1. BASES DE DATOS OBJETO-RELACIONALES 2
1.1. CARACTERÍSTICAS 2
1.2. SISTEMAS GESTORES 3
2. ORACLE DATABASE 4
2.1. TIPOS DE OBJETOS 4
2.2. MÉTODOS 4
2.3. TABLAS Y COLECCIONES 5
2.4. REFERENCIAS 7

1. BASES DE DATOS OBJETO-RELACIONALES

Las **Bases de Datos Objeto-Relacionales (BDOR)** son una extensión de las bases de datos relacionales tradicionales a las que se les ha añadido conceptos del modelo orientado a objetos.

Un Sistema Gestor de Bases de Datos Objeto-Relacionales (SGBD-OR) es un sistema gestor de bases de datos que contiene características del modelo relacional y del orientado a objetos, es decir, es un sistema relacional que permite almacenar objetos en tablas.

1.1. CARACTERÍSTICAS

El modelo de datos relacional orientado a objetos extiende el modelo de datos relacional con un **sistema de tipos más rico** (que incluye tipos de datos más complejos) y con la **programación orientada a objetos**. Para trabajar con estos nuevos tipos de datos, los lenguajes de consulta relacionales como SQL también necesitan ser extendidos.

Las extensiones orientadas a objetos que comúnmente se encuentran en las bases de datos objeto-relacionales son: objetos de datos de gran tamaño, tipos de datos estructurados o abstractos, tipos de datos definidos por el usuario, vectores, secuencias y conjuntos, tablas en tablas y procedimientos almacenados.

Las **características** más importantes de los SGBD-OR son:

- Soporte de tipos de datos básicos y complejos. El usuario puede crear sus propios tipos de datos.
- Soporte para crear métodos para esos tipos de datos. Se pueden crear funciones miembro usando los tipos de datos definidos por el usuario.
- Gestión de tipos de datos complejos con un esfuerzo mínimo.
- Herencia.
- Posible almacenamiento de múltiples valores en una columna de una misma fila.
- Relaciones o tablas anidadas.
- Su principal inconveniente es que aumenta la complejidad del sistema y provoca un aumento del coste asociado.

Las razones por las cuales los SGBD-OR son la opción más adecuada actualmente son:

- Las bases de datos objeto-relacionales son compatibles con las bases de datos relacionales tradicionales. Es decir, una aplicación que use una base de datos relacional se puede utilizar con el nuevo modelo orientado a objetos sin tener que reescribirla.
- Los desarrolladores están muy familiarizados con los sistemas relacionales (MySQL, Oracle, Informix, MS-SQL-Server, PostgreSQL) y pueden pasar sus aplicaciones actuales sobre bases de datos relacionales al modelo de datos relacional extendido. Posteriormente, se pueden adaptar estas aplicaciones y bases de datos para que utilicen las funciones orientadas a objetos.

Un SGBD-OR se diferencia básicamente de un SGBD-R en que define el tipo de datos **objeto**. El usuario puede crear sus propios tipos de datos a partir de tipos de datos predefinidos u otros tipos ya creados por el usuario. Además, los SGBD-OR permiten crear métodos (o funciones miembro) para esos tipos de datos definidos, proporcionando flexibilidad y seguridad.

Por tanto, un tipo de dato objeto define una estructura y un comportamiento común para el conjunto de datos de una aplicación. Los usuarios pueden definir tipos de datos mediante dos mecanismos de construcción:

