Proyecto Final

Facultad de Ciencias Exactas y Naturales, Universidad Nacional de Costa Rica

Escuela de Informática

Bases de datos

Estudiantes

Manuel Mora Sandi

Víctor Quesada Rodríguez

Nicole Vivas Montero

Santiago Solís Bolaños

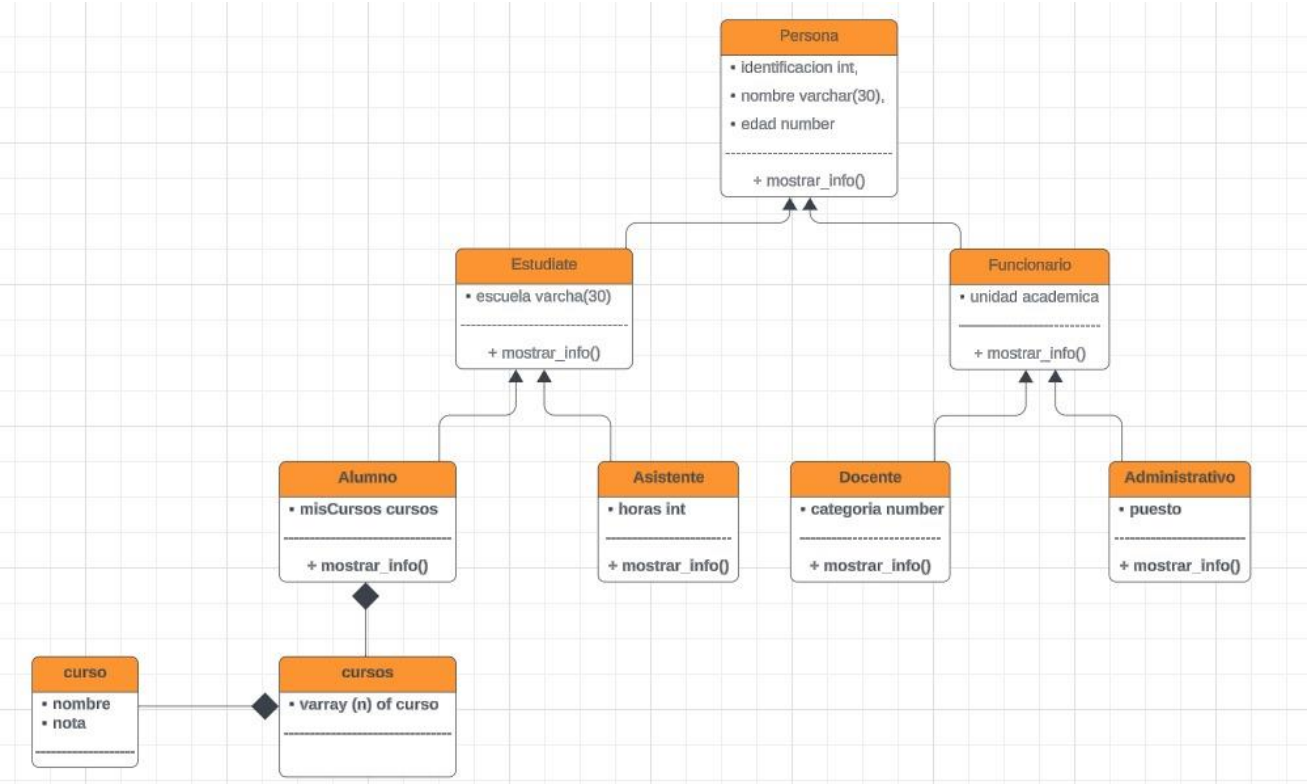
Profesor

Msc. Johnny Villalobos Murillo

I Ciclo 2024

20 de junio de 2024

Para este proyecto se diseñó e implementó un modelo de base de datos completamente orientados a objetos, se crearon las clases Alumno, Cursos, Curso, Estudiante, Asistente, Persona, Funcionario, Docente y Administrativo. A continuación, se muestre el diagrama **UML** que se generó para este modelo:



Como se muestra en el diagrama, hay una relación de composición entre cursos y curso, y una más entre alumno y cursos. También hay varias relaciones de generalización (herencia) las cuales son: estudiante y funcionario heredan de la clase persona, alumno y asistente heredan de estudiante y docente y administrativo heredan de funcionario, gracias a esto, se cumple que deben existir al menos dos clases heredadas. Para tener esta relación en Oracle se utilizó el siguiente código:

```
------------------Objeto persona----------------
CREATE OR REPLACE TYPE persona AS OBJECT (
    identificacion INT,
    nombre VARCHAR2(30),
    edad NUMBER,
    MEMBER PROCEDURE mostrar_info,
    CONSTRUCTOR FUNCTION persona RETURN SELF AS RESULT,
    CONSTRUCTOR FUNCTION persona (p_id INT, p_nombre VARCHAR2, p_edad NUMBER) RETURN SELF AS RESULT
) NOT FINAL;
/
```

