ACCESO A DATOS

TÉCNICO EN DESARROLLO DE APLICACIONES MULTIPLATAFORMA

Bases de datos objeto relacionales

Tablas y Tipos de objetos

VARRAY

Tabla anidada

Definición de base de datos 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. Esto 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, entre otras cosas, 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 practicas orientadas a objetos realizadas frecuentemente en diferentes lenguajes como C++, Java, C#, etc.

Con esta información que se ha adelantado de base, 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**.

Muchos **lenguajes**, tal y como hemos comentado anteriormente, han sabido adaptarse con **extensiones y frameworks** para poder **trabajar** con estas nuevas **bases de datos objeto relacionales de Oracle**.

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**.

Características de las bases de datos objeto relacionales

Una de las principales características 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**.

Con dicha tipología de datos, podremos crear también **funciones** que tengan código en **diferentes lenguajes**, como **SQL, Java, C#**, etc.

Características de las bases de datos objeto-relacionales:

- 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**.

Triggers

*Trigger también llamado **disparador**, es simplemente un **script** de código que puede estar **escrito en diferentes lenguajes**.*

*Consiste básicamente en **ejecutar** una serie de **procedimientos**, **según** ciertas **instrucciones**, **cuando se realicen** determinadas **operaciones** en la información de la base de datos.*

Definición de tablas y tipos de objetos

Cuando se crea un **tipo de dato**, realmente estamos definiendo cierto **comportamiento** para una **agrupación de datos** de nuestra aplicación. En **Oracle**, con la base de datos objeto-relacional, tendremos la opción de **definir nuestros propios tipos** de datos. Para los tipos de objetos, usaremos object type y, para los tipos de colecciones, collection type. Para **construir** dichos **tipos** de usuario, deberemos **usar los básicos** que poseemos en el sistema.

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**.

Diremos que 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, por ejemplo:

```
PRAGMA RESTRICT_REFERENCES;
```

De esta forma: **evitamos la manipulación** de los diferentes **datos** o de las distintas **variables PL/SQL**.

Ejemplo de código de creación de un tipo de dato nuevo (address_t) en el lenguaje, establecido por la base de datos Oracle:

```
CREATE TYPE address_t AS OBJECT (
        street VARCHAR2(200),
        city VARCHAR2(200),
        prov CHAR(2),
        poscode VARCHAR2(20)
);
```

Como podemos, observar en la creación de la tabla:

- Indicaremos el **nombre del tipo a definir** “address_t”.
- **Establece**remos cuatro **atributos** diferentes que **definen la estructura** creada:
    - **Street**: Es un tipo **texto** VARCHAR2, con **200 caracteres máximo**.
    - **City**: Otro VARCHAR2, también con 200 de extensión.
    - **Prov**: Válido para introducir **2 caracteres máximo**.
    - **Poscode**: Un VARCHAR2 de 20 caracteres máximo.

Nulos

Vamos a explicar la cláusula nula (null) en los objetos de tipo usuario. Cuando creamos un objeto de tipo, tendrá “x” atributos, y por lo tanto el objeto nunca será atómicamente nulo, es decir que no será nulo completamente. En el ejemplo a continuación se crea un objeto llamado person_typ, y se definen sus atributos. Vamos a usar este objeto person_typ: creamos una tabla llamada contact, la cual tiene un atributo tipo person_typ. Al hacer un INSERT incluimos un objeto person_type, el cual tiene algunos atributos nulos (idno NUMBER, name VARCHAR y phone VARCHAR). Pero como ya hemos dicho, un objeto de tipo no es atómicamente nulo, y sus atributos pueden ser inicializados a nulo como en este caso, o reemplazados mas tarde. Esta es una característica de los objetos tipo usuario.

```
CREATE OR REPLACE TYPE person_typ AS OBJECT (
        idno 1 NUMBER,
        name VARCHAR2 (30),
        phone VARCHAR2 (20),
        MAP MEMBER FUNCTION get_idno RETURN NUMBER,
        MEMBER PROCEDURE display details (SELF IN OUT NOCOPY person_typ ) );
/
CREATE OR REPLACE TYPE BODY person_typ AS
        MAP MEMBER FUNCTION get_idno RETURN NUMBER IS
        BEGIN
                RETURN idno;
        END;
        MEMBER PROCEDURE display_details (SELF IN OUT NOCOPY person_typ ) IS
        BEGIN
                —— use the PUT_LINE procedure of the DBMS_OUTPUT package to display
                details
                DBMS_OUTPUT.PUT_LINE(TO_CHAR(idno) || ‘ - ‘ || mame || ‘ - ‘ || phone);
        END;
END;
/
CREATE TABLE contacts (
        contact person_typ,
        contact_date DATE );

INSERT INTO contacts VALUES (
        person_typ (NULL, NULL, NULL), '24 Jun 2003' );

INSERT INTO contacts VALUES (
        NULL, '24 Jun 2003’ );
```

