Unidad VII: Bases de Datos Orientadas a objetos

7.1 Visión general

En una base de datos orientada a objetos, la información se representa mediante objetos como los presentes en la programación orientada a objetos. Cuando se integra las características de una base de datos con las de un lenguaje de programación orientado a objetos, el resultado es un sistema gestor de base de datos orientada a objetos (ODBMS, object database management system). Un ODBMS hace que los objetos de la base de datos aparezcan como objetos de un lenguaje de programación en uno o más lenguajes de programación a los que dé soporte. Un ODBMS extiende los lenguajes con datos persistentes de forma transparente, control de concurrencia, recuperación de datos, consultas asociativas y otras capacidades.

Las bases de datos orientadas a objetos se diseñan para trabajar bien en conjunción con lenguajes de programación orientados a objetos como Java, C#, Visual Basic.NET y C++. Los ODBMS usan exactamente el mismo modelo que estos lenguajes de programación.

Los ODBMS son una buena elección para aquellos sistemas que necesitan un buen rendimiento en la manipulación de tipos de dato complejos.

7.2 Tipos de datos complejos

Estos son los tipos de datos que soporta el SQL. Los sinónimos son palabras equivalentes al tipo de dato indicado. El tamaño indica cuánto ocupará una columna del tipo indicado.

Tipo de dato	Sinónimos	Tamaño	Descripción
BINARY	VARBINARY BINARY VARYING BIT VARYING	1 byte por carácter	Se puede almacenar cualquier tipo de datos en un campo de este tipo. Los datos no se traducen (por ejemplo, a texto). La forma en que se introducen los datos en un campo binario indica cómo aparecerán al mostrarlos.
BIT	BOOLEAN LOGICAL LOGICAL1 YESNO	1 byte	Valores Sí y No, y campos que contienen solamente uno de dos valores.

TINYINT	INTEGER1 BYTE	1 byte	Un número entero entre 0 y 255.
COUNTER	AUTOINCREMENT		Se utiliza para campos contadores cuyo valor se incrementa automáticamente al crear un nuevo registro.
MONEY	CURRENCY	8 bytes	Un número entero comprendido entre – 922.337.203.685.477,5808 y 922.337.203.685.477,5807.
DATETIME	DATE TIME	8 bytes	Una valor de fecha u hora entre los años 100 y 9999
UNIQUEIDENTIFIER	GUID	128 bits	Un número de identificación único utilizado con llamadas a procedimientos remotos.
DECIMAL	NUMERIC DEC	17 bytes	Un tipo de datos numérico exacto con valores comprendidos entre 1028 - 1 y -1028 - 1. Puede definir la precisión (1 - 28) y la escala (0 - precisión definida). La precisión y la escala predeterminadas son 18 y 0, respectivamente.
REAL	SINGLE FLOAT4 IEEESINGLE	4 bytes	Un valor de coma flotante de precisión simple con un intervalo comprendido entre – 3,402823E38 y – 1,401298E-45 para valores negativos, y desde 1,401298E-45 a 3,402823E38 para valores positivos, y 0.
FLOAT	DOUBLE FLOAT8 IEEEDOUBLE NUMBER	8 bytes	Un valor de coma flotante de precisión doble con un intervalo comprendido entre – 1,79769313486232E308 y – 4,94065645841247E-324 para valores negativos, y desde 4,94065645841247E-324 a 1,79769313486232E308 para valores positivos, y 0.
SMALLINT	SHORT INTEGER2	2 bytes	Un entero corto entre – 32.768 y 32.767.
INTEGER	LONG INT INTEGER4	4 bytes	Un entero largo entre – 2.147.483.648 y 2.147.483.647.
IMAGE	LONGBINARY GENERAL OLEOBJECT	Lo que se requiera	Desde cero hasta un máximo de 2.14 gigabytes. Se utiliza para objetos OLE.
TEXT	LONGTEXT LONGCHAR MEMO NOTE NTEXT	2 bytes por carácter. (Consulte las notas).	Desde cero hasta un máximo de 2.14 gigabytes.
CHAR	TEXT(n) ALPHANUMERIC CHARACTER STRING	2 bytes por carácter. (Consulte	Desde cero a 255 caracteres.

	VARCHAR CHARACTER VARYING NCHAR NATIONAL CHARACTER NATIONAL CHAR NATIONAL CHARACTER VARYING NATIONAL CHAR VARYING	las notas).	

7.3 Tipos estructurados y herencia en SQL

En SQL estos tipos se denominan tipos definidos por el usuario. Las especificaciones **final** indica que no se puede crear subtipos de *nombre*, mientras que la especificación **not final** de dirección indica que se pueden crear subtipos de *dirección*. Ahora se pueden usar estos tipos para crear atributos compuestos en las relaciones, con sólo declarar que un atributo es de uno de estos tipos. Por ejemplo, se puede crear una tabla *cliente* de la siguiente manera.

create table *cliente* (
nombre Nombre,
direccion Direccion,
fecha_nacimiento date)

O bien, realizando una estructura más del tipo Cliente y generar la tabla a partir de ella:

create type TipoCliente as
(*nombre* **Nombre**,
direccion **Direccion**,
fecha_nacimiento **date**)
not final

create table *cliente* **of** *TipoCliente*

Se puede tener acceso a los componentes de los atributos compuestos usando la notación “punto”; por ejemplo, *nombre.nombre_pila* devuelve el componente nombre de pila del atributo nombre. El acceso al atributo *nombre* devolvería un valor del tipo estructurado **Nombre**.
La siguiente consulta ilustra la manera de tener acceso a los atributos componentes de los atributos compuestos. La consulta busca el apellido y la ciudad de cada cliente.

select nombre.apellido, direccion.ciudad
from cliente

7.4 Herencia de tablas

La herencia puede hallarse en el nivel de los tipos o en el nivel de las tablas. En primer lugar se considerara la herencia de los tipos y después en el nivel de las tablas.

