Documentación Empresa Web

TABD

CARLOS BELTRÁN ROMERO
CESAR CERROLAZA SALAS

Documentación sobre la implementacion y uso de la base de datos de una empresa destinada a desarrolo web.

Curso 22/23

Resumen

Este proyecto esta dirigido a usar una base de datos que agrupa Empleados, Equipos, Clientes, y Proyectos para da utilidad a una empresa que almacene estos datos correspondientes. La implementación web del administrador podrá tanto insertar como eliminar y observar distintos datos de la base de datos.

La base de datos tiene como objetivo servir a una empresa que se encarga de producir paginas web a sus clientes, así como de almacenar proyectos ya finalizados.

Índice

1. Requisitos funcionales y no funcionales

La base de datos cuenta con una gran cantidad de funciones que nos permiten hacer multitud de acciones. Los requisitos funcionales de la base de datos para los **Empleados** son los siguientes:

- Nombre del equipo del empleado: Devuelve el nombre del equipo para el que trabaja un empleado dado su ID.
- Mostrar los proyectos en activo del empleado: Se muestran los proyectos en los que trabaja actualmente un empleado dado su ID.
- Mostrar los proyectos finalizados del empleado: A partir de su ID, la base de datos es capaz de devolver aquellos proyectos finalizados en los que trabajó el empleado.
- Datos del empleado: A partir de su ID, se muestra el nombre, apellidos y teléfono del cliente.
- Cambio de equipo: Un empleado podrá cambiar de equipo dando su ID y el ID del nuevo equipo.
- Eliminar empleado: Se podrá eliminar un empleado de la base de datos y de todos los equipos a los que pertenecía dando su ID.

Para los requisitos funcionales de **Proyectos** tenemos lo siguiente:

- Ver estado del proyecto: Devuelve el estado del proyecto dado su ID.Es decir devuelve si está completo, y por el contrario, sus tareas finalizadas y su fecha prevista de finalización
- Cliente del proyecto: Devuelve el nombre, apellido y DNI del cliente del proyecto dado e ID del proyecto.
- Finalizar proyecto: Añade contenido a la fecha de finalización del proyecto dado su ID. Además añade a la lista de proyectos finalizados de cada empleado que participó en el mismo
- Actualizar fecha prevista: Actualiza la fecha prevista del proyecto dado su ID y la nueva fecha prevista.
- Cambiar equipo del proyecto: Cambia el equipo que trabajará en el proyecto dado su ID y el ID del nuevo equipo.
- Información del proyecto por ID: Devuelve el nombre del proyecto así como el equipo que se encarga de realizarlo.

- Finalizar tareas del proyecto: Podremos ir finalizando tareas del proyecto para ver cuanto nos queda por completar y para informar al cliente cuando podrá tener listo su proyecto.

Para los requisitos funcionales de los **Equipo** encontramos:

- Componentes del equipo: Devuelve los empleados que forman parte del equipo
- Proyectos activos del equipo: Devuelve los proyectos activos en los que participa un equipo dado su ID.
- Proyectos finalizados del equipo: Devuelve los proyectos en los que participó un equipo y ya finalizaron dado su ID.
- Equipo por ID: Devuelve el nombre del equipo dado su ID.
- Eliminar equipo: Elimina un equipo de la base de datos dado su ID.Los empleados que formaban parte de ese equipo, se quedan sin equipo a la espera de entrar en uno nuevo.
- Eliminar empleado de un equipo: Elimina un empleado de un equipo dado el ID del equipo y el ID del empleado.

Los requisitos funcionales de los **Clientes** constan de:

- Cliente por ID: Devuelve el nombre, apellido y número de teléfono de un cliente dado su DNI.
- Proyectos del cliente: Muestra los proyectos solicitados de un cliente , tanto finalizados como no finalizados dado el DNI del cliente.

Luego, la base de datos será capaz de realizar también una factura que devuelve datos como el ID de la factura, el proyecto al que pertenece, el cliente que solicitó el proyecto y la cantidad a pagar.

En cuanto a los requisitos no funcionales podríamos destacar:

- El sitio web contendrá una estructura clara, dividido en diferentes secciones según la información que deseas obtener. Los formularios de obtención o modificación de datos son intuitivos.
- Los datos están almacenados de forma segura en una base de datos Oracle mediante requerimiento de usuario y contraseña

2. Diagrama de clases persistente

La estructura de clases de nuestra base de datos incluirá diferentes clases de datos, incluyendo entidades como los los clientes, los empleados, los equipos a los que pertene-

cen estos, los proyectos llevados a cabo por los equipos para los clientes, las tareas de las que se componen estos proyectos y los pagos. Entre estas clases habrá diferentes tipos de relaciones que tienen lugar entre las entidades, como son la composición, la agregación, la asociación (de diferentes cardinalidades y multiplicidad) y la asociación con clase de enlace. Cada una de las clases posee métodos propios para cumplir los requisitos funcionales y no funcionales de la aplicación

