Egibide

Unidad Didáctica 10:

BASE DE DATOS OBJETOS RELACIONALES

1. Introducción

Limitaciones del modelo relacional

Las bases de datos objeto relacionales surgen como respuesta a una serie de limitaciones del modelo relacional:

- Tipos de datos predefinidos
- Tipos de datos son atributos mono valuados en dominios escalares
- Operaciones SQL predefinidas
- Lenguaje SQL computacionalmente limitado: Necesidad de lenguajes externos como Java, .NET etc.
- Problema de adaptación a OO: **Impedancia**

Definición

Base de datos que ha evolucionado desde el modelo relacional hasta una base de datos híbrida que contiene tanto tecnología relacional como de orientación a objetos. Las bases de datos objeto relacionales son compatibles con las bases de datos relacionales: no es necesario reescribirlas, ni adaptar sus aplicaciones. Tienen capacidad para crear consultas *ad hoc(*personalizar una consulta en tiempo real *).* Soporta: identidad de objetos, encapsulación, herencia, polimorfismo, objetos complejos e interrelaciones.

2. Novedades del modelo objeto relacional.

La programación a objetos se ha integrado en el mundo de las bases de datos relacionales principalmente con la evolución de los datos mono valuados a datos representados con objetos que pueden tener sus propios atributos y comportamiento.
Nuevos conceptos:

- Dominios:
  - Actúan como tipos de datos
  - Permiten almacenar múltiples valores en una columna de una misma fila
  - en cada columna de cada fila solo podrá contener un objeto
  - Almacenar procedimientos propios de su clase

- Tablas de Objetos
  - Cada fila es un objeto
  - Con las restricciones del modelo relacionales
  - Con las operaciones de concatenación del modelo relacional

3. Tipos del modelo objeto relacional

3.1 Tipos predefinidos

- Carácter:CHAR,VARCHAR2,NCHAR,NVARCHAR2, LONG…
- Numéricos: NUMBER, INTEGER, FLOAT…
- DATE
- ROWID
- LOB(CLOB, BLOB,..)

3.2 Tipos definidos por el usuario

- Tipos estructurados
- Tipos objeto
- Tipos REF(referencia)
- Tipos colección

4. Tipos estructurados

Son aquellos con los que podemos describir tipos de datos complejos para los que se pueden definir varios atributos propios: *dirección, apellidos…*

```
CREATE OR REPLACE TYPE direccion
AS OBJECT (
        calle VARCHAR2(30),
        ciudad VARCHAR2(20),
        provincia VARCHAR2(2),
        codigoPostal VARCHAR2(5));
```

```
CREATE OR REPLACE TABLE persona(
nombre VARCHAR2(30),
vive_en direccion,
foto BLOB);
```

5. Tipos Objeto

Representa a una entidad del mundo real y se compone de:
• Nombre
• Atributos: pueden ser de un tipo de dato básico o de un tipo creado por el usuario.
• Métodos: La especificación de un método se hace junto a la creación de su tipo, y debe llevar siempre asociada una directiva de compilación (PRAGMA RESTRICT_REFERENCES), para evitar que los métodos manipulen la base de datos o las variables del paquete PL/SQL. Tienen el siguiente significado:

- WNDS: no se permite al método modificar las tablas de la base de datos
- WNPS: no se permite al método modificar las variables del paquete PL/SQL
- RNDS: no se permite al método leer las tablas de la base de datos
- RNPS: no se permite al método leer las variables del paquete PL/SQL

Los métodos se pueden ejecutar sobre los objetos de su mismo tipo.

Podemos representar **entidades o estructuras individuales** que tienen ciertos atributos y/o métodos propios: *empleado, estudiante, departamento…*
Constan de 2 partes:

- Especificación: declaración de atributos. Especificación de métodos.

```
CREATE TYPE nmbre_obj AS OBJECT(
Atributos tipos_dato,
….,
[MEMBER FUNCTION nombre metodo RETURN tipo dato])
```

```
CREATE OR REPLACE TYPE Cliente
AS OBJECT(
        DNI NUMBER,
        Nombre VARCHAR2(30),
        FechaNac DATE,
        MEMBER FUNCTION EDAD RETURN NUMBER,
        PRAGMA RESTRICT_REFERENCES (EDAD, WNDS,WNPS,RNPS, RNDS))
```