Al marcar un tipo de objeto como **NOT FINAL**, se permite que otros tipos de objetos hereden de él.

```
CREATE OR REPLACE TYPE estudiante UNDER persona (
    escuela VARCHAR2(30),
    OVERRIDING MEMBER PROCEDURE mostrar_info,
    CONSTRUCTOR FUNCTION estudiante RETURN SELF AS RESULT,
    CONSTRUCTOR FUNCTION estudiante(p_id INT, p_nombre VARCHAR2, p_edad NUMBER, p_escuela VARCHAR2) RETURN SELF AS RESULT
) NOT FINAL;
/
```

Se utiliza el **UNDER** para especificar de qué clase se está heredando. Se hace lo mismo con las demás herencias. **(Debe existir al menos dos clases heredadas)** por ejemplo alumno que hereda de estudiante y estudiante que hereda de persona esto es nuestra implementación.

Otro de los objetivos del proyecto es que **cada clase debe tener al menos dos constructores**, para lograr esto se crearon los constructores **con parámetros** y **sin parámetros** para cada una de las clases:

Primero se definen los constructores dentro del objeto.

```
------------------Objeto estudiante----------------
CREATE OR REPLACE TYPE estudiante UNDER persona (
    escuela VARCHAR2(30),
    OVERRIDING MEMBER PROCEDURE mostrar_info,
    CONSTRUCTOR FUNCTION estudiante RETURN SELF AS RESULT,
    CONSTRUCTOR FUNCTION estudiante(p_id INT, p_nombre VARCHAR2, p_edad NUMBER, p_escuela VARCHAR2) RETURN SELF AS RESULT
) NOT FINAL;
/
```

Luego, dentro del cuerpo del objeto se desarrollan los constructores para poder inicializar los objetos.

```
    CONSTRUCTOR FUNCTION estudiante RETURN SELF AS RESULT IS
    BEGIN
        SELF.identificacion := 0;
        SELF.nombre := 'Desconocido';
        SELF.edad := 0;
        SELF.escuela := 'Desconocido'; -- Asignar un valor por defecto para escuela
        RETURN;
    END estudiante;

    CONSTRUCTOR FUNCTION estudiante(p_id INT, p_nombre VARCHAR2, p_edad NUMBER, p_escuela VARCHAR2) RETURN SELF AS RESULT IS
    BEGIN
        -- Inicializar los atributos de la clase base
        SELF.identificacion := p_id;
        SELF.nombre := p_nombre;
        SELF.edad := p_edad;
        SELF.escuela := p_escuela; -- Asignar el valor de p_escuela al atributo escuela
        RETURN;
    END estudiante;
END;
```

También en este modelo se logra que cada clase que heredó, deba poder **sobrescribir el método mostrar_info** de la clase que hereda utilizando el overriding member procedure:

```
------------------Objeto Alumno----------------
CREATE OR REPLACE TYPE alumno UNDER estudiante (
    misCursos cursos,
    OVERRIDING MEMBER PROCEDURE mostrar_info,
    CONSTRUCTOR FUNCTION alumno RETURN SELF AS RESULT,
    CONSTRUCTOR FUNCTION alumno (p_identificacion INT, p_nombre VARCHAR2, p_edad NUMBER, p_escuela VARCHAR2, p_cursos cursos) RETURN SELF AS RESULT
);
/
```

Su implementación es la siguiente por ejemplo en la entidad alumno:

```
------------------Cuerpo de Alumno----------------
CREATE OR REPLACE TYPE BODY alumno AS
    OVERRIDING MEMBER PROCEDURE mostrar_info IS
    BEGIN
        DBMS_OUTPUT.PUT_LINE('Nombre: ' || nombre || ' Edad: ' || edad || ' Identificacion: ' || identificacion||' escuela: ' || escue
        FOR i IN 1..misCursos.COUNT LOOP
            DBMS_OUTPUT.PUT_LINE('Curso: ' || misCursos(i).nombre || ', Nota: ' || misCursos(i).nota);
        END LOOP;
    END mostrar_info;
```

Cada clase que heredó, debe poder sobrescribir un método de la clase que hereda, este punto se abarco con el método de **mostrar_info** que se sobrescribe y se ajusta a las necesidades de cada objeto mostrando la información del padre y los nuevo atributos de del objeto.

Todas las clases incluidas deberán estar relacionadas con otra clase por relaciones tales como: (asociación, agregación, generalización, dependencia, composición..). En nuestra implementación la mayoría de las clases presentan la relación de **dependencia** e incluso se observa la relación de **composición**.