4.1 Bases de Datos orientadas a objetos

4.1.1 Características

4.1.2 Arquitecturas

4.1.3 Análisis de ventajas y desventajas

4.1.4 Aplicaciones

Paradigma Orientado a Objetos

- Un paradigma es un enfoque o manera de visualizar la realidad.
- Este enfoque permite ver a los sistemas como grupos cooperativos de objetos cada uno de los cuales representa una instancia de alguna clase.

Modelo de objetos

En el paradigma de la **orientación a objetos**, un sistema se concibe como un conjunto de objetos que se comunican entre sí mediante mensajes.

A nivel conceptual, un **objeto** es una entidad percibida por el analista; mientras que a nivel de implementación, un objeto corresponde con el encapsulamiento de un conjunto de operaciones que pueden ser invocadas, y de un estado que recuerda el efecto de las operaciones.

Los objetos se comunican entre sí mediante el **paso de mensajes**, por lo que se requiere que los objetos presenten su **interfaz** (firma del estado y comportamiento) a otros objetos.

Características del paradigma OO

Existen varias características que son contempladas en este paradigma:

- **Abstracción:** Es una descripción simplificada o especificación de un objeto que enfatiza algunos de los detalles o propiedades del mismo, mientras suprime otros.

-Encapsulamiento: Es el proceso de ocultar todos los detalles de un objeto que no contribuyen a sus características esenciales.

-Polimorfismo: En ocasiones una operación tiene el mismo nombre en diferentes clases, pero puede realizar diferentes acciones, en este caso cada clase sabe como realizar dicha acción.

-Herencia: Una superclase tiene una relación de herencia cuando una subclase hereda todos los atributos y métodos de la clase que proviene, y los suyos propios.

Objetos

Un objeto se describe por sus propiedades, también llamadas **atributos**, y por los servicios que puede proporcionar -su **comportamiento**-.

El **estado** de un objeto viene determinado por los valores que toman sus atributos, y que cumplen las restricciones impuestas.

Se denomina **clase** a la conceptualización de la estructura y características de un objeto, sirviendo como "plantilla" para la **instanciación** de estos objetos.

Todo objeto tiene una característica que lo identifica unívocamente, llamada el identificador del objeto **OID**, y que generalmente es asignado por el sistema de administración de datos.

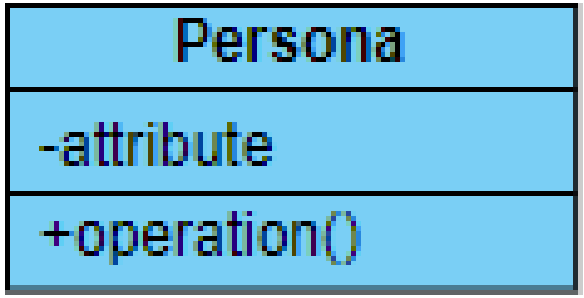
Diagrama de clases

Un diagrama de clases o estructura estática muestra el conjunto de clases y objetos importantes que forman parte de un sistema, junto con las relaciones existentes entre clases y objetos.

Muestra de una manera estática la estructura de información del sistema y la visibilidad que tiene cada una de las clases, dada por sus relaciones con los demás en el modelo.

Diagrama de clases

- **Clase:** representa un conjunto de entidades que tienen en común propiedades, operaciones, relaciones y semántica.
- Una clase es un constructor que define la estructura y comportamiento de una colección de objetos denominados instancias de la clase.
- En UML la clase está representada por un rectángulo con tres divisiones internas, y es el elemento fundamental del diagrama.



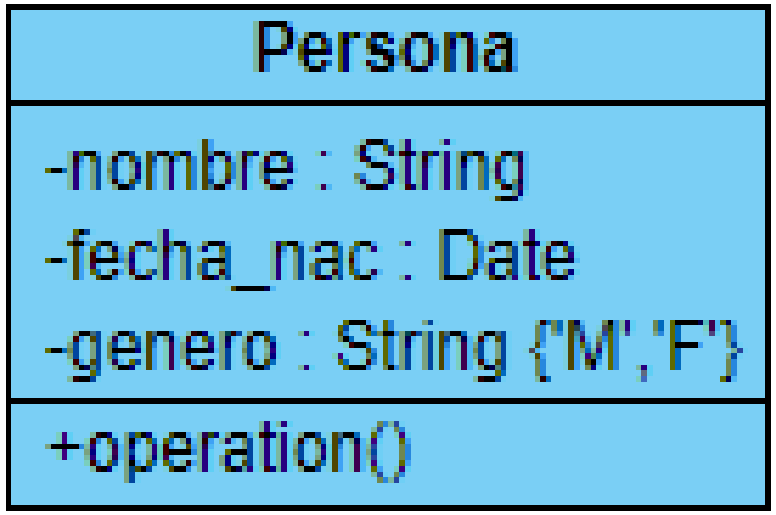


Diagrama de clases

- **Atributo:** Representa una propiedad de una clase. Cada atributo de un objeto tiene un valor que pertenece a un dominio de valores determinado por el tipo de datos.
- Las sintaxis de una atributo es:

  Visibilidad nombre : tipo { propiedades}
- Donde visibilidad es uno de los siguientes indicadores:
  - **+** público.
  - **#** protegido.
  - **-** privado.

Diagrama de clases

- **Operación:** El conjunto de operaciones (métodos) que describen el comportamiento de los objetos de una clase. La sintaxis de una operación en UML es:
  - *Visibilidad nombre (lista de parámetros): tipo que retorna { propiedades}*

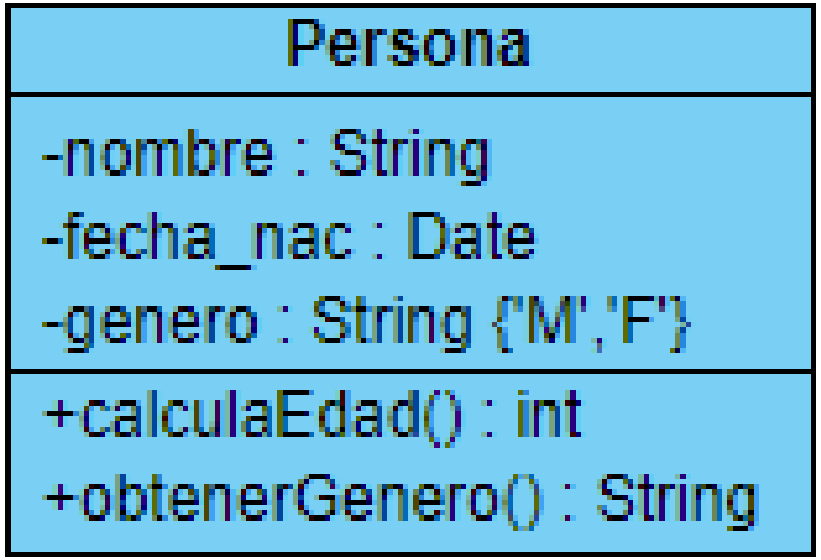
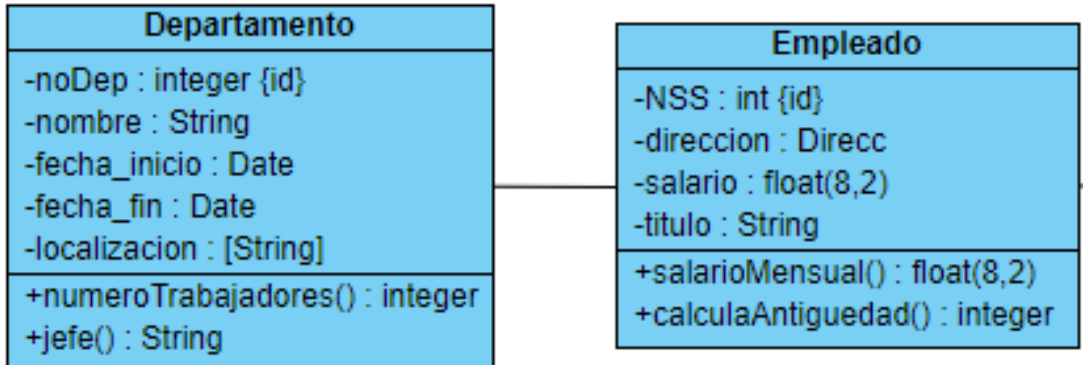


Diagrama de clases

- **Objeto:** es una instancia de una clase. Se caracteriza por tener una identidad única, un estado definido por un conjunto de valores de atributos y un comportamiento representado por sus operaciones y métodos.
- **Asociación:** Una asociación es una abstracción de la relación existente en los enlaces entre los objetos. Una asociación es una línea que une dos o más clases.

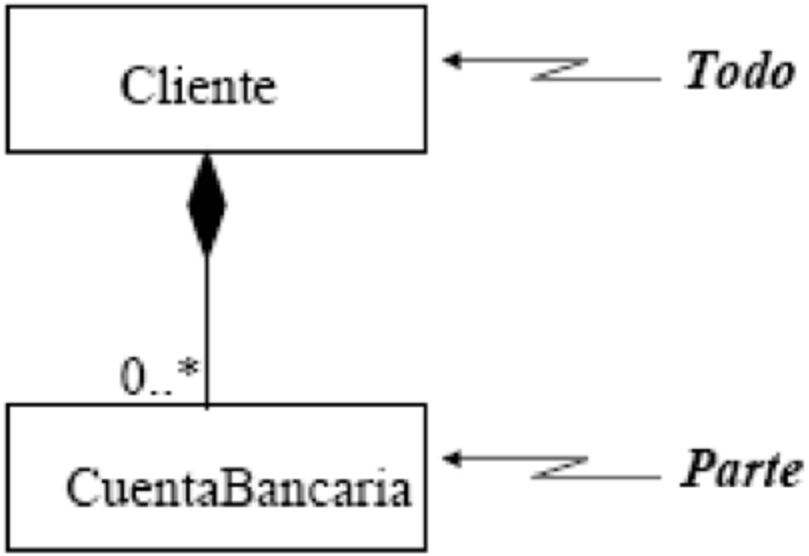


Relaciones entre objetos

- En un sistema basado en objetos, se clasifican las relaciones entre ellos de la siguiente forma:
- **Asociación**. Las relaciones de asociación representan un conjunto de enlaces entre objetos o instancias de clases. Es el tipo de relación más general, y denota básicamente una dependencia semántica.
  - **Generalización (Herencia)**.- aquí se organizan las clases en jerarquías de supertipos/subtipos, que permiten la herencia, en donde las características y comportamientos de una superclase se transmiten a las subclases de las que descienden. Se identifica como tipo “es-un”.
  - **Agregación**.- habilita relaciones entre clases en forma de composición, en las que una clase puede ser necesaria para el correcto funcionamiento de otra. Se identifica como tipo “parte-de”.
  - **Composición**. La composición es una forma de agregación donde la relación de propiedad es más fuerte, e incluso coinciden los tiempos de vida del objeto completo y las partes que lo componen.
  - **Dependencia**. Una relación de dependencia se utiliza entre dos clases o entre una clase y una interfaz, e indica que una clase requiere de otra para proporcionar alguno de sus servicios.

