UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE CIENCIAS

Práctica #2: Creación y Gestión de Esquemas Básicos

PRESENTAN

Paredes Zamudio Luis Daniel
Rivera Morales David
Tapia Hernandez Carlos Alberto

PROFESOR

Erick Daniel Arroyo Martinez

ASIGNATURA

Fundamentos de Bases de Datos

20 de Agosto de 2024

Nota General: En los bloques de código insertado no se escriben palabras con acentos con la letra *ñ* ya que el paquete que se usa para la inserción del mismo no detecta los caracteres (creemos derivado de que son pocos los lenguajes de programación que existen en español y por lo mismo no hay soporte para ellos). Fuera de esto, el archivo `practica2.sql` si los contiene.

1. Crear Esquema con Tablas Relacionadas

1.1. Creacion de Tabla `departments`

```
01 |      CREATE TABLE departments (
02 |    department_id NUMBER PRIMARY KEY,
03 |    department_name VARCHAR2(50) NOT NULL
04 |  );
```

1.2. Creacion de Tabla `employees`

```
01 |      CREATE TABLE employees (
02 |    employee_id NUMBER PRIMARY KEY,
03 |    first_name VARCHAR2(50),
04 |    last_name VARCHAR2(50),
05 |    department_id NUMBER,
06 |    CONSTRAINT fk_department
07 |    FOREIGN KEY (department_id) REFERENCES departments(department_id)
08 |  );
```

1.3. Creación de Tabla `projects`

```
01 |      CREATE TABLE projects (
02 |      project_id NUMBER PRIMARY KEY,
03 |      -- Se considera como variable no nula para siempre tener un nombre para
       referirse al proyecto
04 |      project_name VARCHAR2(100) NOT NULL,
05 |      start_date DATE,
06 |      end_date DATE
07 |  );
```

1.4. Creación de Tabla `employee_projects`

Decidimos crear un campo `rol` ya que en un ámbito real tiene sentido que cada proyecto tenga roles bien definidos para los participantes del mismo.

```
01 |      -- Usar el nombre 'employee-projects' vuelve mas mnemotecnica el nombre
02 |      -- de la relacion
03 |      CREATE TABLE employee_projects (
04 |      employee_id NUMBER,
05 |      project_id NUMBER,
06 |      -- Almacenar el rol del empleado en el proyecto.
07 |      role VARCHAR2(50) DEFAULT 'Por Definir',
08 |      PRIMARY KEY (employee_id, project_id),
09 |      -- Restricciones para siempre tener a donde referenciar informacion.
10 |      CONSTRAINT fk_employee
11 |          FOREIGN KEY (employee_id)
12 |          REFERENCES employees(employee_id),
13 |      CONSTRAINT fk_project
14 |          FOREIGN KEY (project_id)
15 |          REFERENCES projects(project_id)
16 |  );
```

2. Insertar Datos en las Tablas

2.1. Inserción de Registros en `departments`

```
01 | INSERT INTO departments (department_id, department_name) VALUES (1, 'Recursos
       Humanos');
02 | INSERT INTO departments (department_id, department_name) VALUES (2, 'Finanzas');
03 | -- Insertamos el departamento IT como la 2-tupla de valores (3, 'IT')
04 | INSERT INTO departments (department_id, department_name) VALUES (3, 'IT');
```

2.2. Inserción de Registros en `employees`

```
01 | INSERT INTO employees (employee_id, first_name, last_name, department_id) VALUES (1,
       'Juan', 'Perez', 1);
02 | INSERT INTO employees (employee_id, first_name, last_name, department_id) VALUES (2,
       'Ana', 'Garcia', 2);
03 | INSERT INTO employees (employee_id, first_name, last_name, department_id) VALUES (3,
       'Luis', 'Martinez', 3);
```

2.3. Inserción de Proyectos en `projects`

```
01 | -- Los valores se agregan en 2-tuplas (numero-id, nombre).
02 | INSERT INTO projects (project_id, project_name) VALUES (1, 'Implementacion de ERP');
03 | INSERT INTO projects (project_id, project_name) VALUES (2, 'Rediseno Sitio Web');
04 | INSERT INTO projects (project_id, project_name) VALUES (3, 'Migracion a la Nube');
```

2.4. Asociacion Empleados - Proyectos en `employees_projects`

```
01 | INSERT INTO employee_projects (employee_id, project_id) VALUES (1, 1);
02 | INSERT INTO employee_projects (employee_id, project_id) VALUES (2, 2);
03 | INSERT INTO employee_projects (employee_id, project_id) VALUES (3, 3);
04 | INSERT INTO employee_projects (employee_id, project_id) VALUES (1, 2);
```

3. Manipulación de Datos

3.1. Actualización de Registros en `employees`

```
01 |     -- Al ser una actualizacion de valores, usamos UPDATE y WHERE.
02 |     -- El primero como la instruccion principal y el segundo para indicar
03 |     -- donde se realiza la accion. En este caso, en el employee_id = 2.
04 |     UPDATE employees SET department_id = 3 WHERE employee_id = 2;
```

3.2. Eliminación Projecto y Registros Relacionados en `employee_projects`

```
01 | -- Primero se debe eliminar el proyecto de 'employee-projects' y posteriormente de
02 | -- 'projects'. Anidamos un SELECT en la primera instruccion para tener identificada
03 | -- la clave foranea del proyecto en cuestion en su tabla original.
04 | DELETE FROM employee_projects WHERE project_id = (
05 |     SELECT project_id FROM projects WHERE project_name = 'Rediseno Sitio Web');
06 | DELETE FROM projects WHERE project_name = 'Rediseno Sitio Web';
```