- Herencia de tipos: Los tipos derivados heredan los atributos de superclase. Los métodos también se heredan por sus subtipos, al igual que los atributos. Sin embargo, un subtipo puede redefinir el efecto de un método declarándolo de nuevo. Esto se conoce como sobre escritura (overriding) del método.

- Herencia de tablas: Cada tabla almacena la clave primaria (que se puede heredar de una tabla padre) y los atributos definidos localmente. Los atributos heredados (aparte de la clave primaria) no hace falta guardarlos y pueden obtenerse mediante una reunión con la supertabla basada en la clave primaria.

Cada tabla almacena todos los atributos heredados y definidos localmente. Cuando se inserta una tupla se almacena solo en la subtabla en la que se inserta y su presencia se infiere en cada supertabla. El acceso a todos los atributos de una tupla es más rápido, dado que no se requiere una reunión.

7.5 Tipos de arreglo multiconjunto en SQL

SQL soporta dos tipos de conjuntos: arrays y multiconjuntos. Multiconjuntos es un conjunto no ordenado, en el que cada elemento puede aparecer varias veces. A diferencia de los elementos de los multiconjuntos, los elementos de los arrays están ordenados Por ejemplo se ilustra la manera en que se pueden definir en SQL, estos atributos valorados con arrays y multiconjuntos. create type Editor as (nombre varchar(20) sucursal vachar(20) create type Libro as (titulo vachar (20), array_autores vachar (20) array[10], fecha_publicacion date, editor Editor, conjunto_palabras_clave varchar(20) multiset) create table Los atributos multivalorados de los esquemas E-R se pueden asignar en SQL atributos valorados como multiconjuntos si el orden es importante se pueden usar los arrays de SQL en lugar de los multiconjuntos. • Creación y acceso a los valores de los conjuntos array[´silberschatz´,´Korth´, ´Sudarshan´] de manera parecida se puede crear un multiconjunto de palabras clave de la manera siguiente: multiset[´computadora´, ´base de datos´, ´SQL] Consulta de los atributos valorados como conjuntos´ Ahora se considerara la forma de manejar los atributos que se valoran como conjuntos. Las expresiones que se valoran como conjuntos pueden aparecer en cualquier parte en la que pueda aparecer el nombre de una relación, como las clausulas FROM. Al desanidar un arrays la consulta anterior pierde información sobre el orden de los elementos de consulta. Se pueden usar las clausulas UNESTED WITH ORDINALITY para obtener esta información. La clausulas WITH ORDINALITY genera un atributo adicional que se registra la posición del elemento en el arrays. Se puede usra una consulta parecida. Pero sin la clausula WITH ORDINALITY, para generar la relación. Anidamiento y desanidamiento La transformación de una relación anidada en una forma con menos atributos de tipos relación (o sin ellos) se denomina desanidamiento.

7.6 Identidad de los objetos y tipos de referencia en SQL

Los lenguajes orientados a objetos ofrecen la posibilidad de hacer referencia a objetos. Los atributos de un tipo dado pueden servir de referencia para los objetos de un tipo concreto. Por ejemplo, en SQL se puede definir el tipo Departamento con el campo nombre y el campo director,

que es una referencia al tipo Persona, y la tabla departamentos del tipo Departamento, de la manera siguiente:

```
create type Departamento(
nombre varchar(20),
director ref(Persona) scope personas)

create table departamentos of Departamento
```

En este caso, la referencia está restringida a las tuplas de la tabla personas. La restricción del ámbito de referencia a las tuplas de una tabla es obligatoria en SQL, y hace que las referencias se comporten como las claves externas.

La tabla a la que hace referencia debe tener un atributo que guarde el identificador para cada tupla. Ese atributo, denominado atributo autorreferenciable (self-referential attribute), se declara añadiendo una cláusula ref is a la instrucción create table:

```
create table personas of Persona
ref is id_persona system generated
```

7.7 Implementacion de las caracteristicas OR

Los sistemas de bases de datos relacionales orientadas a objetos son básicamente extensiones de los sistemas de bases de datos relacionales ya existentes. Las modificaciones resultan claramente necesarias en muchos niveles del sistema de base de datos.

Las interfaces de programas de aplicación como ODBC y JDBC se han extendido para recuperar y almacenar tipos estructurados; por ejemplo, JDBC ofrece el método getObject() que devuelve un objeto Java Struct, a partir del cual se pueden extraer los componentes del tipo estructurado. También es posible asociar clases de Java con tipos estructurados de SQL, y JDCB puede realizar conversiones entre los tipos.

La orientación a objetos constituye una nueva forma de pensar acerca de problemas empleando modelos que se han organizado tomando como base conceptos del mundo real.

Los modelos orientados a objetos son útiles para comprender problemas, comunicarse con expertos en esa aplicación, modelar empresas, preparar documentación y diseñar programas y bases de datos.