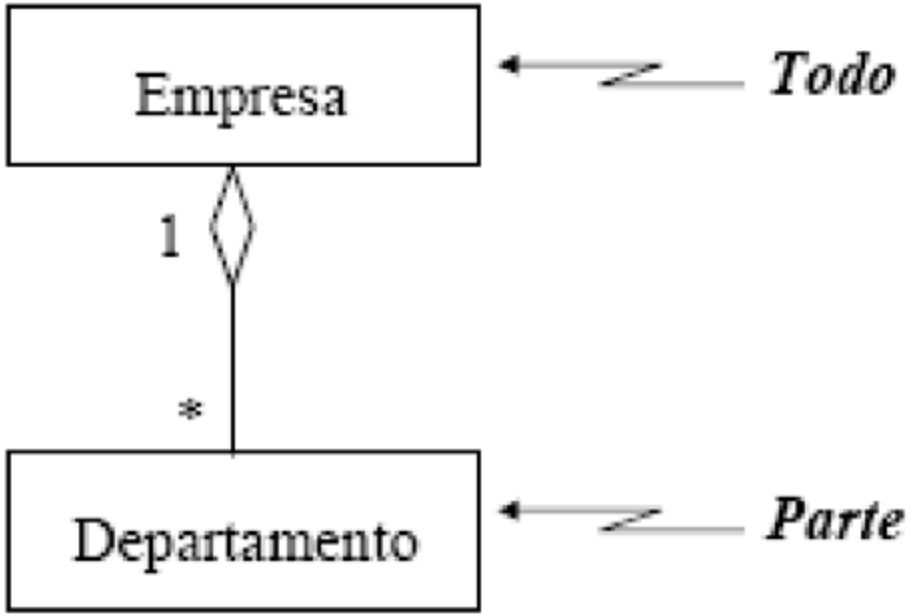
Tipos de Asociaciones

- **Composición:** Es una asociación fuerte que implica:
  - Dependencia existencial. El elemento dependiente desaparece al destruirse el que lo contiene, y si es de cardinalidad 1, es creado al mismo tiempo. Se puede decir que el objeto contenido es parte constitutiva y vital del que lo contiene.
  - Se denota dibujando un rombo del lado de la clase que contiene a la otra en la relación



Tipos de Asociaciones

- **Agregación:** Relaciona una clase con una clase componente. Es también una relación de composición menos fuerte (no se exige dependencia existencial) y se denota por un rombo sin rellenar en uno de los extremos.



Tipos de Asociaciones

- **Generalización:** es un proceso de abstracción en el cual un conjunto de clases existentes, que tienen atributos y métodos comunes, es referido por una clase genérica a un nivel mayor de abstracción.
- La relación de generalización denota una relación de herencia entre clases.
- Se representa dibujando un triángulo sin rellenar en el lado de la superclase. La subclase hereda todos los atributos y mensajes descritos en la superclase.

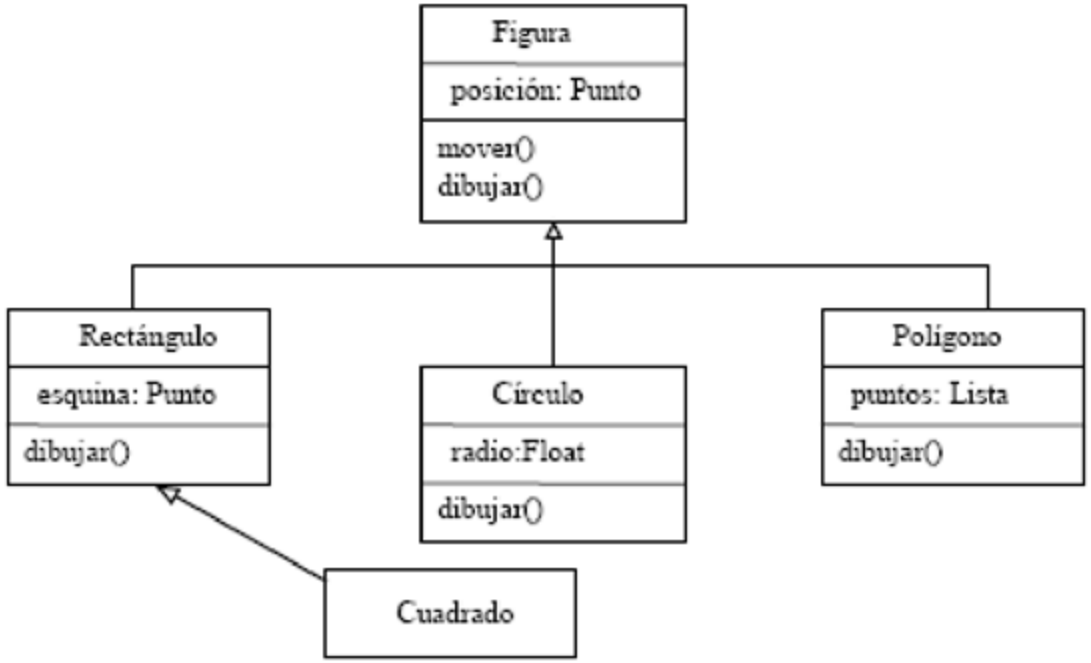


Diagrama de clases

- **Nombre:** Identifica la asociación entre los objetos, caracterizándola.
- **Rol:** Identificado como un nombre al final de la línea, describe la semántica de la relación en el sentido indicado. Cada asociación tiene dos roles; cada rol es una dirección en la asociación. El rol puede estar representado en el nombre de la clase.

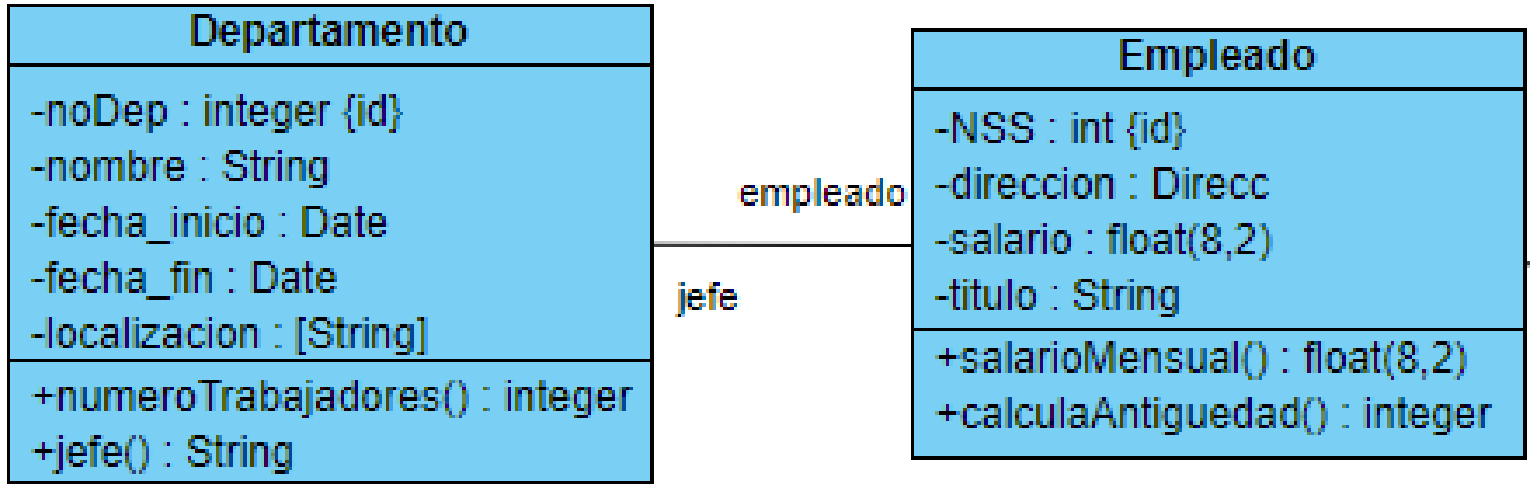
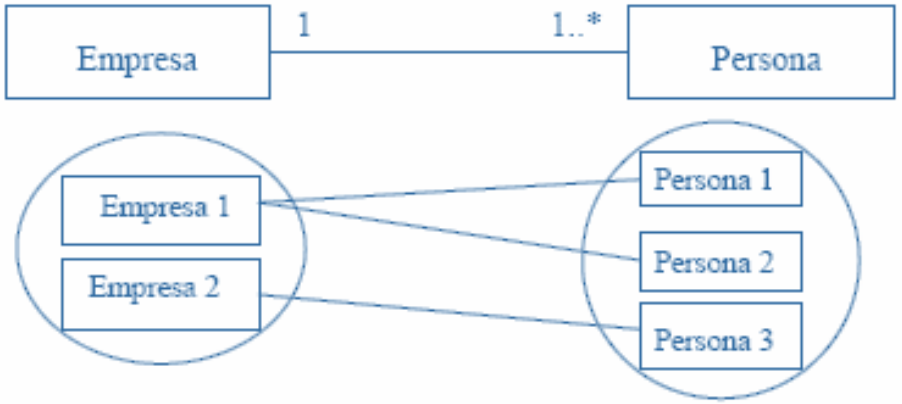