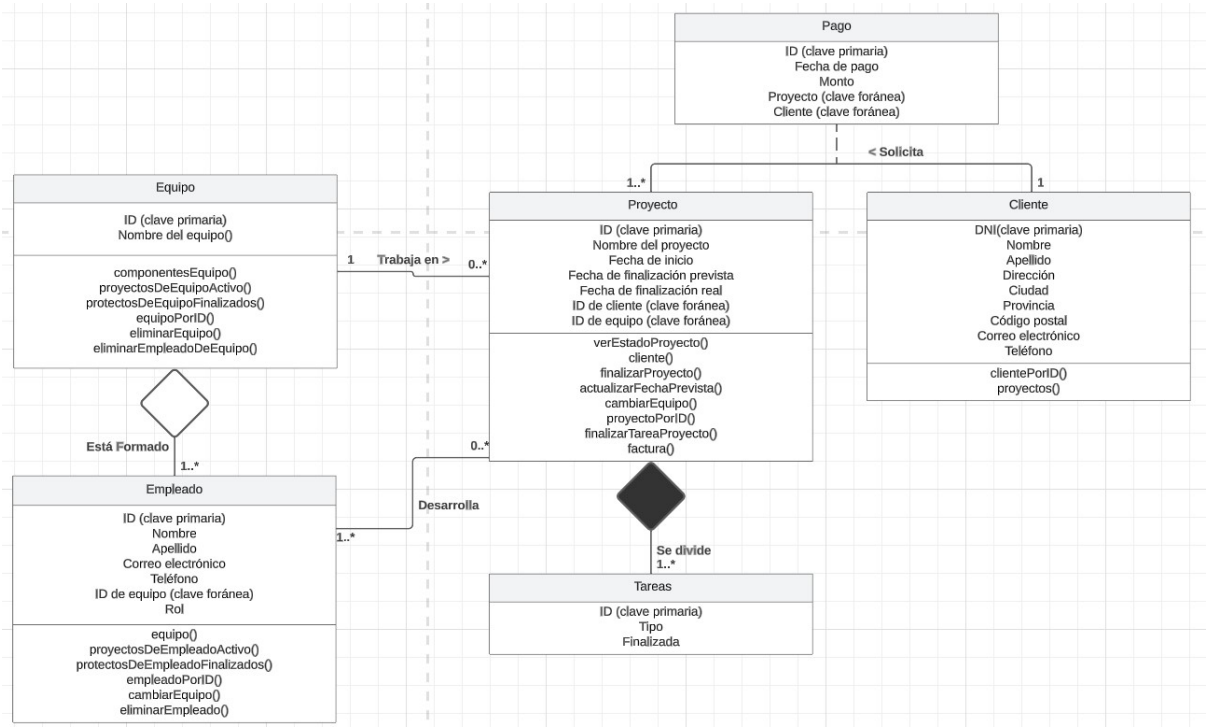

Figura 1: Diagrama de Clases Persistente

3. Esquema lógico específico O/R

Tras hacer el diagrama de clases persistentes, elaboramos más las relaciones entre entidades, creando un esquema lógico más primarias, foráneas, adicionales, referencias, campos no nulos, etc.) y las relaciones toman forma usándose listas, tanto de referencias en caso de las asociaciones y agregación, como de objetos en el caso de la composición. En el caso de los métodos, en este esquema podemos ver a mayor detalle los tipos de retorno de las funciones o si son procedimientos, y el número y tipo de parámetros que reciben.

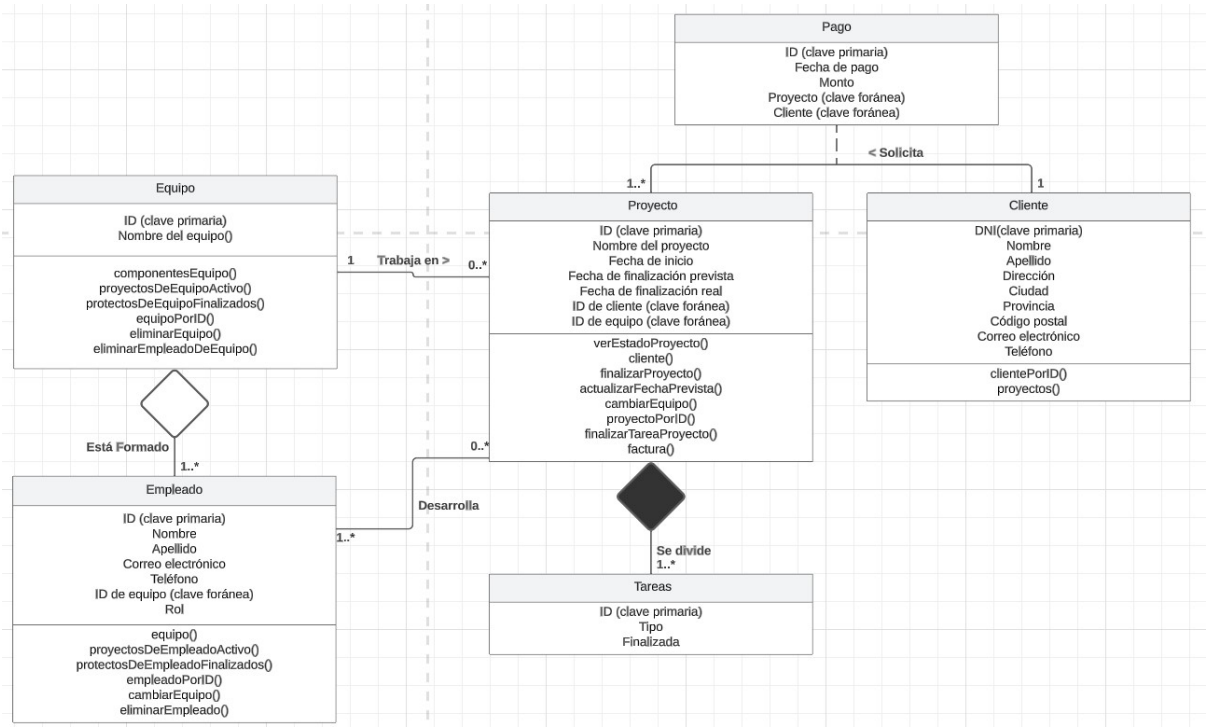

Figura 2: Esquema lógico específico O/R

4. Diseño físico

El diseño físico constara de los tipos, tablas, secuencias y triggers usados durante la realización de la base de datos.

