TEMA-10-ACCESO-A-DATOS.pdf

user_4383848

Acceso a datos

1º Desarrollo de Aplicaciones Multiplataforma

Estudios España

TEMA 10 - ACCESO A DATOS
BBDD DE DATOS OBJETO RELACIONALES

1. DEFINICIÓN DE BD OBJETO RELACIONALES:

Una base de datos objeto-relacional es una combinación de una base de datos orientada a objetos y un modelo de base de datos relacional. Significa que soporta objetos, clases, herencia, etc. que tendríamos en los modelos orientados a objetos y también tiene soporte para tipos de datos o estructuras tabulares, del modelo de datos relacional.
Uno de los mayores objetivos de los modelos de base de datos objeto-relacionales es acortar distancias entre las bases de datos relacionales y las prácticas orientadas a objetos realizadas frecuentemente en diferentes lenguajes como, Java.
Podríamos definir la base de datos objeto-relacional como aquella base de datos relacional que evoluciona desde dicho modelo hacia a algunas de las características del modelo de objetos, haciéndola una base de datos híbrida. Uno de los gestores de bases de datos más conocidos hoy en día es Oracle.
Este implementa el modelo de objeto como una extensión del modelo relacional.
El resultado es un modelo relacional de objetos que ofrece la intuición y la economía de una interfaz de objetos, al mismo tiempo que conserva su alta concurrencia y el rendimiento de una relacional.

2. CARACTERÍSTICAS DE LAS BASES DE DATOS OBJETO RELACIONALES:

Una característica de este tipo de base de datos es que podremos crear nuevos tipos de datos, los cuales permitirán gestionar aplicaciones específicas con mucha riqueza de dominios. Estos nuevos tipos de datos pueden ser tipos compuestos, lo que nos lleva a pensar que se podrán definir, al menos, dos métodos:
• Uno para **convertir de este tipo a caracteres ASCII**
• Y otro que haga esta función a la inversa, **desde caracteres ASCII hasta nuevos tipos de datos**.
Se soportarán distintos tipos complejos como, por ejemplo:
• Registros
• Listas
• Referencias
• Pilas
• Colas
• Arrays
Podremos crear también funciones que tengan código en diferentes lenguajes. Otro de los aspectos que caracterizan a las bases de datos objeto-relacionales son:
• Dispondremos de una **mayor capacidad de expresión** para definir conceptos y diferentes asociaciones.
• Podremos **crear también operadores asignando nombre y existencia** de aquellas consultas más complejas.
• En los tipos de registro, estilo relacional, **podremos usar encadenamiento y herencia**.
• Podremos hacer uso de la reusabilidad, **compartiendo bibliotecas de clases definidas previamente**.
• Posibilidad de introducir **comprobación de reglas de integridad por medio de triggers**.

3. TABLAS Y TIPOS DE OBJETOS: DEFINICIÓN:

Cuando se crea un tipo de dato, realmente estamos definiendo cierto comportamiento para una agrupación de datos de nuestra aplicación. Para los tipos de objetos, **usaremos object type** y, para los tipos de colecciones, **collection type**.
Un objeto representa una entidad en el mundo real y se compone de:
• **Nombre:** Con el que identificaremos el tipo de objeto
• **Atributos:** Con los que definiremos la estructura. Los atributos pueden ser de tipo creado por el usuario o básico del propio sistema.
• **Métodos:** Que pueden ser funciones o procedimientos. Los encontraremos escritos en código PL/SQL cuando están almacenados en la propia base de datos y en el lenguaje C cuando se almacenan externamente.
La creación de un método en Oracle se realiza junto a la creación de su tipología y debe llevar siempre el tipo de compilación como, PRAGMA RESTRICT_REFERENCES; de esta forma, evitamos la manipulación de los diferentes datos o de las distintas variables PL/SQL.