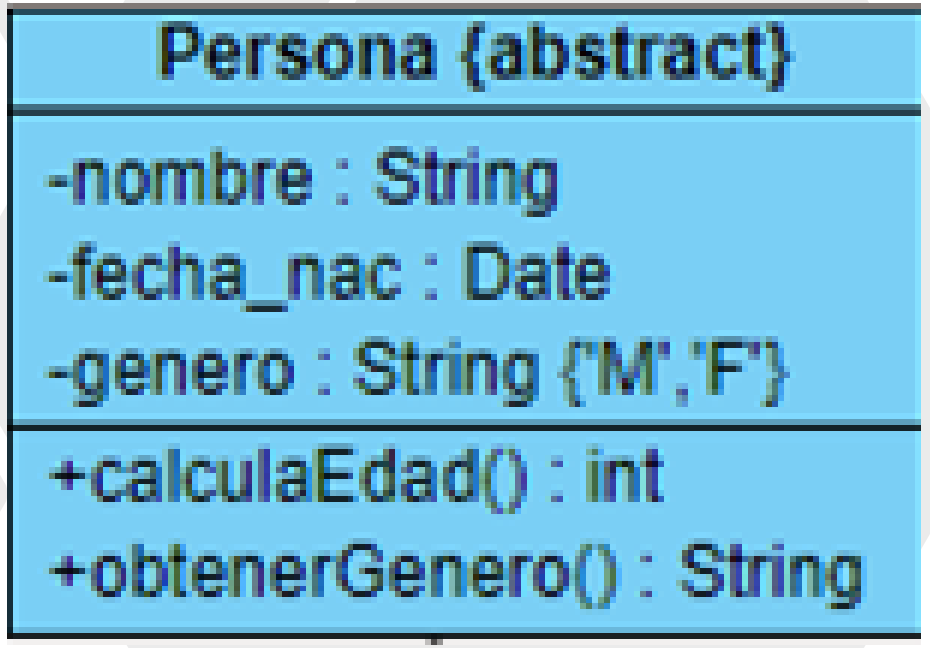
Diagrama de clases

- **Multiplicidad:** Describe la cardinalidad de la relación, es decir, cuántos objetos de esa clase pueden participar en la relación dada.
- Especificación de multiplicidad (mínima...máxima):
  - 1 Uno y sólo uno
  - 0..1 Cero o uno
  - M..N Desde M hasta N (enteros naturales)
  - * Cero o muchos
  - 0..* Cero o muchos
  - 1..* Uno o muchos (al menos uno)



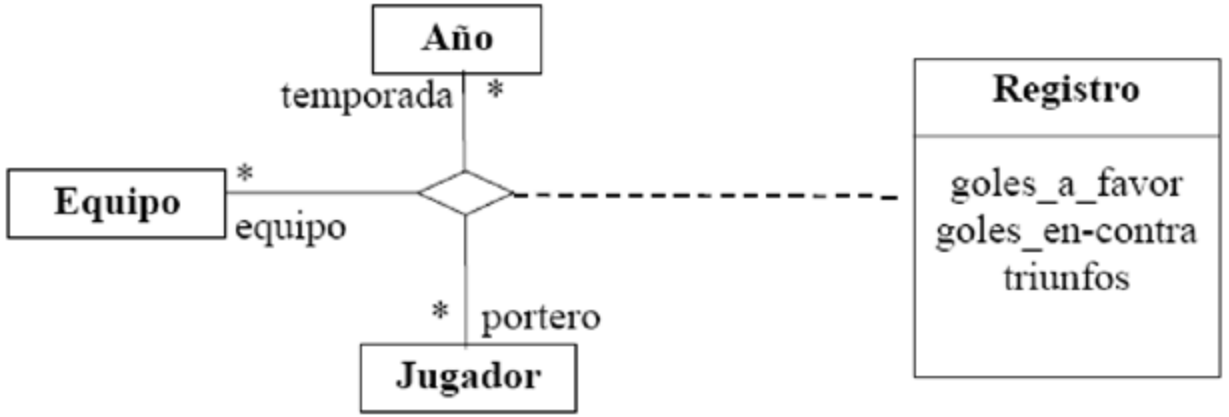
Clases Abstractas

- Representan clases que no contienen objetos. Sólo pueden aparecer dentro de jerarquías de generalización.
- Se pueden asociar a clases de una jerarquía en la que los objetos siempre deben pertenecer a uno de los subtipos definidos.
- Se representan poniendo el nombre de la clase en cursiva o mediante el valor etiquetado {abstract} debajo (o junto) del nombre de la clase



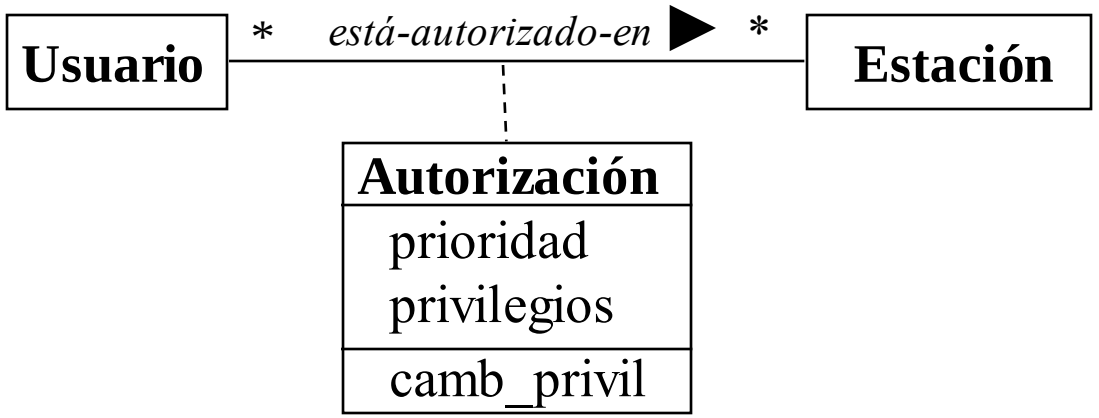
Tipos de Asociaciones

- **Asociación Binaria:** Representa una relación sencilla entre dos clases, no muy fuerte (es decir, no se exige dependencia existencial ni encapsulamiento). Se indica como una línea sólida que une dos clases.
- **Asociación n-aria:** Es una asociación entre tres o más clases. Se representa como un diamante del cual salen líneas de asociación a las clases.



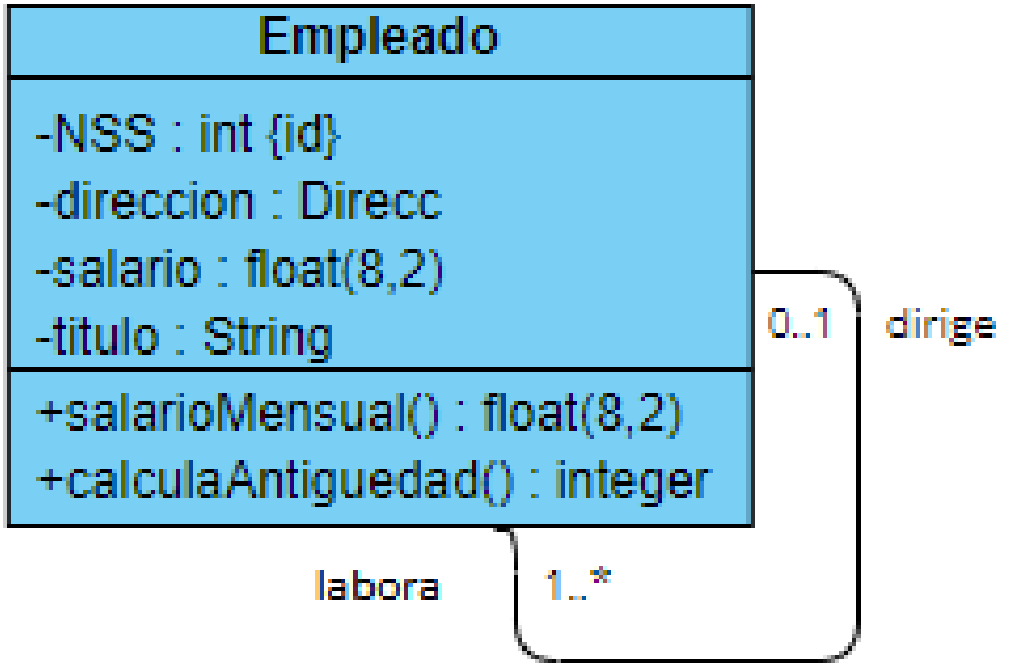
Clase de Asociación

- Una clase asociación permite agregar atributos y operaciones a una asociación entre dos o más clases.

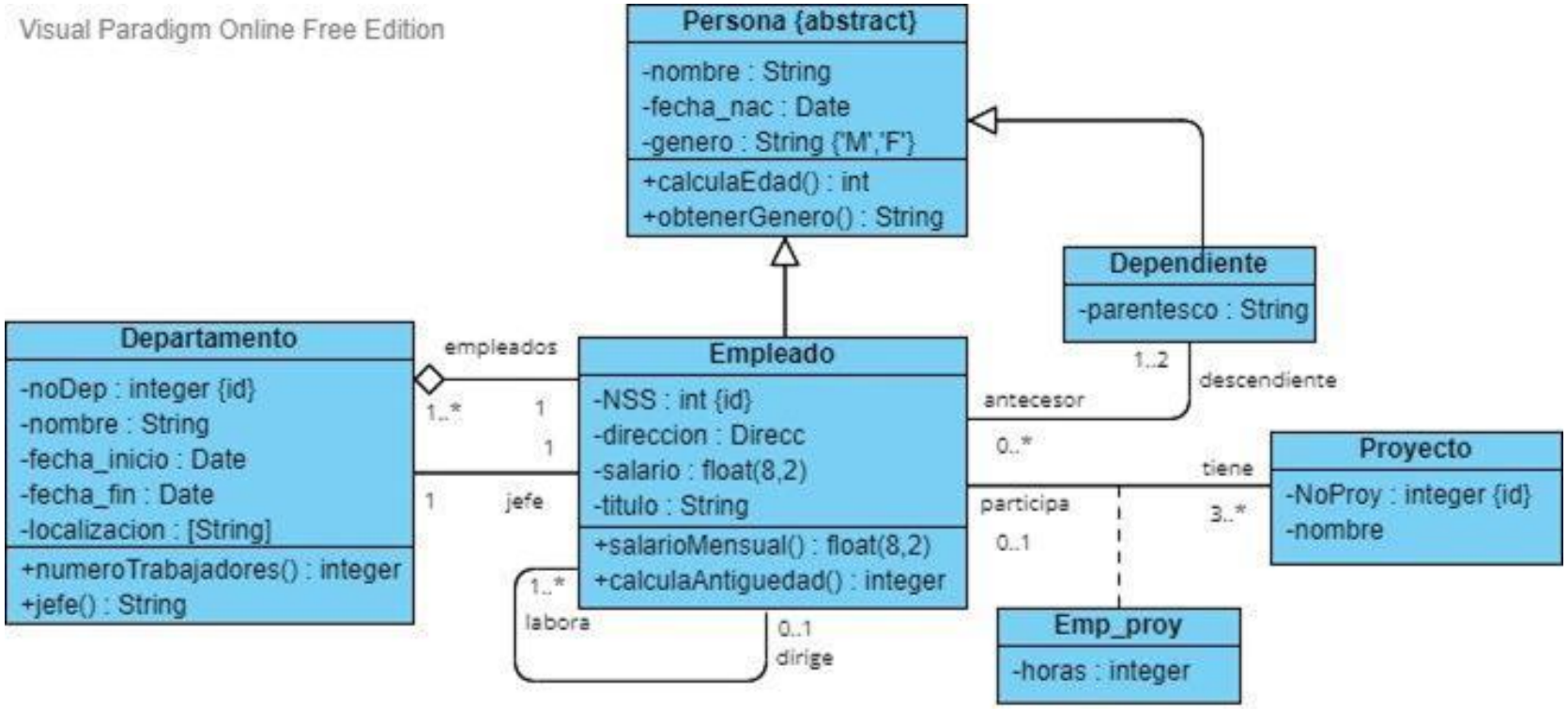


Asociación Reflexiva

- Una clase puede asociarse con sí misma.
- No significa que una instancia está relacionada consigo misma, sino que una instancia de la clase está relacionada con otra instancia de la misma clase



