Proyecto Final.

Fecha ENTREGA: 1/Junio/2019
Forma de entregar: subir a Campus Virtual.

NOTA: Incluir los autores del proyecto final.

Nombre: Expediente:
Michael Brandon Serrato Guerrero **258849**

SUBIR: archivo .txt con el script de creación de tablas, vistas y relaciones, y el código de los procedimientos almacenados y triggers (código PLpgSQL). Archivo Imagen del diseño lógico relacional.

Instrucciones.
Considere la siguiente descripción de un análisis de una empresa de desarrollo de proyectos y su modelo entidad-relación asociado para crear procedimientos almacenados y triggers en la **tecnología de base de datos PLpgSQL**.

Descripción del Sistema.

Una organización de desarrollo de proyectos de ingeniería va a crear un sistema de información de soporte a su administración. La organización debe gestionar los empleados que trabajan en ella, y estos son de tres tipos: empleados (empleados generales como analistas, secretarias(os), soporte técnico y guardias), gerentes e ingenieros de proyectos.

Los casos de uso son:

- Alta, Baja y Promoción de Empleados.
- Alta, Baja, Movimiento y Promoción de Ingenieros.
- Alta, Baja y Movimiento de Gerentes.
- Asignación y Baja de Ingenieros a Proyectos.
- Generar Registro Histórico de cada Baja del Sistema.

Reglas del Negocio:

Generales

- Todo personal solo puede estar desempeñando un tipo de cargo, como empleado, como gerente o como ingeniero de proyectos.
- Si es ingeniero de proyectos también es un empleado general pero se le ve como ingeniero.
- Si es gerente también es un empleado general pero se le ve como gerente.

Contrataciones

- Es posible dar de alta a una *persona* directamente como *empleado* general.
- Una persona que no sea *empleado* puede ser *gerente* directamente o *ingeniero* de proyectos, sin haber sido *empleado*.

Promociones Laborales

- Si un *empleado* asciende a *gerente* y pasa a ser *gerente*, porque ya es empleado.
- Si un *empleado* asciende a *ingeniero* de proyectos y pasa a ser a *ingeniero*, porque ya es empleado.
- Un *ingeniero* de proyectos pasa a ser *gerente*:

    1. El empleado ingeniero conserva su clave de empleado.
    2. Guardar todos los datos del ingeniero en ***histórico de ingenieros***;
    3. Guardar todas sus asignaciones en proyectos en el archivo de ***histórico proyectos-ingenieros***;
    4. Borrar dichas asignaciones en proyectos;
    5. Borrar los datos como ingeniero, y queda solo como empleado;
    6. Promover al empleado como *gerente* creando la instancia necesaria junto con sus atributos de fecha de inicio de gerente y bono económico.

- Un *gerente* pasar a ser *ingeniero* de proyectos:

1. El empleado gerente conserva su clave de empleado.
2. Guardar el registro del gerente en ***histórico de gerentes***;
3. Liberar el vehículo que tuviera asignado, si es el caso;
4. Borrar los datos como gerente, y queda solo como empleado;
5. Promover al empleado como *ingeniero* creando la instancia necesaria junto con sus atributos de especialidad y categoría.

Movimientos Laborales

- Un *gerente* o ingeniero al dejar su puesto pueden pasar a ser solo empleados:

1. El empleado gerente o ingeniero conserva su clave de empleado.
2. Si es gerente: guardar el registro del gerente en ***histórico de gerentes***; liberar el vehículo que tuviera asignado, si es el caso; borrar los datos como gerente, y queda solo como empleado.

a. Si es ingeniero: guardar todos los datos del ingeniero en ***histórico de ingenieros***; guardar todas sus asignaciones en proyectos en el archivo de ***histórico proyectos-ingenieros***; borrar dichas asignaciones en proyectos; borrar los datos como ingeniero, y queda solo como empleado;

- Al dar de baja a un *empleado* general del sistema simplemente se da de baja y se salva su registro en el ***histórico de empleados***.
- Al dar de baja a un *ingeniero* del sistema, primero pasarlo a cargo de *empleado* y proceder a la baja como *empleado*.
- Al dar de baja a un *gerente* del sistema, primero pasarlo a cargo de *empleado* y proceder a la baja como *empleado*.