5. Tipos

```
------------------------CREACION DE TIPOS------------------------

CREATE OR REPLACE TYPE Tipo_Tarea AS OBJECT (
    ID NUMBER(10),
    Tipo VARCHAR2(50),
    Finalizada NUMBER(1)
);
/

CREATE OR REPLACE TYPE ListaTareas AS TABLE OF Tipo_Tarea;
/

CREATE OR REPLACE TYPE Tipo_Cliente AS OBJECT (
    DNI VARCHAR2(10),
    Nombre VARCHAR2(50),
    Apellido VARCHAR2(50),
    Direccion VARCHAR2(50),
    Ciudad VARCHAR2(50),
```

```
    Provincia VARCHAR2(50),
    Codigo_Postal VARCHAR2(50),
    Correo_Electronico VARCHAR2(50),
    Telefono VARCHAR2(50)
);
/

CREATE OR REPLACE TYPE Tipo_Proyecto AS OBJECT (
    ID NUMBER(10),
    Nombre VARCHAR2(50),
    Fecha_Inicio DATE,
    Fecha_Finalizacion_Prevista DATE,
    Fecha_Finalizacion_Real DATE,
    Cliente VARCHAR2(10),
    Equipo NUMBER(10),
    SeDivide ListaTareas
);
/

CREATE OR REPLACE TYPE ListaProyectos AS TABLE OF REF Tipo_Proyecto;
/

CREATE OR REPLACE TYPE ListaProyectosTabla AS TABLE OF Tipo_Proyecto;
/


CREATE OR REPLACE TYPE Tipo_Empleado AS OBJECT (
    ID NUMBER(10),
    Nombre VARCHAR2(50),
    Apellido VARCHAR2(50),
    Correo VARCHAR2(50),
    Telefono VARCHAR2(50),
    Equipo NUMBER(10),
    Rol VARCHAR2(50),
    Desarrolla ListaProyectos
);
/

CREATE OR REPLACE TYPE ListaEmpleados AS TABLE OF REF Tipo_Empleado;
/

CREATE OR REPLACE TYPE ListaEmpleadosTabla AS TABLE OF Tipo_Empleado;
/

CREATE OR REPLACE TYPE Tipo_Equipo AS OBJECT (
    ID NUMBER(10),
    Nombre VARCHAR2(50),
    TrabajaEn ListaProyectos,
    EstaFormado ListaEmpleados
);
/

CREATE OR REPLACE TYPE Tipo_Pago AS OBJECT (
    ID NUMBER(10),
    Proyecto NUMBER(10),
```

```
    Cliente VARCHAR2(10),
    Fecha_Pago DATE,
    Monto NUMBER(10)
);
/
```

Disponemos de una creación de tipos previo a la creación de tablas. En el código se está definiendo la estructura de datos a ser almacenada en la base de datos para gestionar tareas, clientes, proyectos, empleados, equipos y pagos.

Para la creación de tablas tenemos el siguiente código:

```
-----------------------------------CREACION DE TABLAS USANDO LOS
   TIPOS-----------------------------------------------------------------
CREATE TABLE Tarea OF Tipo_Tarea
(
    CONSTRAINT PK_Tarea PRIMARY KEY (ID)
);


CREATE TABLE Cliente OF Tipo_Cliente
(
    CONSTRAINT PK_Cliente PRIMARY KEY (DNI)
);


CREATE TABLE Equipo OF Tipo_Equipo
(
    CONSTRAINT PK_Equipo PRIMARY KEY (ID),
    CONSTRAINT UK_Nombre_Equipo UNIQUE (Nombre)
)
NESTED TABLE TrabajaEn STORE AS ProyectosEquipo,  --ALMACENA LOS
   PROYECTOS ACTIVOS Y FINALIZADOS DE CADA EQUIPO
NESTED TABLE EstaFormado STORE AS Empleados;


CREATE TABLE Empleado OF Tipo_Empleado
(
    CONSTRAINT PK_Empleado PRIMARY KEY (ID),
    CONSTRAINT FK_Empleado_Equipo FOREIGN KEY (Equipo) REFERENCES
       Equipo(ID)
)
NESTED TABLE Desarrolla STORE AS ProyectosFinEmpleado; --ALMACENA LOS
   PROYECTOS FINALIZADOS DE CADA EMPLEADO


CREATE TABLE Proyecto OF Tipo_Proyecto
(
    CONSTRAINT PK_Proyecto PRIMARY KEY (ID),
    CONSTRAINT FK_Proyecto_Equipo FOREIGN KEY (Equipo) REFERENCES
       Equipo(ID),
    CONSTRAINT UK_Nombre_Proyecto UNIQUE (Nombre),
    Equipo NOT NULL,
    Cliente NOT NULL
```

```
)
NESTED TABLE SeDivide STORE AS Tareas;


CREATE TABLE Pago OF Tipo_Pago
(
    CONSTRAINT PK_Pago PRIMARY KEY (ID),
    CONSTRAINT FK_Pago_Proyecto FOREIGN KEY (Proyecto) REFERENCES
       Proyecto(ID),
    CONSTRAINT FK_Pago_Cliente FOREIGN KEY (Cliente) REFERENCES Cliente
       (DNI),
    CONSTRAINT UK_Pago_Proyecto UNIQUE (Proyecto),
    Cliente NOT NULL,
    Proyecto NOT NULL
);
```

El código crea tablas para almacenar clientes, equipos, empleados, proyectos pagos y tareas. Cada tabla tiene sus propias restricciones, como claves primarias y claves foráneas para establecer relaciones entre las tablas. Además, se están utilizando tablas anidadas para, por ejemplo, almacenar información sobre los proyectos en los que trabaja un equipo y los empleados que forman parte de un equipo.

Para las secuencias tenemos el siguiente código:

```
--PROYECTO--
CREATE SEQUENCE proyecto_seq
START WITH 1
INCREMENT BY 1
NOCACHE
NOCYCLE;

--EQUIPO--
CREATE SEQUENCE equipo_seq
START WITH 1
INCREMENT BY 1
NOCACHE
NOCYCLE;

--TAREA--
CREATE SEQUENCE tarea_seq
START WITH 1
INCREMENT BY 1
NOCACHE
NOCYCLE;
--EMPLEADO--
CREATE SEQUENCE empleado_seq
START WITH 1
INCREMENT BY 1
NOCACHE
NOCYCLE;

--PAGO--
CREATE SEQUENCE pago_seq
```

```
START WITH 1
INCREMENT BY 1
NOCACHE
NOCYCLE;
```

Se están creando secuencias para proyectos, equipos, tareas, empleados y pagos. Cada secuencia comienza en 1 y se incrementa en 1 cada vez que se genera un nuevo valor. Las secuencias no están almacenadas en caché y no se reinician cuando alcanzan el valor máximo.