Ejemplo


Visual Paradigm Online Free Edition
Persona {abstract}
-nombre : String
-fecha_nac : Date
-genero : String {'M','F'}
+calculaEdad() : int
+obtenerGenero() : String
Dependiente
-parentesco : String
Departamento
-noDep : integer {id}
-nombre : String
-fecha_inicio : Date
-fecha_fin : Date
-localizacion : [String]
+numeroTrabajadores() : integer
+jefe() : String
empleados
1..*
1
1
1
jefe
Empleado
-NSS : int {id}
-direccion : Direcc
-salario : float(8,2)
-titulo : String
+salarioMensual() : float(8,2)
+calculaAntiguedad() : integer
1..2
descendiente
antecesor
0..*
tiene
3..*
Proyecto
-NoProy : integer {id}
-nombre
participa
0..1
Emp_proy
-horas : integer
1..*
labora
0..1
dirige

Características de un SABDOO

- De acuerdo con el American National Standards Institute (ANSI), un Sistema Administrador de Bases de Datos Orientados a Objetos (SABDOO), de preferencia debe cumplir las siguientes características:
  - **Modelo de objetos**.- Las clases y los objetos deben representar entidades del mundo real, siguiendo todas las características del paradigma de orientación a objetos.
  - **Extensibilidad**.- Debe permitir al usuario la definición y creación de nuevas clases y modificar las existentes de manera dinámica.
  - **Lenguaje de datos**.- Un SABDOO debe proporcionar enlaces (bindings) a lenguajes OO, lo que permite aprovechar todas sus ventajas. Actualmente, los fabricantes han incluido las características de OO en lenguajes procedurales tipo SQL.
  - **Biblioteca de clases**.- Habilita la reutilización de clases ya validadas por el fabricante, permitiendo reducir el tiempo de desarrollo.
  - **Persistencia**.- Permite conservar los objetos creados en el sistema de manera persistente, en un dispositivo de almacenamiento, para posteriormente ser usados.

Tipos de SBDOO

- A pesar de que el paradigma orientado a objetos data de la década de los años noventa del siglo pasado, no ha tenido el suficiente impulso por parte de la industria para su popularidad.
- De aquí, los fabricantes de estos sistemas se pueden clasificar como:
  - **SABDOO puros**.- En donde estos sistemas de software cumplen cabalmente con la filosofía de orientación a objetos. Como ejemplos tenemos O2, Objectivity, ObjectStore, Ontos, Versant, Jasmine, etc.
  - **SABD Objeto-Relacionales (OR)**.- Son sistemas con bases del modelo relacional, con algunos agregados en la maquinaria para permitir características orientadas a objetos. Como ejemplos se tienen aquellos sistemas que implementan el estándar SQL:2003.

El estándar SQL:2003

- Desde la definición del modelo relacional en la década de los setenta por Ted Codd, los fabricantes han ido mejorando los productos de software para incluir más capacidades requeridas por los usuarios. Entre estas necesidades, tenemos:
  - Nuevos tipos de datos y estructuras, para hacer más práctico el manejo de información más allá de tablas.
  - Almacenamiento de datos de diversos tipos, como formatos de sonido, imágenes, video, y especializados.
  - Capacidad de manejar grandes cantidades de datos en memoria, para mejorar el rendimiento, así como transacciones de más larga duración.
  - Mejoras en los lenguajes de consulta, que permita sacar provecho de los tipos de datos definidos por el usuario.
- Debido a la exigencia de los usuarios, la International Standards Organization (ISO) propuso el estándar SQL:2003, el cual presenta muchas características de OO.

Ventajas del SQL:2003

- Entre las mejoras propuestas al SQL:2003, se tienen las siguientes:
  - *Extensibilidad*: capacidad de ampliar el sistema de tipos de datos para dar soporte a las nuevas necesidades de las aplicaciones:
    - Nuevos tipos que representen mejor el dominio de valores.
    - Nuevas operaciones para soportar el comportamiento de estos tipos.
  - *Poder de expresión*: necesidad de proporcionar objetos y relaciones complejas.
  - *Reusabilidad*: capacidad de compartir librerías de tipos existentes.
  - *Integración*: correspondencia de los modelos relacional y orientado a objetos en un solo lenguaje.
  - *Nuevos tipos de consultas*: mejoras en el lenguaje de consultas para poder extraer datos de los nuevos tipos.

Modelo objeto-relacional de Oracle 10g

- Las primeras versiones comerciales del modelo objeto-relacional fueron, en general, para la aprobación del SQL:1999, primer estándar SQL que incorporaba extensiones de objetos.
- Es por ello que los productos no coinciden totalmente con el estándar, ni con el modelo ni en la sintaxis, existiendo además diferencias entre ellos.
- La primera versión de Oracle que incluía capacidades de objetos fue Oracle8.

M. en C. Alejandro Botello C.

Tipos de datos definidos por el usuario (UDT)

- Aunque la mayoría de los SABD tienen incluidos una gran variedad de tipos de datos nativos, a veces es necesario permitir la definición de los propios para la aplicación.
- En el SQL:2003 se permite definir la estructura de nuevos tipos de datos, así como su comportamiento.
- Una vez que es definido, el nuevo tipo de datos es usado como cualquier otro predefinido.
- Estos UDT se clasifican como:
  - **Tipos distintos**, que soportan la noción de fuerte tipado y además tienen asociado el comportamiento,
  - **Tipos estructurados**, igual que los anteriores, además soportan otras características como encapsulación, polimorfismo y vinculación dinámica.

Sintaxis: creación de tipos

```
CREATE TYPE nombreUDT AS OBJECT
[UNDER nombreSupertipoUDT]
[AS {tipoPredefinido |
listaAtributos}]
[INSTANTIABLE | NOT INSTANTIABLE]
[{FINAL | NOT FINAL}]
[[OVERRIDING] {STATIC | MEMBER}
FUNCTION | PROCEDURE nombreMetodo]
[RETURNS tipoDato]
```

- **Nota**: No es posible establecer la restricción de NOT NULL en los atributos.
- No se pueden usar los tipos de datos *ROWID, LONG*, o *LONG RAW*.

Ejemplo de tipos estructurados (Oracle)

```
CREATE OR REPLACE TYPE tipoDireccion AS OBJECT (
calle VARCHAR2 (30),
colonia VARCHAR2 (30),
ciudad VARCHAR2 (25))
/
CREATE TABLE Jugador (
nombre VARCHAR2(30),
vive_en tipoDireccion,
foto BLOB);
```

```
CREATE OR REPLACE TYPE tipoEmpleado AS OBJECT
(DNI NUMBER,
nombre VARCHAR2 (30),
fecha_nac DATE)
/
CREATE TABLE Empleado OF tipoEmpleado;
```

Modificación de tipos

- Es posible modificar la estructura de un tipo ya definido mediante ***CREATE TYPE***, pero tomando en consideración que puede tener objetos dependientes de él (otros tipos o tablas).

```
ALTER TYPE tipo ADD ATTRIBUTE atributo
tipo_dato CASCADE;
```

```
ALTER TYPE tipo [NOT] INSTANTIABLE [NOT] FINAL;
```

- Para agregar un método, es necesario emplear dos sentencias: una para la especificación y otra para el cuerpo:

```
ALTER TYPE tipo ADD MEMBER {FUNCTION
|PROCEDURE} definicion_metodo CASCADE;
```

```
CREATE OR REPLACE TYPE tipo
AS implementacion_metodo;
```

Eliminación de tipos

- Para eliminar un tipo creado, se debe incluir el modificador ***FORCE*** para evitar los problemas de los objetos dependientes del tipo. Esto puede producir objetos inválidos dentro del esquema del usuario.