- Cuerpo:Implementación privada

```
CREATE TYPE BODY Cliente AS
MEMBER FUNCTION EDAD RETURN NUMBER IS
        a NUMBER;
        d DATE;
        BEGIN
                d:= today();
                a:= d.año – FechaNac.año;
                IF (d.mes < FechaNac.mes) OR
                        ((d.mes = FechaNac.mes) AND (d.dia < FechaNac.dia))
                        THEN a:=a-1;
                ENDIF;
        RETURN a;
        END;
END;
```

Crear una tabla utilizando este objeto:

CREATE OR REPLACE TABLE Tabla_cli OF Cliente;

Atributos

- Al menos tiene que especificarse un atributo para que la especificación del tipo de objeto sea correcto.
- Se declaran mediante un nombre y un tipo.
- Cada atributo debe tener un nombre único dentro del tipo.
- Los atributos pueden ser de cualquier tipo de Oracle, excepto:
  - Tipos Oracle: LONG,LONG RAW, NCHAR,NCLOB,NVARCHAR2,MLSLABEL y ROWID
  - Tipos PL/SQL: BINARY_INTEGER,BOOLEAN,PLS_INTEGER,RECORD,REF,CURSOR, %TYPE y %ROWTYPE
  - Tipos definidos en paquetes PL/SQL
- No se pueden definir con la clausula DEFAULT ni la restricción NOT_NULL.
- El tipo de un atributo puede ser otro tipo de objeto.

Métodos

- Es un subprograma declarado en la especificación mediante la palabra MEMBER.
- Se deben definir después de los atributos.
- Funciones y procedimientos.
- Nombre del método + lista opcional de parámetros + tipo de retorno(Funciones)
- La cabecera del cuerpo del método debe coincidir con la especificación.
- Se puede hacer referencia a atributos del propio tipo de objeto y a otros métodos.
- Pueden existir tipos de objeto para el que no se definan métodos. En este caso el cuerpo no es necesario.

6. Tipo REF.

Los identificadores únicos asignados por Oracle a los objetos que se almacenan en una tabla, permiten que éstos puedan ser referenciados desde los atributos de otros objetos o desde las columnas de tablas. El tipo de datos proporcionado por Oracle para soportar esta facilidad se denomina **REF**. Un atributo de tipo **REF** almacena una referencia a un objeto del tipo definido, e implementa una relación de asociación entre los dos tipos de objetos. Estas referencias se pueden utilizar para acceder a los objetos referenciados y para modificarlos; sin embargo, no es posible operar sobre ellas directamente. Para asignar o actualizar una referencia se debe utilizar siempre **REF** o **NULL**.
Cuando se define una columna de un tipo a **REF**, es posible restringir su dominio a los objetos que se almacenen en cierta tabla. Si la referencia no se asocia a una tabla sino que sólo se restringe a un tipo de objeto, se podrá actualizar a una referencia a un objeto del tipo adecuado con independencia de la tabla donde se almacene. En este caso su almacenamiento requerirá más espacio y su acceso será menos eficiente. El siguiente ejemplo define un atributo de tipo **REF** y restringe su dominio a los objetos de cierta tabla.

```
CREATE TABLE clientes_tab OF cliente_t;


CREATE TYPE ordenes_t AS OBJECT (
        ordnum NUMBER,
        cliente REF clientes_t,
        fechpedido DATE,
        direcentrega direccion_t);

CREATE TABLE ordenes_tab OF ordenes_t (
        PRIMARY KEY (ordnum),
        SCOPE FOR (cliente) IS clientes_tab);
```

Cuando se borran objetos de la BD, puede ocurrir que otros objetos que referencien a los borrados queden en estado inconsistente. Estas referencias se denominan *dangling references*, y Oracle proporciona el predicado llamado **IS DANGLING** que permite comprobar cuándo sucede esto.

7. Tipos Colección

Para poder implementar relaciones **1:N**, Oracle permite definir tipos colección. Un dato de tipo colección está formado por un número indefinido de elementos, todos del mismo tipo. De esta manera, es posible almacenar en un atributo un conjunto de tuplas en forma de array (**VARRAY**), o en forma de tabla anidada.
Al igual que los tipo objeto, los tipo colección también tienen por defecto unas funciones constructoras de colecciones cuyo nombre coincide con el del tipo. Los argumentos de entrada de estas funciones son el conjunto de elementos que forman la colección separados por comas y entre paréntesis, y el resultado es un valor del tipo colección.
En Oracle es posible diferenciar entre un valor nulo y una colección vacía. Para construir una colección sin elementos se puede utilizar la función constructora del tipo seguida por dos paréntesis sin elementos dentro.