1) **Tipos de Objetos.** Representa una entidad del mundo real, proporciona las características básicas de la orientación a objetos (abstracción, encapsulación, herencia de tipos, sobrecarga de operadores y ligadura dinámica) y se compone de los siguientes elementos:
   - *Nombre.* Sirve para identificar el tipo de los objetos.
   - *Atributos.* Modelan la estructura y los valores de los datos de ese tipo. Cada atributo puede ser de un tipo de datos básico o de un tipo definido por el usuario.
   - *Métodos.* Especifican el comportamiento. Son procedimientos y funciones escritos en un lenguaje de programación soportado por los SGBD-OR que se utilicen. En el caso de usar Oracle, se podrían codificar en PL/SQL o Java.
2) **Colecciones.** Los SGBD-OR permiten el almacenamiento de colecciones de elementos en una única columna:
   - *Vectores.* Permiten almacenar un conjunto de elementos, todos del mismo tipo, y cada elemento tiene un índice numérico asociado.
   - *Tablas Anidadas.* Permiten almacenar una tabla en una columna de otra tabla. Para que la tabla anidada se pueda contener en una columna, el tipo de esa columna debe ser un tipo de objeto existente en la base de datos.

Las características de orientación a objetos están especificadas en el **estándar SQL:1999**, que introduce los tipos de datos estructurados definidos por el usuario. Estos tipos estructurados permiten la herencia simple y se definen mediante sentencias SQL `CREATE TYPE`.

1.2. SISTEMAS GESTORES

Existe una oferta variada de **Sistemas Gestores de Bases de Datos Objeto-Relacionales (SGBD-OR)** en el mercado. Como en el caso de los sistemas gestores orientados a objetos, existen:

- Sistemas privativos:
  1) Oracle Database: https://www.oracle.com/database/index.html
  2) Informix: http://www.ibm.com/software/products/es/informix-family
- Sistemas bajo licencias de software libre:
  1) PostgreSQL: https://www.postgresql.org/
  2) CUBRID: http://www.cubrid.org/
  3) OpenLink Virtuoso: https://virtuoso.openlinksw.com/

2. ORACLE DATABASE

Oracle Database es un sistema gestor de bases de datos de tipo objeto-relacional, desarrollado por Oracle Corporation. Es uno de los SGBD más completos con las siguientes características: soporte de transacciones, multiplataforma, escalabilidad y estabilidad.

Su dominio en el mercado de servidores empresariales ha sido casi total hasta ahora. Recientemente tiene la competencia de Microsoft SQL Server y otros SGBD-R con licencia libre (PostgreSQL, MySQL o Firebird).

2.1. TIPOS DE OBJETOS

Para crear un **tipo de objeto** se utiliza la sentencia `CREATE TYPE`. Los siguientes ejemplos PL/SQL muestran la definición de los tipos de objetos *direccion* y *persona*:

```
CREATE OR REPLACE TYPE direccion AS OBJECT (
    calle VARCHAR2(25),
    ciudad VARCHAR2(20),
    codigo_postal NUMBER(5)
);

CREATE OR REPLACE TYPE persona AS OBJECT (
    codigo NUMBER,
    nombre VARCHAR2(35),
    direc direccion,
    fecha_nac DATE
);
```

Una vez creado, se puede usar para declarar e inicializar objetos como si se tratase de cualquier otro tipo predefinido.

Para borrar un tipo de objeto se utiliza la sentencia `DROP TYPE`. Los siguientes ejemplos PL/SQL muestran el borrado de los tipos de objetos *direccion* y *persona*:

```
DROP TYPE persona;
DROP TYPE direccion;
```

2.2. MÉTODOS

Los **métodos** son funciones o procedimientos que se pueden declarar en la definición de un tipo de objeto para implementar el comportamiento deseado para dicho tipo de objeto.

En Oracle, los métodos se escriben en PL/SQL o en Java (como procedimientos almacenados). Existen varios tipos de métodos:

- **MEMBER.** Es un método que sirve para actuar con los objetos. Puede ser funciones o procedimientos.
- **STATIC.** Es un método estático independiente de las instancias del objeto. Puede ser funciones o procedimientos.
- **CONSTRUCTOR.** Sirve para inicializar el objeto. Se trata de una función cuyos argumentos son los valores de los atributos del objeto y que devuelve el objeto inicializado.