4. TABLAS Y TIPOS DE OBJETOS: EXPLOTACIÓN:

Podemos tener distintos objetivos para esos nuevos datos: podemos usarlos para definir nuevos tipos, para almacenarlos en tablas de ese tipo de datos o para definir los distintos atributos de una tabla.
Sabemos que una tabla de objetos es una clase específica de tabla que almacenará un objeto por cada fila y que, al mismo tiempo, facilita el ingreso de los atributos del mismo objeto, como si se tratara de columnas de la tabla.
La diferencia entre la primera y la segunda tabla es que la primera almacena objetos con su propio ID y la segunda, realmente, no es una tabla de objetos, sino una tabla con una columna cuyo tipo de dato es un objeto.
La segunda tabla posee una columna con un tipo de datos complejo y sin identidad de objeto. Es una de las ventajas que ofrece Oracle. Aparte, Oracle nos permite definir como tabla:
• Una columna con tipología de objeto.
• Aquella que tiene el mismo número de columnas como atributos que almacena.

5. TIPOS DE COLECCIÓN: ARRAY:

Para poder establecer relaciones **«uno a muchos» (1:N)**. Una colección está formada por un número no definido de elementos y **todos ellos deben ser del mismo tipo**. De esta forma, podemos guardar en un simple atributo un conjunto de datos en forma de array (Varray) o, también, tendríamos la opción de la tabla anidada.

EL VARRAY

Podríamos definir un array como una serie de elementos ordenados que son del mismo tipo. Estos elementos llevan asociado un índice que nos sirve para saber su posición dentro del array. Oracle permite que el tipo **VARRAY sea un tipo de dato variable,** pero sí se debe establecer el máximo de elementos una vez se declara dicho tipo.
Utilizaremos el VARRAY para:
• Definir la tipología de una columna de una tabla relacional.
• Definir la tipología de un atributo de un tipo objeto.
• Definir una variable del lenguaje PL/SQL.

Una vez declaramos este objeto VARRAY, no se reserva realmente ninguna cantidad de espacio. Si el espacio está disponible, se almacena igual que el resto de columnas, pero, si por el contrario, es superior a 4.000 bytes, se almacenará en una tabla aparte, como un dato de tipo BLOB.

6. TIPOS DE COLECCION: TABLAS ANIDADAS:

Una tabla anidada es una **lista de elementos no ordenados que mantienen una misma tipología**. El máximo no está especificado en la definición de la tabla, y el orden de los elementos no se mantiene.
Realizaremos SELECT, INSERT, DELETE y UPDATE de la misma forma que lo hacemos con las tablas comunes, usando la expresión TABLE. Una tabla anidada puede ser vista o interpretada como una única columna. Si la columna en una tabla anidada es un tipo de objeto de usuario, la tabla puede ser vista como una tabla multicolumna, con una columna por cada atributo del objeto usuario que fue definido.
Una vez definido el tipo, **podremos usarlo para**:
• El tipo de datos de una tabla relacional
• Un atributo de un objeto de tipo usuario
• Una variable PL/SQL, un parámetro o una función que devuelva un tipo.

7. REFERENCIAS:

La base de datos objeto-relacional permite que los identificadores únicos que se les asigna a los objetos de una tabla puedan ser referenciados desde los atributos de otros objetos distintos o desde la columna de una tabla.
Hablamos del **tipo denominado REF**, cuyo atributo guardará una referencia (un enlace) a un objeto de la tipología definida y genera una relación entre ambos objetos.
Este tipo de referencias se usarán para acceder a los objetos relacionados y actualizarlos, pero no es posible realizar operaciones directamente sobre las referencias. Para usar una referencia o actualizarla, usaremos **REF o NULL**.
Una vez hemos definido una columna de tipo REF, se puede acotar el alcance a los objetos que se guarden en una determinada tabla.

8. HERENCIA DE TIPOS:

La herencia de tipos nos permite crear jerarquías de tipos.
Una jerarquía de tipos es una **serie de niveles sucesivos de subtipos, cada vez más especializados, que derivan de un tipo de objeto ancestro común, denominado supertipo**. Esto no es un concepto nuevo, ya que, en programación orientada a objetos, lo usamos muy frecuentemente, sobre todo en lenguaje Java.
Los subtipos derivados heredarán las características del tipo de objeto principal y pueden ampliar la definición de este.
Los tipos especializados pueden añadir nuevos atributos o métodos, o redefinir métodos heredados de la clase tipo padre. La jerarquía del tipo resultante facilita un nivel superior de abstracción para manejar la complejidad de un modelo de una aplicación