6. Disparadores y paquetes

En el archivo paquetes.sql encontramos los paquetes con las funciones y procedimientos necesarios para las operaciones especificadas en el esquema lógico, empaquetadas de la siguiente forma:

- Paquete Proyecto: contiene las funciones y procedimientos relativos a la tabla Proyecto que controlan los comportamientos de sus entidades, cubriendo las necesidades y requisitos funcionales.

```
--------------------------------------------PROYECTO
   ---------------------------------------------
create or replace PACKAGE PACK_PROYECTO AS
    FUNCTION verEstadoProyecto(id_proyecto IN NUMBER) RETURN VARCHAR2 ;
    FUNCTION equipo(id_proyecto IN NUMBER) RETURN VARCHAR2;-- Devuelve
       nombre del equipo
    FUNCTION cliente(id_proyecto IN NUMBER) RETURN VARCHAR2;-- Devuelve
        nombre apellido y dni del cliente
    PROCEDURE finalizarProyecto(id_proyecto IN NUMBER);-- a ades
       contenido a fecha finalizacion
    PROCEDURE actualizarFechaPrevista(id_proyecto IN NUMBER,
       fecha_prevista IN DATE);
    PROCEDURE cambiarEquipo(id_proyecto IN NUMBER, nuevo_equipo IN
       NUMBER);-- MODIFICAR ID DE EQUIPO +TRIGGER
    FUNCTION proyectoPorID(id_proyecto IN NUMBER) RETURN VARCHAR2; --
       DEVUELVE TODA LA INFO DEL PROYECTO
    PROCEDURE finalizarTareaProyecto(id_tarea IN NUMBER, id_proyecto IN
        NUMBER);--Finaliza la tarea de un proyecto
    FUNCTION factura(id_proyecto IN NUMBER) RETURN VARCHAR2;-- Devuelve
        la factura del proyecto
END PACK_PROYECTO;
/

create or replace PACKAGE BODY PACK_PROYECTO AS
   FUNCTION verEstadoProyecto(id_proyecto IN NUMBER) RETURN VARCHAR2 IS
    v_tareas_completadas NUMBER;
    v_tareas_totales NUMBER;
    v_fecha DATE;
BEGIN
    SELECT COUNT(*) INTO v_tareas_completadas FROM TABLE(SELECT p.
       SeDivide FROM Proyecto p WHERE p.ID = id_proyecto) T
```

```
        WHERE T.Finalizada = 1;


    SELECT COUNT(*) INTO v_tareas_totales FROM Tarea t WHERE t.ID IN
        (SELECT ts.ID FROM Proyecto p, TABLE(p.SeDivide) ts WHERE p.ID
           = id_proyecto);

    SELECT Fecha_Finalizacion_Real INTO v_fecha FROM Proyecto WHERE ID
       = id_proyecto;

    IF v_fecha IS NOT NULL THEN
        RETURN 'Proyecto finalizado ' || v_fecha;
    ELSE
        SELECT Fecha_Finalizacion_Prevista INTO v_fecha FROM Proyecto
           WHERE ID = id_proyecto;
        RETURN 'Proyecto en curso, fecha prevista de finalizacion: '||
            v_fecha ||' Tareas completadas: ' || v_tareas_completadas||
             ' de '  ||v_tareas_totales;
    END IF;


END verEstadoProyecto;


    FUNCTION equipo(id_proyecto IN NUMBER)
    RETURN VARCHAR2
    IS
    v_nombre_equipo VARCHAR2(30);
    BEGIN
        SELECT p.Nombre
        INTO v_nombre_equipo
        FROM Equipo e
        JOIN Proyecto p ON e.ID = p.Equipo
        WHERE p.ID = id_proyecto;
        RETURN v_nombre_equipo;
    END;




    FUNCTION cliente(id_proyecto IN NUMBER)
    RETURN VARCHAR2
    IS
    v_nombre_cliente VARCHAR2(30);
    v_apellido_cliente VARCHAR2(30);
    v_dni_cliente VARCHAR2(30);
    BEGIN
        SELECT c.Nombre, c.Apellido, c.DNI
        INTO v_nombre_cliente, v_apellido_cliente, v_dni_cliente
        FROM Cliente c
        JOIN Proyecto p ON c.DNI = p.Cliente
        WHERE p.ID = id_proyecto;
```

```
        RETURN v_nombre_cliente || ' ' || v_apellido_cliente || ' ' ||
           v_dni_cliente;


    END;

     PROCEDURE finalizarProyecto (id_proyecto IN NUMBER) IS --TRIGGER
BEGIN
    UPDATE Proyecto p
    SET p.Fecha_Finalizacion_Real = SYSDATE,
        p.Fecha_Finalizacion_Prevista = NULL
    WHERE p.ID = id_proyecto;
    UPDATE Empleado
SET Desarrolla = ListaProyectos((SELECT REF(P) FROM Proyecto P WHERE P.
   ID = id_proyecto))
WHERE ID IN (SELECT ID FROM Empleado WHERE ID IN (SELECT ID FROM TABLE(
   SELECT Equipo.TrabajaEn FROM Equipo WHERE ID = (SELECT Equipo FROM
   Proyecto WHERE ID = id_proyecto))));

END finalizarProyecto;



    PROCEDURE actualizarFechaPrevista(id_proyecto IN NUMBER,
       fecha_prevista IN DATE) IS
    BEGIN
        UPDATE Proyecto p
        SET Fecha_Finalizacion_Prevista = fecha_prevista
        WHERE p.ID = id_proyecto;
    END actualizarFechaPrevista;

   PROCEDURE cambiarEquipo(id_proyecto IN NUMBER, nuevo_equipo IN
      NUMBER) IS

        v_equipo_actual NUMBER;
        v_proyecto_ref REF Tipo_Proyecto;
        v_lista_proyectos ListaProyectos := ListaProyectos();
    BEGIN

        SELECT Equipo INTO v_equipo_actual FROM Proyecto WHERE ID =
           id_proyecto;

        IF v_equipo_actual != nuevo_equipo THEN
            SELECT REF(p) INTO v_proyecto_ref FROM Proyecto p WHERE p.
               ID = id_proyecto;
            FOR r IN (SELECT COLUMN_VALUE FROM TABLE((SELECT e.
               TrabajaEn FROM Equipo e WHERE v_proyecto_ref MEMBER OF e
               .TrabajaEn)))
            LOOP
                IF r.COLUMN_VALUE != v_proyecto_ref THEN
                    v_lista_proyectos.EXTEND;
                    v_lista_proyectos(v_lista_proyectos.COUNT) := r.
                       COLUMN_VALUE;
                END IF;
            END LOOP;
```

```
            UPDATE Equipo e SET e.TrabajaEn = v_lista_proyectos WHERE
               v_proyecto_ref MEMBER OF e.TrabajaEn;
            UPDATE Proyecto SET Equipo = nuevo_equipo WHERE ID =
               id_proyecto;
        END IF;

    END cambiarEquipo;


FUNCTION proyectoPorID(id_proyecto IN NUMBER) RETURN VARCHAR2 IS
    v_nombre_proyecto VARCHAR2(50);
    v_nombre_equipo VARCHAR2(50);
    BEGIN
        SELECT Nombre INTO v_nombre_proyecto FROM Proyecto WHERE ID =
           id_proyecto;
        SELECT Nombre INTO v_nombre_equipo FROM Equipo WHERE ID = (
           SELECT Equipo FROM Proyecto WHERE ID = id_proyecto);
        RETURN 'Nombre del proyecto: '||v_nombre_proyecto || ' Nombre
           del equipo que lo realiza ' || v_nombre_equipo;
    END proyectoPorID;



 PROCEDURE finalizarTareaProyecto(id_tarea IN NUMBER, id_proyecto IN
    NUMBER) IS
    v_tareas_totales NUMBER;
    v_tareas_completadas NUMBER;
 BEGIN

        UPDATE TABLE(SELECT p.SeDivide FROM Proyecto p WHERE p.ID =
           id_proyecto) T
        SET T.Finalizada = 1
        WHERE T.ID = id_tarea;

        SELECT COUNT(*) INTO v_tareas_totales FROM TABLE(SELECT p.
           SeDivide FROM Proyecto p WHERE p.ID = id_proyecto) T;
        SELECT COUNT(*) INTO v_tareas_completadas FROM TABLE(SELECT p.
           SeDivide FROM Proyecto p WHERE p.ID = id_proyecto) T
        WHERE T.Finalizada = 1;

        IF v_tareas_totales = v_tareas_completadas THEN
            UPDATE Proyecto p
            SET Fecha_Finalizacion_Real = SYSDATE
            WHERE p.ID = id_proyecto;
        END IF;

 END finalizarTareaProyecto;



 FUNCTION factura(id_proyecto IN NUMBER) RETURN VARCHAR2 IS
    v_factura VARCHAR2(100);
    v_proyecto VARCHAR2(100);
    v_cliente VARCHAR2(100);
    v_fecha DATE;
```

```
    v_monto NUMBER;

  BEGIN
    SELECT p.ID, p.Proyecto, p.Cliente, p.Monto INTO v_factura,
       v_proyecto, v_cliente, v_monto
    FROM Pago p

    WHERE p.Proyecto = id_proyecto;
    RETURN ' ID Factura: ' || v_factura || ' ID Proyecto: ' ||
       v_proyecto || ' DNI  Cliente: ' || v_cliente || ' Monto: ' ||
       v_monto;

  END factura;


END PACK_PROYECTO;
/
```