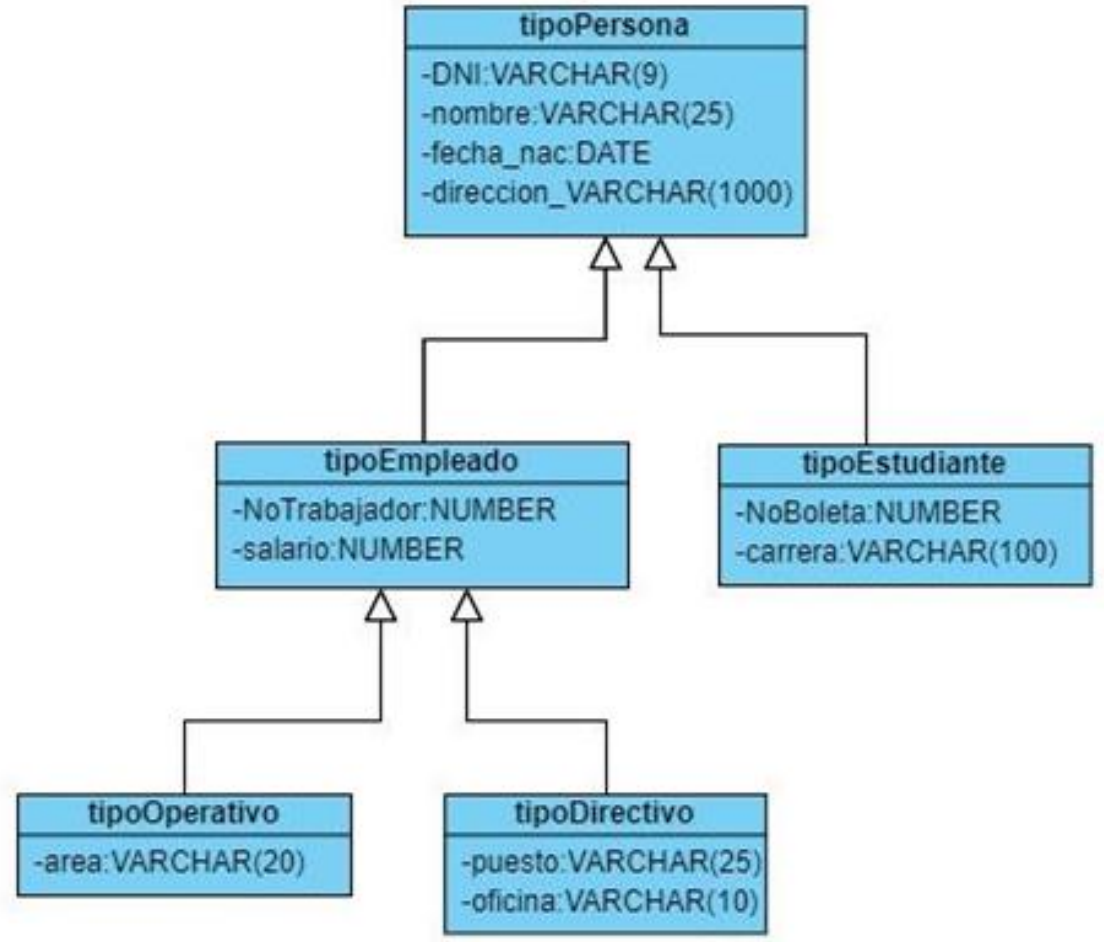
```
DROP TYPE tipo FORCE;
```

- Para eliminar un atributo de un tipo, se emplea la sentencia ALTER TYPE:

```
ALTER TYPE tipo DROP ATTRIBUTE atributo CASCADE [NOT]
INCLUDING TABLE DATA;     -- verifica los datos
```

Herencia

- El estándar SQL:2003 permite la definición de herencia simple de tipos estructurados y de tablas tipadas.
- La herencia de tablas hace referencia a la parte extensional de una jerarquía, es decir, al principio de clasificación por el que toda instancia del subtipo es también una instancia del supertipo.
- Las subtablas heredan los atributos de la supertabla
- Las jerarquías son parciales y exclusivas; si se quiere que sean totales se debe declarar el supertipo como ***NOT INSTANTIABLE***.



Herencia

- El tipo raíz de una jerarquía se crea empleando la sentencia CREATE TYPE y debe ser declarado como NOT FINAL. La opción por defecto es FINAL (indicando así, que no pueden derivarse subtipos del tipo superior).

```
CREATE TYPE tipoPersona AS OBJECT(
DNI VARCHAR(9),
nombre VARCHAR(25),
fecha_nac DATE,
direccion VARCHAR(1000)) NOT FINAL;
CREATE TYPE tipoEmpleado UNDER tipoPersona (
NoTrabajador NUMBER,
salario NUMBER)NOT FINAL;
CREATE TYPE tipoEstudiante UNDER tipoPersona (
noBoleta NUMBER,
carrera VARCHAR(100));
CREATE TYPE tipoOperativo UNDER tipoEmpleado (
area VARCHAR(20));
CREATE TYPE tipoDirectivo UNDER tipoEmpleado (
puesto VARCHAR(25),
oficina VARCHAR(10));
```

```
CREATE TABLE persona OF tipoPersona;
INSERT INTO persona VALUES('12345', 'Juan
Pérez',TO_DATE('12/4/1978','dd/mm/yyyy'),
'Av. Palmas 23, Villas, Toluca');
CREATE TABLE empleado OF tipoEmpleado;
INSERT INTO empleado VALUES (9876,
8756.50);   -- CONSTRUCTOR INCORRECTO
INSERT INTO
empleado VALUES('45678','Laura Lopez',
TO_DATE('9/11/1986','dd/mm/yyyy'), 'Cadiz
564, Torres, Toluca', 9876, 8756.50);
CREATE TABLE operativo OF tipoOperativo;
INSERT INTO operativo
VALUES('86420','José Aguado',
TO_DATE('23/9/1982','dd/mm/yyyy'), 'Calle
34 #298, Oriente, Ecatepec', 6754,
6789.00, 'Soporte');
CREATE TABLE directivo OF tipoDirectivo;
INSERT INTO directivo VALUES('97642','Ana
Padilla',TO_DATE('7/3/1992','dd/mm/yyyy')
, 'Juan Escutia 213, Alamedas, Izcalli',
8798, 9782.80, 'Director de Personal',
'C-14');
```

Tipos abstractos

- Generalmente la raíz de una jerarquía es un prototipo para los tipos descendientes. Si se permite la inserción de registros en una tabla instanciada del tipo raíz, se tienen registros genéricos (no especializados).
- Para solucionar esto, es posible modificar el tipo raíz para que sea a abstracta, mediante el modificador **[NOT] INSTANTIABLE**.

```
SQL>ALTER TYPE tipoPersona NOT INSTANTIABLE
*
ERROR at line 1:
ORA-22327: cannot change a type to NOT INSTANTIABLE if it has dependent tables
```

- Hay que eliminar las tablas y los tipos dependientes (jerarquía)

```
CREATE TABLE Persona OF tipoPersona;
insert into persona values ('12345', 'Juan Pérez',
TO_DATE('12/4/1978','dd/mm/yyyy'), 'Av. Palmas 23, Villas, Toluca');
ERROR at line 1:
ORA-22826: cannot construct an instance of a non instantiable type
```

Funciones constructoras

- Un constructor es un método automáticamente definido por el compilador, que permite la generación de instancias de un tipo determinado.
- Como características presenta el mismo nombre del tipo, y presenta tantos parámetros de creación de la instancia como atributos tenga el tipo (y en el mismo orden).
- Regresa como resultado la instancia del objeto creado.
- Se define la palabra reservada ***NEW*** para instanciar a un objeto, aunque en la práctica se omite:

```
UPDATE empleado e SET e.domicilio =
NEW tdireccion('Av.Siempreviva', 444, 'col. San Mateo',
'Ecatepec', '89740') WHERE nss = '111222333';
```

- En la sentencia anterior, se establece el valor del domicilio del empleado invocando al constructor del tipo *tdirección*, con los parámetros correspondientes.

- Se permite que el usuario agregue sus propias funciones constructoras parametrizadas al momento de crear un tipo, vigilando que no haya conflicto en la definición:

```
ALTER TYPE templeado ADD CONSTRUCTOR FUNCTION templeado(SELF IN OUT NOCOPY
templeado, nss INTEGER, nomb VARCHAR, apell VARCHAR, fnac VARCHAR, gen CHAR, calle
VARCHAR, num INTEGER, col VARCHAR, cd VARCHAR, cp VARCHAR) RETURN SELF AS RESULT;

CREATE OR REPLACE TYPE BODY templeado AS

CONSTRUCTOR FUNCTION templeado(SELF IN OUT NOCOPY templeado, nss INTEGER, nomb
VARCHAR, apell VARCHAR, fnac VARCHAR, gen CHAR, calle VARCHAR, num INTEGER, col
VARCHAR, cd VARCHAR, cp VARCHAR) RETURN SELF AS RESULT IS

BEGIN
    SELF.nss := nss;
    SELF.nombre := nomb;
    SELF.apellido := apell;
    SELF.fecha_nac := TO_DATE(fnac,'dd-mm-YYYY');
    SELF.genero := gen;
    SELF.domicilio := NEW tdireccion(calle, num, col, cd, cp);
RETURN;
END;
END;
```

Métodos

- Los métodos son funciones SQL ligadas a un tipo estructurado que representan el comportamiento de dicho tipo.
- Si el método va a pertenecer al tipo se establece la palabra ***STATIC***. En cambio, si va a pertenecer a las instancias, se usa la palabra ***MEMBER***.
- Para hacer referencia a una instancia dentro de un subprograma se emplea el identificador ***SELF***.
- La firma del método se especifica junto a la definición del tipo de datos al que va ligado.
- La especificación del cuerpo es separada.
- Es posible redefinir un método de un supertipo mediante la cláusula ***OVERRIDING***.

```
CREATE OR REPLACE TYPE tipoPersona AS OBJECT(

DNI VARCHAR(9),

nombre VARCHAR(25),

fecha_nac DATE,

direccion VARCHAR(1000),

MEMBER FUNCTION calcula_edad(fn DATE) RETURN INTEGER

)NOT FINAL;

CREATE OR REPLACE TYPE BODY tipoPersona AS

 MEMBER FUNCTION calcula_edad(fn DATE) RETURN INTEGER
 AS

  BEGIN

    RETURN TRUNC(MONTHS_BETWEEN(SYSDATE, fn)/12);

  END;

END;
```

Redefinición de Métodos

- Si, en una jerarquía de tipos, alguno de los subtipos desea cambiar la implementación de un método heredado, debe emplear la opción OVERRIDING en la definición del método propio:

```
CREATE OR REPLACE TYPE tipoEmpleado UNDER tipoPersona (
NoTrabajador NUMBER,
salario NUMBER,
OVERRIDING MEMBER FUNCTION calcula_edad(fn DATE) RETURN INTEGER
);
CREATE OR REPLACE TYPE BODY tipoEmpleado AS
OVERRIDING MEMBER FUNCTION calcula_edad(fn DATE) RETURN INTEGER AS
BEGIN
RETURN EXTRACT(YEAR FROM SYSDATE) - EXTRACT(YEAR FROM fn);
END;
END;
```

- No obstante, se debe conservar la firma de los métodos.

