTES.

- **Maneras de acceso a los objetos (persistentes) de la base de datos:**
  1. dar **nombres** a los objetos.
  2. Exponer los **identificadores o punteros persistentes** que pueden guardarse de manera externa.
  3. Guardar **colecciones** de objetos y que los programas iteren sobre ellas para hallar los objetos.
- Las colecciones de objetos pueden ser a su vez objetos de un **tipo colección**.
- **Extensiones de clases** → colección de todos los objetos pertenecientes a una clase.
  - Cuando se crea un objeto de esa clase se inserta en la extensión automáticamente.
  - Cuando se borra se elimina de la extensión.
  - Permiten operar sobre todas las tuplas/objetos de la relación (Permiten que las clases se traten como relaciones).
- La mayor parte de los SGBD orientados a objetos permiten las 3 maneras de acceso a los objetos.

### 8.5.- SISTEMAS C++ PERSISTENTES.

- Hay varias bases de datos orientadas a objetos basadas en las extensiones de C++ persistentes.
- Aunque se puede apoyar la persistencia sin modificar el lenguaje, esto presenta inconvenientes por lo que la mayor parte de implementaciones extiende la sintaxis de C++.
- ODMG → Grupo de gestión de bases de datos de objetos (Object Database Management Group).
- ODMG ha trabajado en la normalización de las extensiones de los lenguajes C++ y Smalltalk.
- La norma ODMG intenta extender C++ lo mínimo posible y proporcionar la mayor parte de la funcionalidad mediante librerías de clases.
- La extensión ODMG de C++ tiene 2 partes:
  1. El lenguaje de definición de objetos de C++ (C++ ODL, Object Definition Language) que extiende la sintaxis de definición de tipos C++.
  2. El lenguaje de manipulación de objetos de C++ (C++ OML, Object Manipulation Language).

#### 8.5.1.- EL LENGUAJE PARA LA DEFINICIÓN DE OBJETOS C++ DE ODMG.

- Los objetos persistentes pertenecen a una clase que subclasifica `Persistent_Object`.
- Hay referencias o punteros persistentes.

#### 8.5.2.- EL LENGUAJE PARA EL TRATAMIENTO DE OBJETOS C++ DE ODMG.