

CAPÍTULO 9.- BASES DE DATOS RELACIONALES ORIENTADAS A OBJETOS.

- Los **modelos de datos relacionales orientados a objetos** → extienden el modelo de datos relacional proporcionando un sistema de tipos más rico que incluye la programación orientada a objetos y añade constructoras a los lenguajes de consultas (como SQL) para trabajar con los tipos añadidos.
- Permiten que los atributos de las tuplas tengan tipos complejos.
- Intentan **conservar** el **acceso declarativo** (bases relacionales) al tiempo que extienden la capacidad de modelado.

9.1.- RELACIONES ANIDADAS.

- La **primera forma normal** (1FN) exige que todos los atributos tengan dominios atómicos (sus elementos son unidades indivisibles).
- **No** todas las aplicaciones son compatibles con 1FN. Un solo objeto puede necesitar varios registros para su representación (un libro con varios autores,...).
- Una **interfaz sencilla** necesita una **correspondencia uno a uno** entre la **noción intuitiva** de objeto del usuario y el concepto de **elemento de datos** del sistema de bd (libro).
- **Modelo relacional anidado** → extensión del modelo relacional en la que los dominios pueden ser atómicos o de relación.
- **Dominio de relación** → las tuplas de los atributos pueden ser una relación, y las relaciones pueden guardarse en otras relaciones.
- Los objetos complejos pueden representarse mediante una única tupla de las relaciones anidadas.

9.2.- LOS TIPOS COMPLEJOS Y LA PROGRAMACIÓN ORIENTADA A OBJETOS.

- Extensiones de SQL para que permita los tipos complejos, incluyendo relaciones anidadas y características orientadas a objetos.
- Borrador norma SQL-3.

9.2.1.- LOS TIPOS ESTRUCTURADOS Y LAS COLECCIONES.

- **Se permiten atributos que sean:**

1. **Tipos estructurados:** `create type nombre_tipo`
`(nombre1 tipo1,`
`nombre2 tipo2,`
`.....`
`nombreN tipoN)`

2. **Colecciones:**

- **Conjuntos.**(nombre_atributo **setof**(tipo_base))
- **Arrays.**(nombre_atributo tipo_base[numero_elementos])
- **Multiconjuntos** → colecciones no ordenadas en las que un elemento puede figurar varias veces.(nombre_atributo **multiset**(tipo_base))

- Permiten que los atributos compuestos y los multivalorados de los diagramas E-R se representen directamente.
- Los **tipos** se registran en el esquema **guardado** en la **bd**, por lo que las instrucciones que tengan acceso a la bd podrán hacer uso de las definiciones de los tipos. (En los lenguajes de programación persistentes no suelen guardarse en las bd y sólo pueden verlas los programas que incluyen un archivo de texto que contenga las definiciones)

9.2.2.- LA HERENCIA. (under)

1.- Herencia de los tipos.

```
Create type nombre_subtipo
(.....)
under nombre_supertipo1, ... , nombre_supertipoN
```

- Admite herencia múltiple pero debe utilizarse con precaución para que no hayan conflictos por herencia repetida,....

2.- Herencia de las tablas.

- Si una entidad debe tener un solo tipo, habría que crear un subtipo para cada combinación posible.
- En bd se puede permitir que un objeto tenga varios tipos utilizando la herencia en el nivel de tablas y permitiendo que las entidades estén en más de una tabla simultáneamente.
- **Requisitos de consistencia:**
 - 1.- Cada tupla de la supertabla puede corresponder como máximo a una tupla de cada una de las subtablas.
 - 2.- Cada tupla en las subtablas debe tener exactamente una tupla correspondiente en la supertabla.
- Los atributos heredados diferentes de la clave primaria de la supertabla no hace falta guardarlos.
- Se puede utilizar herencia múltiple.
- Hace más natural la definición de los esquemas (sino hay que vincular de modo explícito las tablas a las supertablas con la clave primaria y definir las ligaduras para asegurar las ligaduras referenciales y de cardinalidad).
- Permite usar funciones definidas para los supertipos con objetos pertenecientes a los subtipos.