Tipos Colección

- El modelo de datos de Oracle 10g soporta dos tipos de colecciones: VARRAY y NESTED TABLE.
- El tipo VARRAY es similar al ARRAY del SQL:2003, pero difiere ligeramente la sintaxis de definición y su utilización.
- VARRAY es un tipo de longitud variable; es decir, se almacena sólo la longitud ocupada del array.:

```
CREATE OR REPLACE TYPE tipoTelefono AS VARRAY (3) OF VARCHAR (10);
CREATE TABLE Empleado
(DNI NUMBER,
nombre VARCHAR2 (30),
telefonos_contacto tipoTelefono);
INSERT INTO Empleado(DNI, nombre, telefonos_contacto)
VALUES (9876543, 'Pepe', tipoTelefono('914445566', '606445566', '934445566'));
```

Métodos para manejo de colecciones

- Las colecciones son tipos de objetos y, como tales, tienen definidos una serie de métodos, con la salvedad de que sólo pueden llamarse desde órdenes procedimentales PL/SQL y no desde órdenes SQL.
- Estos métodos se resumen en la tabla siguiente:

Método	Descripción
EXIST	Determina si existe un elemento de una colección
COUNT	Devuelve el número de elementos de una colección
LIMIT	Devuelve el número máximo de elementos de una colección
FIRST & LAST	Devuelve el primer (o último) elemento de una colección
NEXT & PRIOR	Devuelve el electo posterior a anterior a uno dado de una colección
EXTEND	Añade elementos a la colección
TRIM	Elimina los elementos del final de una colección
DELETE	Elimina los elementos especificados de una colección

Ejemplo

- Para obtener de manera individual cada uno de los elementos de un VARRAY:

```
DEFINE
TYPE telefonos IS VARRAY(5) OF VARCHAR(15);
tels := telefonos('5298674567','5298568478','5298123421');
mensaje VARCHAR(100) := '';
BEGIN
FOR i IN 1..tels.COUNT LOOP
  mensaje := mensaje || 'teléfono [' || i || ']: ' || tels(i) || ' ';
END LOOP;
DBMS_OUTPUT.PUT_LINE('Los telefonos son: ' || mensaje);
END;
```

Consultando VARRAY

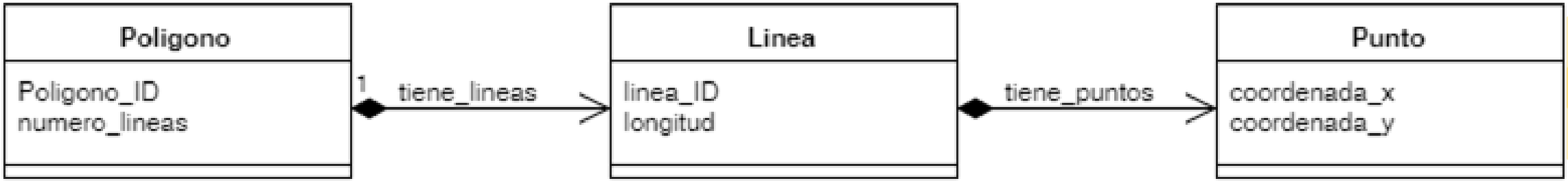
- A diferencia del estándar, desde una sentencia SELECT no es posible acceder a los elementos individuales del VARRAY indexándolos; sólo es posible acceder a la totalidad del mismo.
- Para el tratamiento individual de cada elemento del VARRAY, es necesario hacerlo mediante PL/SQL, que proporciona cláusulas para posicionarse en los distintos elementos del array:

```
SELECT * FROM empleado;

DNI             NOMBRE          TELEFONOS_CONTACTO
_____________________________________________________________________
9876543         Pepe            TIPOTELEFONO ('914445566', '606445566', '934445566');
```

NESTED TABLE

- Oracle soporta otro tipo de datos colección que son las tablas anidadas o NESTED TABLES y que se corresponden con el tipo MULTISET del SQL:2003.
- Es posible definir un tipo de datos como una tabla, y utilizar dicho tipo como el tipo de datos de la columna de otra tabla.
- Aunque Oracle almacena las filas de una NESTED TABLE sin orden, al recuperarlas es posible referirse a ellas según un orden de indexación que empieza en el 1.
- Al definir una tabla tipada que contiene un atributo que es de tipo tabla, es necesario darle un nombre de almacenamiento a la **NESTED TABLE** mediante la cláusula **STORE AS**. Sin embargo, el nombre asignado mediante la cláusula anidada **STORE AS** es un nombre interno y no se puede acceder directamente a la tabla anidada.
- La tabla anidada es un tipo de colección y funciona como tal. Por tanto, el acceso a sus miembros (bien para inserción, borrado o consulta) es necesario hacerlo a través la tabla principal.

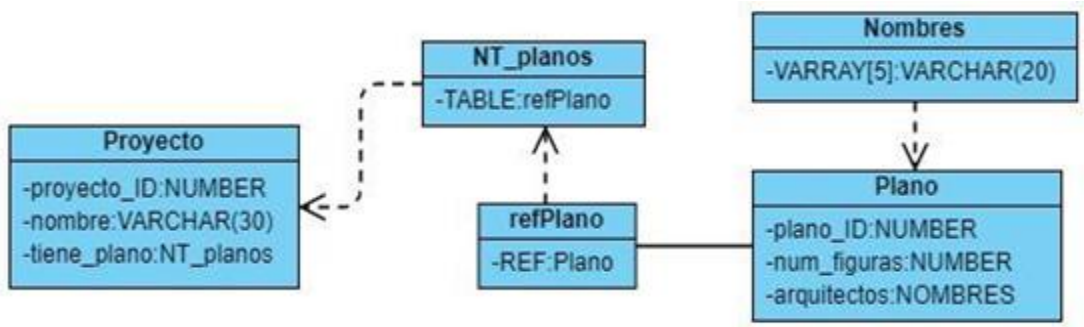


- A continuación, se presenta el código SQL que permite transformar las clases del diagrama superior en tipos objeto definidos por el usuario en la BD. Para ello, se define un tipo *tipoPoligono* que contiene una NESTED TABLE para almacenar objetos de tipo *tipoLinea*; a su vez, cada uno de estos objetos contiene una NESTED TABLE con los puntos origen y final de dicha línea.

```
CREATE TYPE tipoPunto AS OBJECT
(coord_X NUMBER (3),
coord_Y NUMBER (3));
CREATE OR REPLACE TYPE tipoPuntos AS VARRAY (2) OF tipoPunto;
CREATE TYPE tipoLinea AS OBJECT
(linea_ID NUMBER(3),
longitud NUMBER,
tiene_puntos tipoPuntos);
CREATE OR REPLACE TYPE NT_Lineas AS TABLE OF tipoLinea;
CREATE TYPE tipoPoligono AS OBJECT
(poligono_ID NUMBER (12),
numero_lineas NUMBER (3),
tiene_lineas NT_Lineas);
```

```
CREATE TABLE Poligono OF tipoPoligono
(PRIMARY KEY (Poligono_ID), tiene_lineas NOT NULL)
NESTED TABLE tiene_lineas STORE AS ListaLineas;
INSERT INTO poligono VALUES(1,3, NT_lineas(
tipolinea(1,4,tipopuntos(tipopunto(3,5),tipopunto(3,7))),
tipolinea(2,3,tipopuntos(tipopunto(3,5),tipopunto(5,5))),
tipolinea(3,5,tipopuntos(tipopunto(5,5),tipopunto(3,5)))
);
SELECT p.poligono_id, p.numero_lineas, t.linea_id,
t.longitud, t.tiene_puntos FROM poligono p,
TABLE(tiene_lineas) t;
```

```
CREATE TYPE tipoNombres AS VARRAY(5) OF
VARCHAR(20)
/
CREATE TYPE tipoPlano AS OBJECT
(plano_ID NUMBER,
num_figuras NUMBER,
arquitectos tipoNombres)
/
CREATE TABLE Plano OF tipoPlano
(PRIMARY KEY (plano Id));
CREATE OR REPLACE TYPE refTipoPlano AS OBJECT
(refPlano REF tipoPlano)
/
CREATE OR REPLACE TYPE NT_Planos AS
TABLE OF refTipoPlano
/
CREATE OR REPLACE TYPE tipoProyecto AS OBJECT
(proyecto_ID NUMBER,
nombre VARCHAR (30),
tiene_plano NT_Planos)
/
CREATE TABLE Proyecto OF tipoProyecto
(PRIMARY KEY(proyecto_ID), UNIQUE(nombre))
NESTED TABLE tiene_plano STORE AS ListaPlanos;
```




```
INSERT INTO Plano(plano_ID,num_figuras,arquitectos) VALUES (1,
5, tipoNombres('Juan Perez', 'Maria Lopez', 'Marco Marquez'));

INSERT INTO Plano (plano_ID,num_figuras,arquitectos) VALUES(2,
4, tipoNombres('Jose Martínez','Laura Lopez'));

INSERT INTO Proyecto(proyecto_ID,nombre) VALUES (1, 'MIDAS',
 NT_Planos((SELECT refTipoPlano(REF(p)) FROM Plano p WHERE
 p.plano_id=1)));
```

```
DECLARE
p_ref refTipoPlano;
BEGIN
SELECT refTipoPlano(REF(p)) INTO p_ref FROM plano p WHERE
 p.plano_id=2;
UPDATE proyecto p SET tiene_plano = NT_Planos(p_ref) WHERE
 proyecto_id=1;
END;
/

SELECT p.nombre, t.refplano.plano_id FROM proyecto p,
TABLE(tiene_plano) t;

SELECT p.nombre, t.refplano.plano_id, t.refplano.arquitectos
FROM proyecto p, TABLE(tiene_plano) t;
```

Tipos colección multinivel

- Oracle permite anidar colecciones empleando los tipos colección multinivel, (no soportados hasta la versión 9i del producto) y que son tipos colección cuyos elementos son en sí mismos (directa o indirectamente) otros tipos colección.
- Existen varios tipos colección multinivel:
  - ***NESTED TABLE*** de tipo ***NESTED TABLE***
  - ***NESTED TABLE*** de tipo ***VARRAY***
  - ***VARRAY*** de tipo ***NESTED TABLE***
  - ***VARRAY*** de tipo ***VARRAY***
  - ***NESTED TABLE*** o ***VARRAY*** de un tipo definido por el usuario que tiene un atributo que es a su vez de tipo ***NESTED TABLE*** o ***VARRAY***.

Diferencias entre VARRAY y NESTED TABLE

- Los VARRAYs tienen un tamaño máximo fijo que se especifica en la definición del tipo, mientras que las NESTED TABLES son de tamaño variable (no se dimensionan)
- Los VARRAYs se guardan en el mismo espacio que la tabla. Sin embargo, las NESTED TABLES se almacenan como otra tabla independiente asociada a la tabla sobre la que está definida.
- A la hora de elegir entre una NESTED TABLE o un VARRAY hay varios criterios a seguir:
  - Si el orden en que se almacenan los elementos de la colección es relevante, se emplea un VARRAY, puesto que la NESTED TABLE no conserva el orden.
  - En caso de saber de antemano el número de elementos que tendrá la colección, se emplea un VARRAY porque permite limitar su tamaño en el momento de su definición.
- En general, si el tamaño y el orden no son especialmente relevantes, el criterio a seguir para elegir entre un VARRAY o una NESTED TABLE será el siguiente: si se necesitan consultas sobre la colección, que nos permitan tratar los elementos de la colección por separado, se emplea una NESTED TABLE, mientras que si se desea recuperar la colección como un todo, se empleará un VARRAY, aunque también se permitiría el acceso a los elementos individuales.