- Paquete Empleado: contiene las funciones y procedimientos relativos a la tabla Empleado que controlan los comportamientos de sus entidades, cubriendo las necesidades y requisitos funcionales.

```
    ----------------------------------------EMPLEADO
       ---------------------------------------------
create or replace PACKAGE PACK_EMPLEADO AS


 -- Funci n que devuelve el nombre  equipo al que pertenece un
    empleado
  FUNCTION equipo(id_empleado NUMBER) RETURN VARCHAR2;
   -- Funci n que devuelve los proyectos en los que participa un
      empleado activo
  FUNCTION proyectosDeEmpleadoActivo(id_empleado NUMBER) RETURN
     SYS_REFCURSOR;
   -- Funci n busca en ListaProyectos los proyectos finalizados en los
       que participa un empleado
  FUNCTION proyectosDeEmpleadoFinalizados(id_empleado NUMBER) RETURN
     SYS_REFCURSOR;
   -- Funci n que devuelve el nombre,apelllido y telefono de empleado
      correspondiente a un ID dado
  FUNCTION empleadoPorID(id_empleado NUMBER) RETURN VARCHAR2;
   -- Procedimiento que cambia el equipo al que pertenece un empleado
  PROCEDURE cambiarEquipo(id_empleado NUMBER, nuevo_equipo NUMBER);
    -- Procedimiento que elimina un empleado de la base de datos
  PROCEDURE eliminarEmpleado(id_empleado NUMBER);

END PACK_EMPLEADO;
/

create or replace PACKAGE BODY PACK_EMPLEADO AS


  FUNCTION equipo(id_empleado NUMBER) RETURN VARCHAR2 IS
    v_nombre_equipo VARCHAR2(50);
```

```
  BEGIN
    SELECT e.Nombre
    INTO v_nombre_equipo
    FROM Equipo e
    JOIN Empleado em ON e.ID = em.Equipo
    WHERE em.ID = id_empleado;
    RETURN v_nombre_equipo;

  END;


  FUNCTION proyectosDeEmpleadoActivo(id_empleado NUMBER) RETURN
     SYS_REFCURSOR IS

      c_proyectos SYS_REFCURSOR;
  BEGIN
      OPEN c_proyectos FOR
          SELECT p.Nombre
          FROM Proyecto p
          JOIN Equipo e ON p.Equipo = e.ID
          JOIN Empleado em ON e.ID = em.Equipo
          WHERE em.ID = id_empleado AND p.Fecha_Finalizacion_Real IS
             NULL;
      RETURN c_proyectos;
  END;


  FUNCTION proyectosDeEmpleadoFinalizados(id_empleado NUMBER) RETURN
     SYS_REFCURSOR IS
      l_cursor SYS_REFCURSOR;
  BEGIN
      OPEN l_cursor FOR
          SELECT tp.Nombre
          FROM Empleado e,
              TABLE(e.Desarrolla) d,
              Proyecto tp
          WHERE e.ID = id_empleado AND
                d.COLUMN_VALUE = REF(tp);
      RETURN l_cursor;
  END;


  FUNCTION empleadoPorID(id_empleado NUMBER) RETURN VARCHAR2 IS
    v_nombre Empleado.Nombre%TYPE;
    v_apellido Empleado.Apellido%TYPE;
    v_telefono Empleado.Telefono%TYPE;
  BEGIN
    SELECT Nombre, Apellido, Telefono INTO v_nombre, v_apellido,
       v_telefono FROM Empleado WHERE ID = id_empleado;
    RETURN v_nombre || ' ' || v_apellido || ' ' || v_telefono;

  END empleadoPorID;

 PROCEDURE cambiarEquipo(id_empleado NUMBER, nuevo_equipo NUMBER) IS --
     +TRG_MOD_LISTA_EMPLEADOS
```

```
    employee_ref REF Tipo_Empleado;
    new_list ListaEmpleados := ListaEmpleados();
    v_equipo_actual NUMBER;
  BEGIN
    SELECT Equipo INTO v_equipo_actual FROM Empleado WHERE ID =
       id_empleado;

    IF v_equipo_actual != nuevo_equipo THEN
        SELECT REF(e) INTO employee_ref FROM empleado e WHERE e.ID =
           id_empleado;
        FOR r IN (SELECT COLUMN_VALUE FROM TABLE(SELECT e.EstaFormado
           FROM Equipo e WHERE employee_ref MEMBER OF e.EstaFormado))
        LOOP
            IF r.COLUMN_VALUE != employee_ref THEN
                new_list.EXTEND;
                new_list(new_list.COUNT) := r.COLUMN_VALUE;
            END IF;
        END LOOP;
        UPDATE Equipo e SET e.EstaFormado = new_list WHERE employee_ref
            MEMBER OF e.EstaFormado;
        UPDATE Empleado SET Equipo = nuevo_equipo WHERE ID =
           id_empleado;
    END IF;
  END cambiarEquipo;


  PROCEDURE eliminarEmpleado(id_empleado NUMBER) IS  --+
     TRG_ELIMINAR_EMPLEADO
  BEGIN
    DELETE FROM TABLE(SELECT e.EstaFormado FROM Equipo e
        WHERE e.ID = (SELECT Equipo FROM Empleado WHERE ID =
           id_empleado))
      WHERE COLUMN_VALUE = (SELECT REF(e) FROM Empleado e WHERE e.ID =
         id_empleado);

    DELETE FROM Empleado WHERE ID = id_empleado;

  END eliminarEmpleado;

END PACK_EMPLEADO;
/
```