7.1 El tipo VARRAY

Son vectores SQL que permiten almacenar listas de objetos de una longitud máxima pero variable. No se pueden hacer consultas sobre ellos.

Las siguientes declaraciones crean un tipo para una lista de precios, y un valor para dicho tipo.

```
CREATE TYPE precios AS VARRAY(10) OF NUMBER(12);

precios('35', '342', '3970');
```

Se puede utilizar el tipo **VARRAY** para:
• Definir el tipo de dato de una columna de una tabla relacional.
• Definir el tipo de dato de un atributo de un tipo de objeto.
• Para definir una variable PL/SQL, un parámetro, o el tipo que devuelve una función.

Cuando se declara un tipo **VARRAY** no se produce ninguna reserva de espacio. Si el espacio que requiere lo permite, se almacena junto con el resto de columnas de su tabla, pero si es demasiado largo (más de 4000 bytes) se almacena aparte de la tabla como un **BLOB**.

En el siguiente ejemplo, se define un tipo de datos para almacenar una lista de teléfonos. Este tipo se utiliza después para asignárselo a un atributo del tipo de objeto **empleado**.

CREATE OR REPLACE TYPE telefonos **AS** VARRAY(5) **OF VARCHAR**(10);

CREATE TABLE EMPLEADO
(DNI NUMBER,
Nombre VARCHAR2(30),
Telefonos_contacto **telefonos**);

INSERT INTO EMPLEADO VALUES ('82828282','Ramón',
telefonos ('945111111', '945222222','945333333');

7.2 Tablas anidadas

Sirven para representar en la intersección de una fila y una columna mediante una tabla.

Gráficamente se podría representar de la siguiente forma:

id_orden	fecha	detalles
11	Mayo 13 de 2016	id_producto: 1, cantidad: 100; id_producto: 2, cantidad: 90
34	Mayo 2 de 2016	Vacía
78	Abril 23 de 2016	id_producto: 1, cantidad: 150

Primero se debe definir el *tipo* de la tabla anidada que desea crear.

El tipo de datos de la tabla anidada puede estar basado en un tipo de datos:

- Primitivo
- Definido por el usuario (típicamente)
- Incluso en el de otra tabla anidada (tablas anidadas de tablas anidadas etc.)

Siguiendo con el ejemplo anterior:

```
CREATE OR REPLACE TYPE detalle_tip
AS OBJECT(
 id_producto NUMBER(3),
 cantidad NUMBER(10));
```

Se crea el tipo de la tabla anidada basada en el tipo detalle_tip :

```
CREATE OR REPLACE TYPE
nest_detalle AS TABLE OF detalle_tip;
```

Ahora ya es posible declarar la columna detalles de tipo **nest_detalle** (tabla anidada de detalles):

```
CREATE TABLE orden (

id_orden NUMBER(3) PRIMARY KEY,
 fecha DATE NOT NULL,
 detalles nest_detalle)
NESTED TABLE detalles STORE AS store_detalles;
```

Donde:

detalles es el nombre de la columna y contiene para cada orden su tabla anidada de detalles.

store_detalles es el nombre físico del lugar (tabla) donde se almacenan **todas** las tablas anidadas de la columna detalles.

Inserción de datos:

```
INSERT INTO orden VALUES(11,'Mayo 13 de 2016',
        nest_detalle( detalle_tip(1,100),
                 detalle_tip(2,90))
                );
```

8. Instrucciones útiles

Modificar la especificación de un tipo

```
ALTER TYPE NOMBRETIPO ADD ATTRIBUTE NUEVO VARCHAR1;
ALTER TYPE NOMBRETIPO DROP ATTRIBUTE VIEJO;
```

Forzar compilación para detección de errores

```
ALTER TYPE NOMBRETIPO COMPILE;
SELECT * FROM USER_OBJECTS WHERE STATUS = 'INVALID';
SELECT * FROM USER_ERRORS;
```

Borrar un tipo o su cuerpo

```
DROP TYPE NOMBRETIPO FORCE;
DROP TYPE BODY NOMBRETIPO;
```