Tipos REFERENCIA

- Un tipo Referencia (REF) es un tipo de datos que contiene el valor del atributo REF (equivalente con un OID) de una fila de una tabla tipada.
- El tipo referencia permite implementar relaciones prescindiendo de la utilización de llaves foráneas. Un atributo definido como tipo REF contendrá el valor del atributo REF del objeto referenciado.

```
CREATE TYPE tipoPropiedad AS OBJECT (
propietario REF(tipoEmpleado),
precio INTEGER,
num_habitaciones INTEGER,
tamaño DECIMAL(8,2),
ubicación tipoDireccion);
CREATE TABLE Propiedades OF tipoPropiedad;
```

Tipo REF

- Es posible definir un atributo como una referencia a un tipo de objeto:

```
CREATE OR REPLACE TYPE tipoDepartamento AS OBJECT
nombre_dep VARCHAR2 (30))
/
CREATE OR REPLACE TYPE tipoEmpleado AS OBJECT
(DNI NUMBER,
nombre VARCHAR2 (30),
fecha_nac DATE,
pertenece_a REF tipoDepartmento)
/
CREATE TABLE Departamento OF tipoDepartamento;
CREATE TABLE Empleado OF tipoEmpleado;
```

Para insertar un registro en la tabla que tiene el atributo de tipo ***REF***, es necesario hacer una consulta para obtener el único registro que cumpla la condición, y así tener su referencia mediante el operador ***REF***().

```
INSERT INTO Empleado VALUES (12345, 'Juan
Pérez', TO_DATE('18/09/1997', 'dd/mm/yyyy'),
(SELECT REF(d) FROM departamento d
WHERE nombre_dep = 'Ventas'));
```

Se emplea la notación punto "." para acceder a los datos:

```
SELECT nombre, e.pertenece_a.nombre_dep
FROM Empleado e
WHERE DNI = 9687452;
```

Funciones de manipulación de referencias

- ***REF()*** designa un modificador del tipo de dato a un operador para obtener un apuntador lógico de un objeto. Este apuntador encapsula el OID y simplifica la navegación entre los objetos relacionados.

```
REF(alias_tabla)
```

- Se puede verificar que no haya apuntadores "colgados" (dangling), que se pueden producir por eliminación de los objetos a los cuales apuntan. Para ello se incluye el modificador IS DANGLING:

```
UPDATE departamento d SET d.jefe = NULL WHERE d.jefe IS DANGLING;
```

- ***VALUE()*** obtiene un renglón, como objeto, más que como formato de tabla.

```
VALUE(alias_tabla)
```

- ***DEREF()*** se usa para obtener el contenido de un objeto para el cual existe una referencia.

```
DEREF(alias_tabla)
```

Tipos largos de datos (LOB)

- Los tipos de objetos largos (Large OBject) permiten almacenar una gran cantidad de información (hasta Gigabytes).
- Son útiles para el almacenamiento de imágenes, sonido, texto formateado, y otras necesidades de tipo multimedia.
- Existen dos tipos:
  - **BLOB** (Binary Large OBject): permite almacenar datos en formato binario.
  - **CLOB** (Character Large OBject): permite almacenar grandes cantidades de texto.
- Estos tipos de datos almacenan toda la información en el sistema, no de manera externa, por lo que el tamaño de la base de datos crece de manera considerable.

LOBS

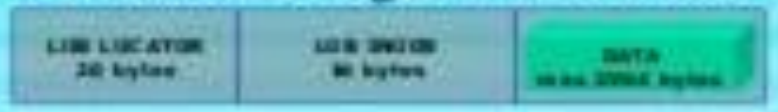
- El tipo de datos predefinido LOB (Large OBjects) permite manejar las necesidades de almacenamiento de imágenes, videos, documentos y en general, cualquier tipo de dato no estructurado y de tamaño grande.
- Además, estos datos no estructurados, en general texto, imágenes, vídeos o archivos de sonido, suelen tener gran volumen.
- Si el texto o información multimedia se guarda dentro de la base de datos, el contenido se almacena en un segmento separado de la tabla. Este segmento es de tipo LOB y almacena solo el LOB Value mientras la tabla que se definió con él campo LOB solo lleva el LOB Locator como puntero al segmento LOB.
- Con el fin de minimizar el espacio de almacenamiento en la BD, la información de un LOB se guardará en archivos del sistema operativo: en el campo correspondiente al valor de tipo LOB habrá un apuntador que permitirá acceder al contenido de archivo.

Tipos de LOB

- **INTERNOS**: son aquellos que se almacenan en los espacios de tabla (tablespaces) de la propia BD, bien en la misma tabla (in-line), o bien en un segmento o espacio de tabla distinto (off-line). A este subtipo pertenecen los BLOB (Binary Large objects), compuestos de datos binarios no estructurados, y los CLOB (Character Large oBject) y NCLOB (National Character Large OBjects), que almacenan datos de tipo carácter; estos dos últimos se diferencian sólo en el conjunto de caracteres que emplean para almacenar la información que contienen.
- **EXTERNOS**: los que almacena el sistema de archivos del sistema operativo. Los únicos LOBs externos son los BFILEs, que son grandes cantidades de datos binarios. Una columna o un atributo de tipo BFILE almacena un apuntador al comienzo del archivo que contiene los datos.

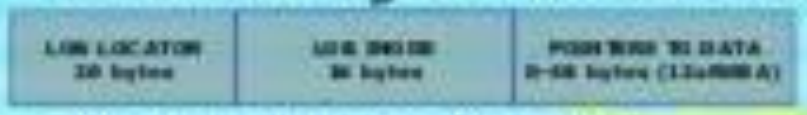
LOB Column Internal Structures

- *enable storage in row* inline LOB



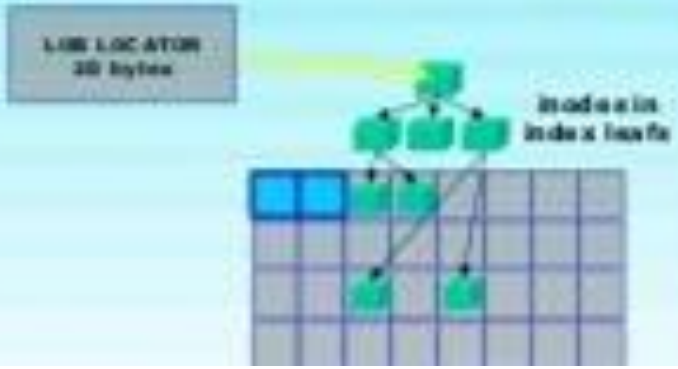
Max. total column size 4000 bytes
No LOB index entries
No LOB segment entries

- *enable storage in row* out-of-line LOB



Up to 12 4-byte relative DBAs are stored inline. If LOB grows larger, LOB index will be used to find data chunks

- *disable storage in row*



LOB index stores inodes for large LOB items, also for old chunk versions for providing read consistency

Almacenamiento de LOB

Oracle puede almacenar en la misma tabla LOBs pequeños de hasta 4K tamaño; cuando el LOB supera ese tamaño, el SABD lo almacena fuera de la tabla, en otra ubicación física y pone en su lugar el localizador que permita acceder a la nueva ubicación de los datos

Los atributos LOBs son capaces de almacenar hasta 4G de datos; además, se pueden definir varias columnas de tipo LOB en una misma tabla, así como varios atributos de un tipo de objeto pueden ser de tipo LOB.

Ejemplo de BLOB

- Como ejemplo de uso de los tipos de datos largos, considere la siguiente definición:

```
CREATE TABLE Libro (
titulo VARCHAR(200),
id_libro INTEGER PRIMARY KEY,
portada BFILE,
texto_libro CLOB,
pelicula BLOB
);
```

- Oracle recomienda inicializar un campo ***CLOB*** con un ***LOB Locator*** vacío y no dejarlo como NULL.
- Se puede hacer mediante la función ***EMPTY_CLOB()*** desde la creación de la tabla o después, por ejemplo:

```
CREATE TABLE Libro (
titulo VARCHAR(200),
id_libro INTEGER PRIMARY KEY,
portada BFILE,
texto_libro CLOB DEFAULT EMPTY_CLOB(),
pelicula BLOB DEFAULT EMPTY_BLOB()
);
```

- ó

```
ALTER TABLE Libro MODIFY texto_libro DEFAULT EMPTY_CLOB();
ALTER TABLE Libro MODIFY pelicula DEFAULT EMPTY_BLOB();
```

Modificación de LOBs

- Para insertar un valor de texto sobre un campo **CLOB**, se realiza de manera similar como si fuera un campo VARCHAR, ejemplo:

```
INSERT INTO Libro(id_libro, texto_libro) Libro VALUES (2,'texto muy muy largo del libro');
```

- Para obtener una parte del contenido de un **CLOB** se usa la función **DBMS_LOB.SUBSTR**:

```
SELECT DBMS_LOB.SUBSTR(texto_libro,20,10) FROM Libro;   --obtiene 20 caracteres de texto desde la posición 10
```

- Para obtener la posición de un texto se usa la función **DBMS_LOB.INSTR**:

```
SELECT DBMS_LOB.INSTR(texto_libro,'y',1,2) FROM Libro;  --obtiene la posición del carácter 'y' en su segunda ocurrencia desde la primera posición
```

- Para agregar más texto a un **CLOB** se usa la función **DBMS_LOB.WRITEAPPEND**:

```
DECLARE

v_clob CLOB;

BEGIN

SELECT texto_libro INTO v_clob FROM Libro WHERE id_libro = 2 FOR UPDATE;

DBMS_LOB.WRITEAPPEND(v_clob, LENGTH(' al final'), ' al final');

COMMIT;

END;

/
```

Manejo de directorios

- Para trabajar con **BLOB** y **BFILEs** se requiere de objetos **DIRECTORY** (directorios) en la base de datos.
- Los directorios no son objetos que le pertenecen a un esquema, todos los directorios creados son propiedad del usuario SYS.
  - Para crear directorios es necesario el privilegio de sistema ***CREATE ANY DIRECTORY***.
- Estos objetos directorios serán una referencia a una ubicación de un directorio del sistema operativo:

```
CREATE OR REPLACE DIRECTORY repositorio AS '/repositorio'  -- en Windows será c:\repositorio
```

- Se pueden agregar permisos a otros usuarios (precaución!!!):

```
GRANT READ, WRITE ON DIRECTORY repositorio TO user1, user2;
```

- Para obtener los metadatos de los directorios se consulta la tabla del sistema *DBA_DIRECTORIES*.