- Paquete Cliente: contiene las funciones y procedimientos relativos a la tabla Cliente que controlan los comportamientos de sus entidades, cubriendo las necesidades y requisitos funcionales.

```
    ----------------------------------------CLIENTE
       ------------------------------------------
create or replace PACKAGE PACK_CLIENTE AS
  -- Funci n que devuelve el nombre apellido y numero de telefono de
     un cliente dado su DNI

  FUNCTION clientePorID(dni_cliente VARCHAR2) RETURN VARCHAR2;
```

```
  -- Procedimiento que muestra los proyectos de un cliente
  FUNCTION proyectos(dni_cliente VARCHAR2) RETURN SYS_REFCURSOR;
END PACK_CLIENTE;
/

create or replace PACKAGE BODY PACK_CLIENTE AS

  FUNCTION clientePorID(dni_cliente VARCHAR2) RETURN VARCHAR2 IS
    v_nombre VARCHAR2(50);
    v_apellido VARCHAR2(50);
    v_telefono VARCHAR2(50);
  BEGIN
    SELECT Nombre, Apellido, Telefono INTO v_nombre, v_apellido,
       v_telefono FROM Cliente WHERE DNI = dni_cliente;
    RETURN v_nombre || ' ' || v_apellido || ' ' || v_telefono;

  END clientePorID;

  FUNCTION proyectos(dni_cliente VARCHAR2) RETURN SYS_REFCURSOR IS
  v_proyectos SYS_REFCURSOR;
  BEGIN
    OPEN v_proyectos FOR
      SELECT Nombre FROM Proyecto WHERE Cliente = dni_cliente;

    RETURN v_proyectos;
  END proyectos;

END PACK_CLIENTE;
/
```