Para cada objeto existe un constructor predefinido por Oracle. No obstante, se pueden crear y/o sobreescribir otros constructores adicionales, incluyendo, por ejemplo, valores por defecto y restricciones.

Los siguientes ejemplos PL/SQL muestran la definición de los tipos de objetos *direccion* y *rectangulo*, incluyendo diversos métodos:

```
CREATE OR REPLACE TYPE direccion AS OBJECT (
    calle VARCHAR2(25),
    ciudad VARCHAR2(20),
    codigo_postal NUMBER(5),
    MEMBER PROCEDURE set_calle(calle VARCHAR2),
    MEMBER FUNCTION get_calle RETURN VARCHAR2
);

CREATE OR REPLACE TYPE rectangulo AS OBJECT (
    base NUMBER,
    altura NUMBER,
    area NUMBER,
    CONSTRUCTOR FUNCTION rectangulo(base NUMBER, altura NUMBER)
        RETURN SELF AS RESULT
);
```

Una vez creado un tipo de objeto con la especificación de sus métodos, su implementación se escribe mediante la sentencia `CREATE TYPE BODY`. Los siguientes ejemplos PL/SQL muestran la implementación de los métodos de los tipos de objetos *direccion* y *rectangulo*:

```
CREATE OR REPLACE TYPE BODY direccion AS
    MEMBER PROCEDURE set_calle(calle VARCHAR2) IS
    BEGIN
        SELF.calle := calle;
    END;
    MEMBER FUNCTION get_calle RETURN VARCHAR2 IS
    BEGIN
        RETURN calle;
    END;
END;

CREATE OR REPLACE TYPE BODY rectangulo AS
    CONSTRUCTOR FUNCTION rectangulo(base NUMBER, altura NUMBER)
        RETURN SELF AS RESULT
    AS
    BEGIN
        SELF.base := base;
        SELF.altura := altura;
        SELF.area := base * altura;
        RETURN;
    END;
END;
```

Para borrar la implementación de un tipo de objeto se utiliza la sentencia `DROP TYPE BODY`.

2.3. TABLAS Y COLECCIONES

Tras la definición de objetos simples o complejos, éstos se pueden utilizar para definir nuevos tipos, para definir columnas de tablas de ese tipo o para definir tablas que almacenen objetos. Una **tabla de objetos** es una tabla que almacena un objeto en cada fila y el acceso a los atributos de dichos objetos se realiza como si se tratasen de columnas de la tabla.

El siguiente ejemplo PL/SQL crea la tabla *alumnos* de tipo *persona*, con la columna *codigo* como clave primaria:

```
CREATE TABLE alumnos OF persona (
    codigo PRIMARY KEY
);
```

Los siguientes ejemplos PL/SQL muestran la inserción de datos en la tabla *alumnos* y la consulta del nombre y de la calle de los alumnos que viven en Zaragoza:

```
INSERT INTO alumnos (codigo, nombre, direc, fecha_nac)
VALUES (
    1, 'Juan Pérez',
    direccion ('Av. Navarra 141', 'Zaragoza', 50017),
    '26/10/1985'
);

SELECT nombre, a.direc.get_calle()
FROM alumnos a
WHERE a.direc.ciudad = 'Zaragoza';
```

Las bases de datos objeto-relacionales pueden permitir el almacenamiento de colecciones de elementos en una única columna. En Oracle:

- Un **VARRAY** permite almacenar un conjunto de elementos, todos del mismo tipo, y con un índice asociado para cada elemento.
- Una **tabla anidada** permite almacenar una tabla en una columna de otra tabla.

Para crear una **colección** de elementos se usa la sentencia `CREATE TYPE` con `VARRAY`. En la declaración de un tipo `VARRAY` no se produce ninguna reserva de espacio. En las consultas no se pueden poner condiciones sobre los elementos almacenados dentro del `VARRAY`, puesto que todos los valores del `VARRAY` son accedidos y recuperados como bloque, no de forma individual.