“OBJECT TYPES”

Un tipo de objeto es un tipo de dato que puede utilizarse de manera similar a tipos de datos estándar, como NUMBER o VARCHAR2 en Oracle Database. Puede especificarse como el tipo de datos de una columna en una tabla relacional y declarar variables de ese tipo. Una instancia de un tipo de objeto se denomina objeto.
Los tipos de objetos actúan como planos o plantillas que definen tanto la estructura como el comportamiento. Son objetos de esquema de base de datos y están sujetos al mismo control administrativo que otros objetos de esquema. El código de aplicación puede recuperar y manipular instancias de estos objetos.
Se utiliza la instrucción SQL CREATE TYPE para definir tipos de objetos. Para crear un tipo de objeto llamado person_typ podríamos seguir el siguiente ejemplo:

```
CREATE TYPE person_typ AS OBJECT (
        idno
        NUMBER,
        first_name
        VARCHAR2(20),
        last_name
        VARCHAR2(25),
        email
        VARCHAR2(25),
        phone
        VARCHAR2(20),
        MAP MEMBER FUNCTION get_idno RETURN NUMBER,
        MEMBER PROCEDURE display_details ( SELF IN OUT NOCOPY person_typ ));
/
CREATE TYPE BODY person_typ AS
        MAP MEMBER FUNCTION get_idno RETURN NUMBER IS
        BEGIN
                RETURN idno;
        END;
        MEMBER PROCEDURE display_details ( SELF IN OUT NOCOPY person_typ ) IS
        BEGIN
                -- utilizar PUT_LINE del paquete DBMS_OUTPUT para mostrar detalles
                DBMS_OUTPUT.PUT_LINE(TO_CHAR(idno) || ' ' || first_name || ' ' || last_name);
                DBMS_OUTPUT.PUT_LINE(email || ' ' || phone);
        END;
END; /
```

El símbolo / que vemos en el código anterior sirve para indicar al motor de base de datos que ese bloque de código ya está listo y puede ser ejecutado.

Ejemplo de creación de nuevos tipos de objetos

Vamos a crear estos dos objetos:

- **Objeto Vehículo** con los atributos: número ruedas, peso, largo.
- **Objeto Coche** con los atributos: tipo, marca.

Hay que tener en cuenta que el objeto Coche es un **tipo de Vehículo**.

En Oracle, con la base de datos objeto-relacional, podremos agregar nuestros propios tipos de objetos a la base de datos.

Definiremos los 2 objetos que se requieren. Habría que reflexionar sobre **qué tipología** tienen dichos objetos.

En primer lugar, definiríamos el objeto vehículo teniendo en cuenta que, tanto para el número de ruedas, como el peso, y el largo, son variables del tipo básico del sistema NUMBER:

Definición de objeto vehículo

```
CREATE TYPE vehiculo_t AS OBJECT (
        nRuedas NUMBER,
        peso NUMBER,
        largo NUMBER);
```

En la siguiente definición, habría que caer en la cuenta de que **conlleva herencia**, por lo tanto, la **tipología** de este **coche es vehículo**, precisamente el mismo objeto que definimos previamente.