- Paquete Equipo: contiene las funciones y procedimientos relativos a la tabla Equipo que controlan los comportamientos de sus entidades, cubriendo las necesidades y requisitos funcionales.

```
----------------------------------------EQUIPO
   ---------------------------------------------
create or replace PACKAGE PACK_EQUIPO AS
    -- Funci n que devuelve los empleados que pertenecen a un equipo
    FUNCTION componentesEquipo(id_equipo NUMBER) RETURN SYS_REFCURSOR;
    -- Funci n que devuelve los proyectos en los que participa un
       equipo
    FUNCTION proyectosDeEquipoActivos(id_equipo NUMBER) RETURN
       SYS_REFCURSOR;
    -- Funci n que devuelve los proyectos en los que particip  un
       equipo y ya finalizaron
    FUNCTION proyectosDeEquipoFinalizados(id_equipo NUMBER) RETURN
       SYS_REFCURSOR;
  -- Funci n que devuelveel nombre del equipo dado un ID
  FUNCTION equipoPorID(id_equipo NUMBER) RETURN VARCHAR2;
  -- Procedimiento que elimina un equipo de la base de datos
  PROCEDURE eliminarEquipo(id_equipo NUMBER);
    -- Procedimiento que elimina un empleado de un equipo
  PROCEDURE eliminarEmpleadoDeEquipo(id_equipo NUMBER,id_empleado
     NUMBER);
```

```
END PACK_EQUIPO;
/

create or replace PACKAGE BODY PACK_EQUIPO AS

    FUNCTION componentesEquipo(id_equipo NUMBER) RETURN SYS_REFCURSOR
       IS
      l_cursor SYS_REFCURSOR;
    BEGIN
        OPEN l_cursor FOR
            SELECT e.ID, e.Nombre, e.Apellido
            FROM Empleado e
            WHERE e.Equipo = id_equipo;
        RETURN l_cursor;
    END;




    FUNCTION proyectosDeEquipoActivos(id_equipo NUMBER) RETURN
       SYS_REFCURSOR IS
      c_proyectos SYS_REFCURSOR;
    BEGIN
        OPEN c_proyectos FOR
            SELECT p.Nombre
            FROM Proyecto p
            JOIN Equipo e ON p.Equipo = e.ID
            WHERE e.ID = id_equipo AND p.Fecha_Finalizacion_Real IS
               NULL;
        RETURN c_proyectos;
    END;

    FUNCTION proyectosDeEquipoFinalizados(id_equipo NUMBER) RETURN
       SYS_REFCURSOR IS
      c_proyectos SYS_REFCURSOR;
    BEGIN
        OPEN c_proyectos FOR
            SELECT p.Nombre
            FROM Proyecto p
            JOIN Equipo e ON p.Equipo = e.ID
            WHERE e.ID = id_equipo AND p.Fecha_Finalizacion_Real IS NOT
                NULL;
        RETURN c_proyectos;
    END;

  FUNCTION equipoPorID(id_equipo NUMBER) RETURN VARCHAR2 IS
    v_nombre VARCHAR2(50);
  BEGIN
    SELECT Nombre INTO v_nombre FROM Equipo WHERE ID = id_equipo;
    RETURN v_nombre;

  END equipoPorID;

  PROCEDURE eliminarEquipo(id_equipo NUMBER) IS
  BEGIN
```

```
    DELETE FROM Equipo WHERE ID = id_equipo;
  END eliminarEquipo;

    PROCEDURE eliminarEmpleadoDeEquipo(id_equipo NUMBER,id_empleado
       NUMBER) IS
    BEGIN
      DELETE FROM TABLE(
        SELECT TrabajaEn FROM Equipo WHERE ID = id_equipo) t
      WHERE t.COLUMN_VALUE.ID = id_empleado;

      UPDATE Empleado SET Equipo = NULL WHERE ID = id_empleado;
    END eliminarEmpleadoDeEquipo;
END PACK_EQUIPO;
/
```

En el archivo disparadores.sql encontramos los diferentes disparadores que contendrá nuestra aplicación: 5 trigger de los cuales 4 tienen como finalidad controlar errores al modificar tablas, impidiendo las modificaciones erróneas, y un trigger para gestionas eliminaciones de equipos. Algunos de estos disparadores corresponden directamente a algunas funciones de los paquetes incluidos, como es el caso de Eliminar Equipo o Modificar Fecha Proyecto. Los otros 3 trigger no corresponden a ninguna función existente, pero las hemos incluido para modificaciones directas en la base de datos, sin usar la interfaz de la página.

- Disparador ELIMINAR EQUIPO: Se dispara con la eliminación de un equipo y se encarga de cambiar de equipo a los proyectos en los que trabajaba este.

```
-- TRIGGER QUE SE DISPARA AL ELIMINAR UN EQUIPO. MODIFIQUA EL ID DE
   EQUIPO DE TODOS LOS EMPLEADOS DE ESE EQUIPO Y DE LOS PROYECTOS EN
   LOS QUE ESTE

CREATE OR REPLACE TRIGGER TRG_ELIMINAR_EQUIPO
AFTER DELETE ON Equipo
FOR EACH ROW
DECLARE
    max_id_equipo NUMBER(10);
BEGIN
  SELECT MAX(Equipo) INTO max_id_equipo FROM Empleado;
  IF(:OLD.ID = max_id_equipo) THEN
    UPDATE Proyecto SET Equipo =
    (SELECT MAX(Equipo) FROM Empleado WHERE Equipo < max_id_equipo)
    WHERE Equipo = :OLD.ID;

    UPDATE Empleado SET Equipo =
    (SELECT MAX(Equipo) FROM Empleado WHERE Equipo < max_id_equipo)
    WHERE Equipo = :OLD.ID;

  ELSE
    UPDATE Proyecto SET Equipo = max_id_equipo WHERE Equipo = :OLD.ID;
    UPDATE Empleado SET Equipo = max_id_equipo WHERE Equipo = :OLD.ID;
  END IF;
END;
/
```

- Disparador ELIMINAR CLIENTE: Se dispara con la eliminación de un cliente y comprueba que el cliente no tenga proyectos asignados. Si tiene algún proyecto se lanzaría una excepción con un código de error de -20000 y un mensaje que indica que no se puede eliminar un cliente con proyectos asignados.

```
-- TRIGGER QUE SE DISPARA ANTES DE ELIMINAR UN CLIENTE. CUANDO SE
   DISPARA DA UN ERROR.

CREATE OR REPLACE TRIGGER TRG_ELIMINAR_CLIENTE
BEFORE DELETE ON Cliente
FOR EACH ROW
DECLARE
    num_proyectos NUMBER(10);
BEGIN
  SELECT COUNT(*) INTO num_proyectos FROM Proyecto WHERE Cliente = :OLD
     .DNI;
  IF(num_proyectos > 0) THEN
    RAISE_APPLICATION_ERROR(-20000, 'No se puede eliminar el cliente
       porque tiene proyectos asociados');
  END IF;
END;
/
```