Modificación de un BLOB

- Para almacenar el contenido multimedia en un campo **BLOB**, es necesario copiar los archivos que contienen los datos binarios en la carpeta indicada por el objeto directorio y aplicar las funciones de manejo de **BLOB** de Oracle:

```
DECLARE

v_blob BLOB;

v_file BFILE;

BEGIN

INSERT INTO Libro( id_libro, pelicula) VALUES (3,
EMPTY_BLOB()) RETURNING pelicula INTO v_blob;

v_file := BFILENAME('repositorio','imagen01.jpg');

DBMS_LOB.OPEN(v_file, DBMS_LOB.LOB_READONLY);

DBMS_LOB.LOADFROMFILE(v_blob, v_file,
DBMS_LOB.GETLENGTH(v_file));

DBMS_LOB.CLOSE(v_file);

COMMIT;

END;

/
```

Columnas de tipo BFILEs

- Los **BFILEs** almacenan información multimedia pero el contenido es almacenado físicamente en el sistema operativo; por dicha razón los **BFILEs** sólo se pueden acceder en modo lectura.
- El campo **BFILE** solo almacena el *LOB Locator* hacia una dirección donde se encuentra físicamente el contenido multimedia en el Sistema Operativo.
  - Es importante considerar en la política de backup incluir los directorios de los archivos que son referenciados en las columnas **BFILE** de la base de datos.
- Para insertar un referencia a un archivo en una columna **BFILE** se emplea la siguiente sintaxis:

```
INSERT INTO Libro(id_libro, portada)
VALUES (3, BFILENAME('repositorio',
'imagen02.jpg'));
```

Información de BFILEs

- Se pueden emplear funciones de manejo de **BFILE** para obtener información:

```
DECLARE
v_file BFILE;
v_nombre VARCHAR2(255);
v_ruta VARCHAR(255);
BEGIN
SELECT portada INTO v_file FROM Libro WHERE id_libro = 3;
DBMS_LOB.FILEGETNAME(v_file, v_ruta, v_nombre);
DBMS_OUTPUT.PUT_LINE('Longitud: ' ||
DBMS_LOB.GETLENGTH(v_file) || ' bytes, en la ruta ' ||
v_ruta || '/' || v_archivo);
END;
/
```

TIPOS ANY

Oracle permite crear variables y columnas que pueden almacenar datos de cualquier tipo, permitiendo comprobar el valor de ese dato en cualquier momento

Empleando este tipo de datos, una misma columna puede almacenar en una fila un valor numérico, en otra fila una cadena de caracteres y en otra un objeto

Existen tres tipos de datos que permiten el tipado dinámico:

- **SYS.ANYDATA**, que almacena un valor de cualquier tipo escalar o tipo de objeto,
- **SYS.ANYDATASET**, que almacena valores de cualquier tipo colección, y
- **SYS.ANYTYPE** que permite manipular y comprobar información de tipos.

Uso de ANY

```
CREATE TABLE tabla
(un_valor sys.AnyData);
INSERT INTO tabla
VALUES (SYS.AnyData.ConvertNumber (502));
INSERT INTO tabla
VALUES (SYS.AnyData.ConvertVarchar ('datos'));
INSERT INTO tabla
VALUES (SYS.AnyData.ConvertDate (SYSDATE));
CREATE TYPE tipoEmpleado AS OBJECT(
numE INTEGER,
nombre VARCHAR (20))
/
INSERT INTO tabla
VALUES (sys.AnyData.ConvertObject(tipoEmpleado
(5555, 'Pepe')));
```

```
SELECT ID,
       (CASE SYS.ANYDATA.getTypeName(un_valor)
          WHEN 'SYS.VARCHAR2' THEN
SYS.ANYDATA.accessVarchar2(un_valor)
          WHEN 'SYS.NUMBER'   THEN
TO_CHAR(SYS.ANYDATA.accessNumber(un_valor))
          WHEN 'SYS.DATE'     THEN
TO_CHAR(SYS.ANYDATA.accessDate(un_valor), 'DD-MON-YYYY')
        END) AS CONTENT
FROM   tabla;
```

```
        ID CONTENT
---------- --------------------
         1 502
         2 datos
         3 01-JAN-2021
```

Restricciones

Es muy importante tener en cuenta que Oracle no soporta la herencia de tablas; es decir, la definición de jerarquías de tablas sobre tipos que están integrados en una jerarquía de tipos.

Oracle sólo permite asegurar que los atributos y métodos del supertipo de la tabla padre se heredarán en las tablas definidas sobre los subtipos.

Sin embargo, las restricciones, disparadores, etc. definidos para una tabla no podrán ser heredados por otras tablas, aunque sus tipos subyacentes pertenezcan a la jerarquía.

Ejemplo

- En este ejemplo, es necesario definir en cada tabla sus propias restricciones, ya que éstas no se propagan:

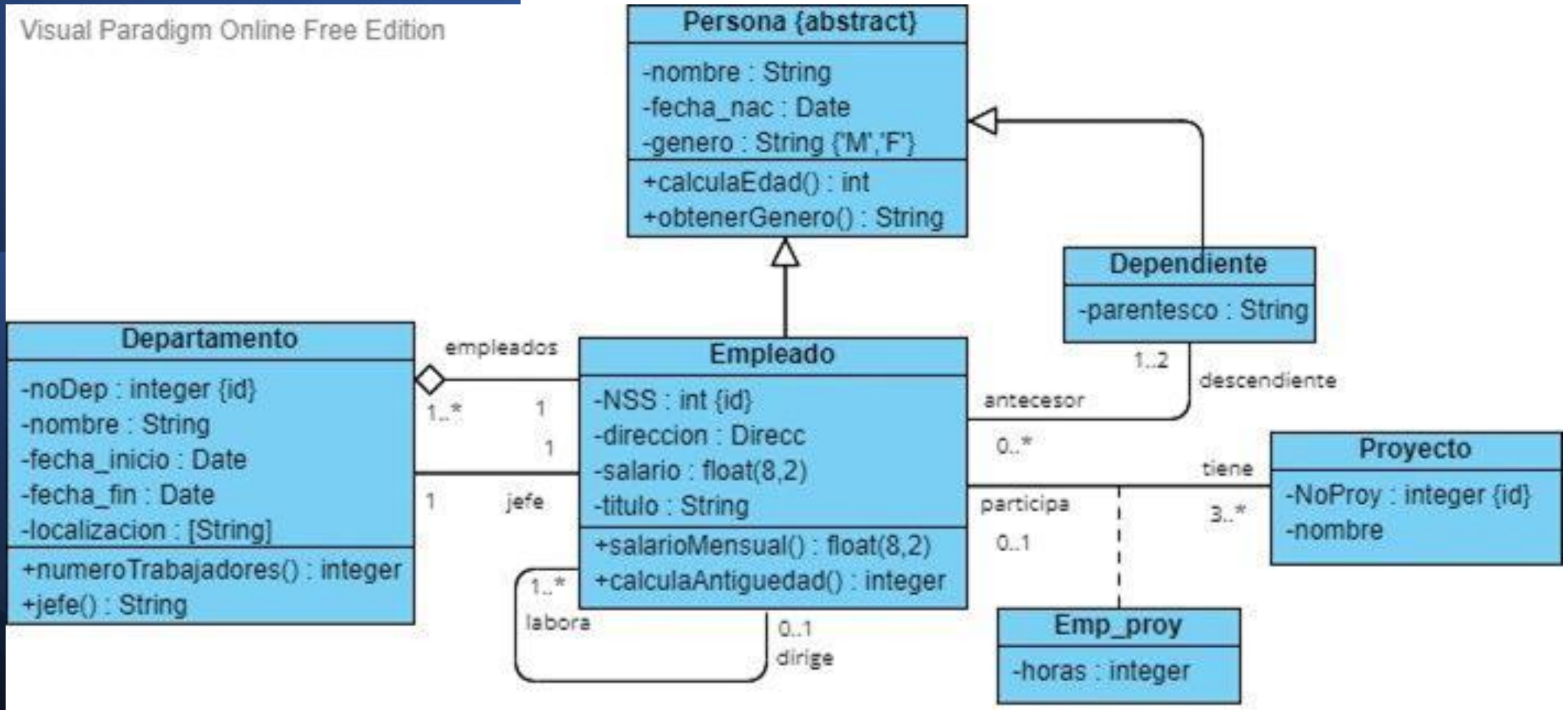
```
CREATE TABLE Persona OF tipoPersona
(PRIMARY KEY (DNI),
CHECK (direccion like ('%CDMX%')));
CREATE TABLE Estudiante OF
tipoEstudiante
(PRIMARY KEY (nombre));
CREATE TABLE Empleado OF tipoEmpleado
(PRIMARY KEY (DNI));
```

Falta de jerarquías de tablas

Pero, además de que las tablas no hereden las restricciones de la tabla definida para el supertipo, existe un problema aún mayor, y es que con el ejemplo anterior, no podemos recoger el hecho, implícito en toda jerarquía, de que todo estudiante es persona y de que todo empleado es persona.

Esto es debido a que la parte extensional de la jerarquía se representa mediante las tablas, y si no existe relación alguna entre ellas, no hay nada que le indique al SGBD que un empleado o un alumno es también una persona. Por ello, al realizar una consulta a persona, obtendríamos sólo aquellas personas que no fueran ni estudiantes ni empleados

De este modo, para la implementación de una jerarquía en Oracle, aunque podamos apoyarnos en ocasiones en la utilización de la herencia de tipos, necesitamos además recurrir a los clásicos mecanismos empleados en relacional (claves foráneas, o referencias, entre las tablas, restricciones, vistas, etc.)


Visual Paradigm Online Free Edition
Persona {abstract}
-nombre : String
-fecha_nac : Date
-genero : String {'M','F'}
+calculaEdad() : int
+obtenerGenero() : String
Dependiente
-parentesco : String
Departamento
-noDep : integer {id}
-nombre : String
-fecha_inicio : Date
-fecha_fin : Date
-localizacion : [String]
+numeroTrabajadores() : integer
+jefe() : String
empleados
1..*
1
1
1
jefe
Empleado
-NSS : int {id}
-direccion : Direcc
-salario : float(8,2)
-titulo : String
+salarioMensual() : float(8,2)
+calculaAntiguedad() : integer
1..2
descendiente
antecesor
0..*
tiene
3..*
Proyecto
-NoProy : integer {id}
-nombre
participa
0..1
1..*
labora
0..1
dirige
Emp_proy
-horas : integer