9.2.3.- TIPO REFERENCIA.

- El atributo de un tipo puede ser una referencia a un objeto de un tipo especificado (ref(Persona)).
- Se puede hacer **referencia a las tuplas** utilizando:
 - 1.- la **clave primaria**.
 - 2.- un **identificador**.
- Las subtablas heredan de manera implícita el atributo del identificador de las tuplas como cualquier otro.

9.3.- CONSULTAS CON TIPOS COMPLEJOS.

- Extensión de SQL para trabajar con tipos complejos.
- Notación con punto para atributos compuestos (atributo.subatributo) (SQL-3 quizá dos puntos ..)

9.3.1.- ATRIBUTOS DE RELACIÓN.

- Una expresión de relación puede aparecer en cualquier lugar en el que pueda aparecer el nombre de una relación.
- Hace posible aprovechar la estructura de las relaciones anidadas.

9.3.2.- EXPRESIONES DE CAMINO.

- Notación de punto con las referencias → oculta operaciones reunión entre distintas tablas → **expresión de camino**.
- Los atributos en una expresión de camino pueden ser una colección.
- Simplifican las consultas.

9.3.3.- ANIDAMIENTO Y DESANIDAMIENTO.

- **Desanidamiento** → transformar una relación anidada en 1FN.
- **Anidamiento** → transformar una relación 1FN en una relación anidada.

9.3.4.- FUNCIONES.

- Permiten que los usuarios definan **funciones en**:
 - 1.- **SQL extendido**.
 - 2.- **un lenguaje de programación**.
- Con un **lenguaje** de programación pueden ser **más eficientes** y pueden realizar **cálculos** que **no** son posibles con **SQL**.
- Las definidas con un lenguaje de programación y compiladas externamente al SGBD deben cargarse y ejecutarse con el código del SBD lo que conlleva **riesgo** de que un fallo del programa pueda **deteriorar** las estructuras internas de la **bd**.
- **Diferencias** entre las **funciones** definidas **con** un **lenguaje** de **programación** y el **SQL incorporado** al lenguaje de programación:
 - En **SQL incorporado** el programa pasa la consulta al SGBD para que la ejecute. Los resultados se devuelven al programa. El **código** escrito por el **usuario no** necesita nunca tener **acceso** a la propia **bd**.
 - En las **funciones codificadas por el usuario**, el propio SBD tiene que ejecutar ese código, lo que puede generar **vulnerabilidad** en términos de integridad y de seguridad; o hay que copiar en un espacio de datos separado los datos sobre los que opera la función con la consiguiente **sobrecarga**.

9.4.- CREACIÓN DE VALORES Y DE OBJETOS COMPLEJOS.

- Los objetos compuestos se crean **dando** los valores de sus **elementos componentes**.
- Las funciones **constructoras** (crear un objeto del tipo T es T();;) devuelven un objeto nuevo del tipo sin inicializar (rellena el campo identificador del objeto oid y devuelve el objeto).

9.5.- COMPARACIÓN ENTRE LAS B.D. ORIENTADAS A OBJETOS Y LAS B.D. RELACIONALES ORIENTADAS A OBJETOS.

	SISTEMAS RELACIONALES	BD ORIENTADAS OBJ. BASADAS EN LENGUAJES PERSISTENTES	SISTEMAS RELACIONALES ORIENTADOS A OBJETOS
Tipos datos	sencillos	complejos	complejos
Lenguajes de consulta	potentes	no suelen ser potentes	Potentes
Protección	elevada	Menor (más potente más errores)	Elevada
Consultas declarativas	Si (menor rendimiento)	No (mayor rendimiento aplicaciones en memoria y gran nº accesos a bd)	Si (menor rendimiento)
Optimización de alto nivel (E/S disco)	Fácil	Difícil	fácil
		Integración lenguaje programación (no requiere traducción datos, lo que da mayor rendimiento)	Simplificación modelos y consultas (mediante datos complejos)
		bd de CAD	

Hay que tener en cuenta que algunos sistemas de bd no respetan estas fronteras.