Por último, la marca nos valdría con un Varchar de 50 caracteres.

Definición objeto coche

```
CREATE TYPE coche_t AS OBJECT (
        tipo vehiculo_t,
        marca VARCHAR2(50) );
```

Explotación de tablas y tipos de objetos

Una vez se ha realizado la definición de tipos, podemos tener **distintos objetivos** para esos nuevos datos. Podemos usarlos:

- para definir **nuevos tipos**,
- para **almacenarlos en tablas** de ese tipo de datos,
- 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. Por ejemplo, podríamos tener una tabla de vehículos del año actual y otra para guardar vehículos de años anteriores:

Tablas Oracle

Tabla que **almacena objetos** con su **propio ID**:

```
CREATE TABLE vehiculos_año_tab OF vehiculo_t (
        numVehiculo PRIMARY KEY);
```

No es una tabla de objetos, sino una tabla con una columna cuyo tipo de dato es un objeto.

Posee una columna con un tipo de datos complejo y sin identidad de objeto. Es una de las ventajas que ofrece Oracle:

```
CREATE TABLE vehiculos_antiguos_tab (
        anio NUMBER, vehiculo vehiculo_t);
```

Aparte, **Oracle** nos permite **definir como tabla**:

• Una **columna** con tipología de **objeto**. (Según el documento original: una tabla con una sola columna cuyo tipo es el de un tipo de objetos).

• Aquella que tiene el **mismo número de columnas** como **atributos** que almacena. (O mejor dicho desde el documento original: una tabla que tiene **tantas columnas como atributos los objetos que almacena**).

Tipos de colección: array

Para poder establecer relaciones «uno a muchos» (1:N), Oracle nos permite definir **colecciones**. 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

Como bien sabemos, 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. Podemos ver algún ejemplo:

VARRAY

```
CREATE TYPE numeros AS VARRAY(10)
    OF NUMBER(10);
numeros ('6', '18', '75870');
```

Tal y como se observa en el código de arriba, hemos definido un nuevo tipo «números» como un VARRAY con máximo 10 elementos y con posiciones cuyo tipo serán NUMBER máximo 10 cifras.

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**.

BLOB

BLOB viene de las iniciales **Binary Large Object**. Es un lugar en la base de datos donde se va a almacenar **información binaria** de este tipo de array que añadamos. Existen los BLOB y los **CLOB** (**Character Large Object**), que son objetos grandes que almacenan **cadenas de caracteres**.

Tipos de colección: tablas anidadas

Los tipos de colección en Oracle objeto-relacional son:

- **VARRAYS**,
- **Tablas anidadas**.

Tablas anidadas

Tenemos la posibilidad de **anidar objetos con herencia** en nuestra base de datos objeto-relacional.

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.

Sintaxis para crear una tabla anidada:

```
CREATE TYPE nombre_tipo AS TABLE OF tabla_tipo;
```

Con este código, estaremos creando una **tabla anidada tabla_tipo**, la cual contendrá **objetos de tipo de usuario nombre_tipo**.

La definición que hemos visto justo arriba **no asignará espacio**. 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.

Ejemplo de tabla anidada

```
CREATE TYPE nested_table_type AS TABLE OF VARCHAR2(50);
```

Aquí, se crea un tipo llamado nested_table_type, que es una tabla anidada de tip VARCHAR2 con una longitud máxima de 50 caracteres.

Referencias

La base de datos objeto-relacional Oracle 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**. A continuación, veremos, en el siguiente Código, un atributo de tipo REF que **restringe su dominio a una determinada tabla**.