3.3. Añadir Columna `email` a `employees`

```
01 |  -- Se ocupa el comando ALTER TABLE y despues indicamos el nombre y el tipo de
02 |  -- valor que va a ocupar la columna nueva.
03 |      ALTER TABLE employees ADD email VARCHAR2(100);
```

3.4. Insertar Correos Correspondientes en `employees`

```
01 |  UPDATE employees SET email = 'juan.perez@empresa.com' WHERE employee_id = 1;
02 |  UPDATE employees SET email = 'ana.garcia@empresa.com' WHERE employee_id = 2;
03 |  UPDATE employees SET email = 'luis.martinez@empresa.com' WHERE employee_id = 3;
```

3.5. Eliminar Tabla `employee-projects`

Justificación de la Eliminación: En un contexto real, eliminar la tabla `employee_projects` podría ser necesario si se decide cambiar la estructura de la base de datos o si se adopta un nuevo sistema de gestión que maneje las relaciones entre empleados y proyectos de manera diferente.

```
01 |      DROP TABLE employee_projects;
```

4. Consultas con Condiciones

4.1. Listar Empleados en Departamento 'IT'

```
01 |  -- Se seleccionaron todas las columnas en las que se cumpliera con el filtro de
       pertenecer al departamento de IT, que en este caso es el 3.
02 |  SELECT * FROM employees WHERE department_id = 3;
```

4.2. Encontrar Empleados Sin Asignar a Proyectos

```
01 |  -- Se hace un filtro usando un SELECT buscando el employee_id correcto.
02 |  SELECT * FROM employees WHERE employee_id NOT IN (SELECT employee_id FROM
       employee_projects);
```

4.3. Listar Departamentos con Menos de 2 Empleados

```
01 |
02 |  -- HAVING COUNT nos sirve para limitar la busqueda a menos de dos empleados.
03 |      SELECT department_id, COUNT(*) as num_employees FROM employees GROUP BY
         department_id HAVING COUNT(*) < 2;
```

5. Demostración de la Integridad Referencial

5.1. Intento de eliminación de un registro de la tabla `departments` referenciado en la tabla `employees`

```
01 |       DELETE FROM departments WHERE department_id = 1;
```

5.2. Observación del Error de Consistencia Generado y Descripción del Mismo

```
01 |      ORA-02292: integrity constraint (SCHEMA.FK_DEPARTMENT) violated - child record
         found
```

Este error ocurre debido a que con la línea anterior se está tratando de eliminar una instancia de la que dependen referencias secundarias, si esto pasara se perdería la consistencia de la base de datos.

5.3. Explicación de Posible Solución

Lo que debería hacerse es eliminar primero las dependencias del registro padre y ya que todas estas estén eliminadas podría eliminarse el registro padre. Primero sería el registro `employees` porque este depende de `departments`.

```
01 |  -- Al existir una condicion precedente en employees, eliminamos dicha condicion
         primero
02 |  ALTER TABLE employees DROP CONSTRAINT fk_department;
03 |  -- Agregamos la nueva condicion y anadimos ON DELETE CASCADE para que al querer
         eliminar un registro padre se eliminen los hijos y asi se mantenga la integridad
          referencial.
04 |  ALTER TABLE  employees
05 |  ADD  CONSTRAINT fk_department
06 |    FOREIGN KEY (department_id)
07 |          REFERENCES departments(department_id)
08 |          ON DELETE CASCADE;
```

6. Conclusiones

En esta práctica, hemos utilizado tanto operaciones DDL (Data Definition Language) como DML (Data Manipulation Language) para interactuar con la base de datos.

6.1. DDL (Data Definition Language)

- **CREATE TABLE**: Utilizamos este comando para definir la estructura de nuestras tablas (*departments*, *employees*, *projects*, *employee_projects*).
- **ALTER TABLE**: Lo usamos para modificar la estructura de una tabla existente, como agregar la columna *email* a la tabla *employees*.
- **DROP TABLE**: Empleamos este comando para eliminar la tabla *employee_projects*.
- **Constraints**: Definimos restricciones como *PRIMARY KEY* y *FOREIGN KEY* para mantener la integridad referencial.

6.2. DML (Data Manipulation Language)

- **INSERT**: Utilizamos este comando para agregar nuevos registros en nuestras tablas.
- **UPDATE**: Lo usamos para modificar datos existentes, como actualizar el departamento de un empleado o agregar correos electrónicos.
- **DELETE**: Empleamos este comando para eliminar registros, como el proyecto *Rediseño Sitio Web*.
- **SELECT**: Utilizamos consultas *SELECT* para recuperar y filtrar datos según diferentes criterios.

6.3. Integridad Referencial

- Comprendimos la importancia de la integridad referencial al intentar eliminar un departamento referenciado por empleados.
- Aprendimos cómo manejar estas situaciones utilizando la cláusula *ON DELETE CASCADE*.

Esta práctica nos permitió aplicar conceptos fundamentales de bases de datos relacionales, desde la creación de esquemas hasta la manipulación de datos y la gestión de relaciones entre tablas. También nos ayudó a comprender la importancia de mantener la integridad de los datos y cómo las restricciones y las operaciones DDL/DML trabajan juntas para lograr este objetivo.