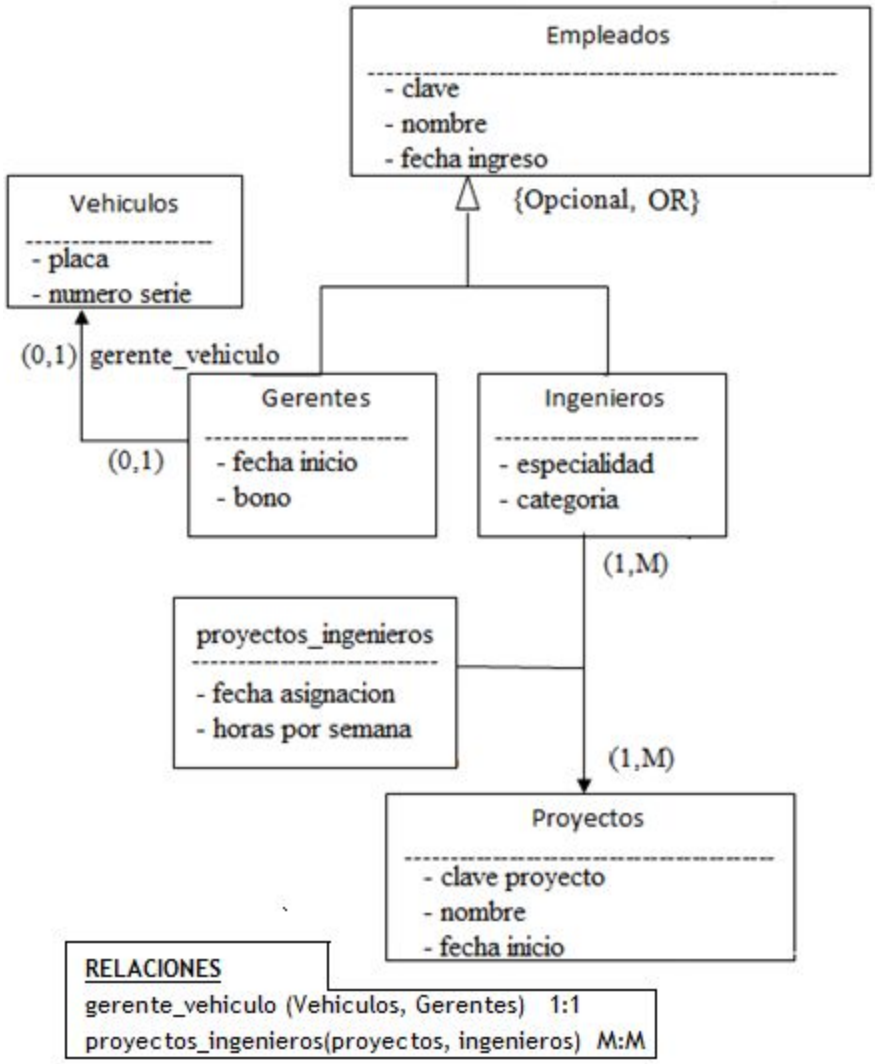
Gestión de Vehículos

- Al momento de dar de alta al *gerente* éste puede no tener *vehículo* asignado pero posteriormente se le puede otorgar un *vehículo*, pero no más de uno.
- Al dar de baja un *vehículo* asignado a algún *gerente* (porque está dañado y tardará mucho tiempo en su reparación), el *gerente* debe quedar sin *vehículo* automáticamente.

Asignar Ingenieros a Proyectos

- La asignación de *proyectos* solo se hace entre un *ingeniero* y un *proyecto*.
- No es posible permitir que un *ingeniero* de proyectos sea asignado a más de tres *proyectos*.

MODELO ENTIDAD-RELACIÓN



Opcional : Puede ser Empleado, Gerente o Ingeniero.