Referencias

```
CREATE TABLE clientes_tab OF clientes_t;

CREATE TYPE ordenes_t AS OBJECT (
        ordennum NUMBER,
        cliente REF clientes_t,
        fechapedido DATE,
        direntrega direccion_t);

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

Tal y como podemos observar, al inicio se **crea una tabla** de tipo clientes_t. A continuación, se crea un **tipo de objeto** llamado ordenes_t con 4 atributos. El segundo de ellos será el **objeto que será referenciado**.

Por último, tenemos la **creación de la tabla ordenes_tab**, donde define una primary key y el segundo dato será, precisamente, esa **referencia** que hemos definido en el objeto anterior. De modo que:

> *El **atributo referenciado** se construye agregando a continuación "**REF**" y seguidamente la **restricción** a objetos de cierta **tabla**.*
>
> *Para **referenciar el atributo**, se usa: "**SCOPE FOR**" seguido del **atributo referenciado entre paréntesis** y a continuación, fuera de los paréntesis, la **tabla** en la que tiene **restringido el dominio** el atributo referenciado.*
>
> *Para ganar consistencia en la BBDD, utilizamos la **cláusula SCOPE IS** para indicar que sí o sí tiene que existir el curso al que hace referencia desde la tabla estudiante:*

```
CREATE TABLE Estudiantes OF TipoEstudiante (
        PRIMARY KEY (id),
        curso REF TipoCurso SCOPE IS Cursos
);
CREATE TABLE Cursos OF TipoCurso (PRIMARY KEY (id));
```

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, como se puede observar, 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.

A continuación, mostraremos un pequeño esquema donde podremos ver la herencia entre objetos de tipo usuario.

Podremos observar cómo partimos de una clase principal e iremos heredando atributos, al mismo tiempo que los hijos podrán ir aportando nuevas características a los de los objetos padre:

Esquema herencia

Podemos observar que el tipo «Persona» poseerá una serie de atributos que serán heredados tanto por el tipo «Estudiante» como por el tipo «Empleado».

«Estudiante» y «Empleado», a su vez, agregarán, en caminos diferentes, nuevos atributos para los tipos hijos que se puedan crear. En este caso, un hijo de «Estudiante» es «EstudianteTiempoParcial» que, finalmente, heredará atributos de su padre «Estudiante» y del padre de su padre, «Persona».

Ejemplo de Herencia de tipos

Teniendo en cuenta los siguientes tipos:

Vehículo_t, coche_t, ciclomotor_t, cocheCarreras_t, cochePaseo_t, motocicleta_t, ciclomotorPaseo_t.

Situamos el tipo padre arriba.

La relación de tipos padre a hijos sería la siguiente:

Esquema herencia de tipos

"**Vehículo**"

es el padre,

del que heredan:

"**Ciclomotor**" "**Motocicleta**" y "**Coche**"

"**CiclomotorPaseo**" hereda de "Ciclomotor"

y

"**CocheCarreras**" y "**CochePaseo**" heredan de "Coche"

Ejemplo de creación de una base de datos objeto relacional, un objeto y una tabla.

El objeto será de tipo **usuario**, tendrá el nombre **vehículo**, y dispondrá de 2 **atributos**:

- Nombre
- Marca

Marca tendrá una **referencia directa** al objeto en sí, y nombre será de tipo STRING.

La tabla tendría que ser creada de **objetos de tipo vehículo**.

Se requiere de la creación de, básicamente, dos elementos:

- Creación de un **objeto** nuevo de tipo usuario,
- Creación de una **tabla**.

Creación del objeto

```
CREATE TYPE vehiculo_typ AS OBJECT (
    nombre VARCHAR2(30),
    marca REF vehiculo_typ);
```

En este código, podemos ver cómo creamos un **nuevo type**, que se denominará vehiculo_typ y que tendrá **2 atributos**, como indica el ejercicio: nombre (Varchar) y marca (será una referencia al propio objeto).

Definición de la tabla

```
CREATE TABLE vehiculo_tabla OF vehiculo_typ;
```

De esta forma, estaremos definiendo una tabla cuyas filas serán objetos del tipo anteriormente definido.

https://docs.oracle.com/