El siguiente ejemplo PL/SQL muestra el uso de VARRAY:

```
CREATE TYPE vector_telefonos AS VARRAY(3) OF VARCHAR2(9);

CREATE TABLE contactos (
    id NUMBER(2),
    nombre VARCHAR2(25),
    telef vector_telefonos
);

INSERT INTO contactos
VALUES (
    1, 'Manuel',
    vector_telefonos ('976001122', '656998877', '639012345')
);

UPDATE contactos
SET telef = vector_telefonos ('976555555', '660777333')
WHERE nombre = 'Manuel';
```

Una **tabla anidada**, que está formada por un conjunto de elementos (todos del mismo tipo), está contenida en una columna de otra tabla. El tipo de esa columna debe se un tipo de objeto existente en la base de datos.

Para crear una tabla anidada se usa la sentencia `CREATE TYPE`, habiendo definido antes el tipo de la columna que contendrá la tabla anidada.

El siguiente ejemplo PL/SQL muestra el uso de tablas anidadas:

```
CREATE TYPE tabla_direcciones AS TABLE OF direccion;

CREATE TABLE contactos (
    id NUMBER(2),
    nombre VARCHAR2(25),
    direc tabla_direcciones
)
NESTED TABLE direc
STORE AS direc_anidada;
```

```
INSERT INTO contactos
VALUES (
    1, 'Elena',
    tabla_direcciones (
        direccion ('Av. Paris 25', 'Toledo', 45005),
        direccion ('C/. Segovia 23', 'Guadalajara', 19004),
        direccion ('C/. Los Manantiales 10', 'Cáceres', 10005)
    )
);
```

La cláusula `NESTED TABLE` identifica el nombre de la columna que contendrá la tabla anidada. La cláusula `STORE AS` especifica el nombre de la tabla (*direc_anidada*) en la que se van a almacenar las direcciones que se representan en el atributo *direc* de cualquier objeto de la tabla *contactos*.

2.4. REFERENCIAS

Las relaciones entre objetos se establecen mediante el operador `REF` asociado a un atributo. Un atributo de este tipo almacena una **referencia** al objeto del tipo definido e implementa una relación de asociación entre los dos tipos de objetos. Una columna de tipo `REF` guarda un puntero a una fila de otra tabla y contiene el OID (identificador del objeto) de dicha fila.

El siguiente ejemplo PL/SQL muestra el uso de referencias:

```
CREATE TYPE empleado AS OBJECT (
    nombre VARCHAR2(30),
    jefe REF empleado
);

CREATE TABLE empleados OF empleado;

INSERT INTO empleados
VALUES (empleado ('Gil', NULL));

INSERT INTO empleados
SELECT empleado ('Arroyo', REF(e))
FROM empleados e
WHERE e.nombre = 'Gil'
```

En Oracle, las relaciones pueden estar restringidas:

- Cuando se restringe mediante **SCOPE**, todos los valores almacenados en la columna REF apuntan a objetos de la tabla indicada. Sin embargo, puede ocurrir que haya valores que apunten a objetos que no existan.
- La restricción mediante **REFERENTIAL** obliga a que las referencias sean siempre a objetos que existen en la tabla referenciada.

Dos funciones muy utilizadas en las consultas SQL son:

- **ref(objeto).** Devuelve el OID de un objeto.
- **deref(referencia).** Obtiene los datos del objeto al que apunta la referencia.

Los siguientes ejemplos PL/SQL muestran el uso de estas funciones en varias consultas:

```
-- obtener el identificador del objeto cuyo nombre es 'Gil'
SELECT REF(e)
FROM empleados e
WHERE nombre = 'Gil';

-- obtener el nombre del empleado y los datos del jefe
-- de cada empleado
SELECT nombre, DEREF(e.jefe)
FROM empleados e;
```