- Disparador MODIFICAR CLIENTE PROYECTO: Se dispara antes de que se actualice la columna Cliente en la tabla Proyecto. Comprueba si el valor nuevo de Cliente existe en la tabla Cliente. Si el valor no existe, se lanza una excepción con un código de error de -20000 y un mensaje que indica que el cliente no existe.

```
CREATE OR REPLACE TRIGGER TRG_MODIFICAR_CLIENTE_PROYECTO
BEFORE UPDATE OF Cliente ON Proyecto
FOR EACH ROW
DECLARE
    v_existe NUMBER(10);
BEGIN
  SELECT COUNT(*) INTO v_existe FROM Cliente WHERE DNI = :NEW.Cliente;
  IF(v_existe = 0) THEN
    RAISE_APPLICATION_ERROR(-20000, 'El cliente no existe');
  END IF;
END;
/
```

- Disparador MODIFICAR FECHA PROYECTO: Se dispara antes de que se actualice la columna FechaFinalizacionPrevista en la tabla Proyecto. Comprueba si el valor nuevo de FechaFinalizacionPrevista es anterior a la fecha actual. Si el valor es anterior, se lanza una excepción con un código de error de -20000 y un mensaje que indica que la fecha prevista no puede ser anterior a la actual.

```
CREATE OR REPLACE TRIGGER TRG_MODIFICAR_FECHA_PROYECTO
BEFORE UPDATE OF Fecha_Finalizacion_Prevista ON Proyecto
FOR EACH ROW
DECLARE
    v_fecha_actual DATE;
BEGIN
  SELECT SYSDATE INTO v_fecha_actual FROM DUAL;
```

```
  IF(:NEW.Fecha_Finalizacion_Prevista < v_fecha_actual AND :NEW.
     Fecha_Finalizacion_Prevista IS NOT NULL) THEN
    RAISE_APPLICATION_ERROR(-20000, 'La fecha prevista no puede ser
       anterior a la actual');
  END IF;
END;
/
```

- Disparador MODIFICAR FECHA FIN PROYECTO: Se dispara antes de que se actualice la columna FechaFinalizacionReal en la tabla Proyecto. Comprueba si el valor nuevo de FechaFinalizacionReal es posterior a la fecha actual. Si el valor es posterior, se lanza una excepción con un código de error de -20000 y un mensaje que indica que la fecha de fin no puede ser posterior a la actual.

```
CREATE OR REPLACE TRIGGER TRG_MODIFICAR_FECHA_FIN_PROYECTO
BEFORE UPDATE OF Fecha_Finalizacion_Real ON Proyecto
FOR EACH ROW
DECLARE
    v_fecha_actual DATE;
BEGIN
  SELECT SYSDATE INTO v_fecha_actual FROM DUAL;
  IF(:NEW.Fecha_Finalizacion_Real > v_fecha_actual AND :NEW.
     Fecha_Finalizacion_Real IS NOT NULL) THEN
    RAISE_APPLICATION_ERROR(-20000, 'La fecha de fin no puede ser
       posterior a la actual');
  END IF;
END;
/
```

7. Conclusiones

La realización de este proyecto ha causado en nosotros un gran aprendizaje en la creación y manejo de bases de datos de Oracle, y su conexión vía php con una página web. Mentiríamos al decir que no hemos tenido momentos de mucha frustración al realizar el proyecto cuando obteníamos fallos que nos parecían indescifrables, sobre todo al principio del proyecto, con cualquier función que creásemos o al crear los tipos de datos o al ejecutar casi cualquier script en la base de datos. Por esta parte tenemos una mala experiencia que no repetiríamos, pero es muy cierto que a base de cometer estos errores y enfrentarnos a ellos múltiples veces cada vez teníamos mejor ojo para ver donde estaban los errores y solucionarlos, además de aprender muchísimas peculiaridades del funcionamiento de Oracle, las cuales estudiando únicamente la teoría o haciendo pequeñas prácticas nunca hubiésemos aprendido. Además la satisfación que queda al arreglar estos errores y funcione todo es proporcional a la frustración cuando no lo hacía. En resumen, la realización del proyecto nos ha enseñado a diseñar una base de datos y seguir los pasos necesarios para implementarla y conectarla a una página web, además de a manejarnos mucho mejor con el entorno de Oracle.

8. Referencias bibliográficas

1. Oracle Tutorial. PL/SQL Nested Tables. Oracle Tutorial, https://www.oracletutorial.com/plsql-tutorial/plsql-nested-tables/.
2. Oracle Ask TOM. Accessing nested tables elements. Oracle Ask TOM, https://asktom.oracle.com/pls/apex/asktom.search%3Ftag%3Daccessing-nested-tables-element
3. Oracle Tutorial. Oracle Sample Database. Oracle Tutorial, https://www.oracletutorial.com/getting-started/oracle-sample-database/.
4. Java2s. Delete one record in the nested table. Java2s, http://www.java2s.com/Tutorial/Oracle/0520__Collections/Deleteonerecordinthenestedtable.htm.
5. IBM. Accessing old and new column values in triggers using transition variables. IBM, https://www.ibm.com/docs/en/db2/9.7?topic=dt-accessing-old-new-column-value
6. Oracle. COLUMN-VALUE Pseudocolumn. Oracle Database SQL Language Reference, https://docs.oracle.com/en/database/oracle/oracle-database/19/sqlrf/COLUMN_VALUE-Pseudocolumn.html.
7. Sayantan Blog On Oracle. Oracle Collection in SQL. Sayantan Blog On Oracle, https://sayantanblogonoracle.in/oracle-collection-in-sql/.
8. Oracle FAQ. NESTED TABLE. Oracle FAQ, https://www.orafaq.com/wiki/NESTED_TABLE.