OR : Si es Empleado no puede ser otro tipo; Si es Gerente no puede ser otro tipo; Si es Ingeniero no puede ser otro tipo; es decir, un empleado solo puede cubrir un tipo de puesto, como empleado general, como ingeniero o como gerente.

Todos los archivos históricos deben contener:

historico_XXXXX (id_secuencial, fecha registro, datos de la Entidad);

Casos de Uso

Alta Empleado:

- Ingresar un empleado con los datos clave, nombre completo y fecha de ingreso.

clave: 100
nombre: Aracely Cuellar Salinas
fecha ingreso: 2018-01-15 [YYYY-mm-dd]

Baja Empleado:

- Eliminar un empleado por medio de su clave de empleado. Enviar a histórico de empleados.

histórico de empleados(idregistro (pk), clave, nombre, fecha ingreso, fecha de registro)
idRegistro: entero secuencial.
fecha de registro: fecha del momento de la transacción.

Alta Gerente:

- Si un empleado es ascendido a gerente sólo se ingresa la clave del empleado que es ascendido, con esto se busca su registro, se consigue su nombre y la fecha de ingreso; además proporcionar fecha de inicio de gerente, el bono en pesos mexicanos y la placa del auto que se le asignará, pero no siempre se otorga vehículo. Verificar que fecha de ingreso sea anterior a fecha de inicio de gerente. Se ingresa al sistema como gerente (ver reglas del negocio).

El bono en pesos mexicanos está en el rango de 30 mil y 40 mil pesos.

- Si es nuevo en el sistema ingresar clave, nombre completo, fecha de ingreso, fecha inicio de gerente, bono en pesos mexicanos y la posible placa del auto; se verifica

que la fecha de ingreso sea anterior a la fecha de inicio de gerente. Se ingresa al sistema como gerente (ver reglas del negocio) pero además es empleado general.

Baja Gerente:

- Pedir la clave de gerente y leer los datos del gerente; si no existe terminar. Si existe eliminarlo de los *gerentes*. Salvar los datos del gerente en *histórico de gerentes*. El gerente pasa a ser un empleado general. Después podría eliminarse pero ya como empleado.

Alta Ingeniero:

- Si un empleado es ascendido a ingeniero sólo se ingresa la clave del empleado que es ascendido, con esto se busca su registro, se consigue su nombre y la fecha de ingreso; ingresar especialidad y categoría. Se ingresa al sistema como ingeniero (ver reglas del negocio).

- Si es nuevo en el sistema ingresar clave, nombre completo, fecha de ingreso, su especialidad y categoría. Se ingresa al sistema como ingeniero (ver reglas del negocio) pero además es empleado general.

Categoría: JUNIOR y SENIOR.
Especialidad: ELECTRÓNICA, MECATRÓNICA, MECÁNICA, INDUSTRIAL, SOFTWARE, SERVIDORES Y REDES.

Baja Ingeniero:

- Solicitar clave de ingeniero y se procede la baja. Enviar a *histórico de ingenieros* y todas sus asignaciones al *histórico de proyectos-ingenieros*.

histórico de ingenieros(idregistro (pk), fecha de registro, todos los datos del ingeniero)
idRegistro: entero secuencial.
fecha de registro: fecha del momento de la transacción.

histórico de proyectos-ingenieros (id secuencial, fecha de registro y todos los datos de la asignación)

Alta-Baja de Proyectos:

- Registrar una clave de ingeniero, una clave de proyecto, la fecha de asignación y el número de horas por semana.

Número de horas dedicadas al proyecto por semana: no exceder 40 horas.

- La suma total de las horas asignadas de algún ingeniero a los diversos proyectos no deben rebasar las 40 horas a la semana. Si el ingeniero número 210 está asignado a tres proyectos distintos, la suma de las horas asignadas no debe superar las 40 horas por semana.
- Verificar que no sea asignado a más de tres proyectos a la vez (puede estar en 1, 2 o 3 máximo).
- Al retirar a un ingeniero de un proyecto, luego de retirarlo enviar esos datos al histórico de proyectos-ingenieros. Recordar salvar los datos de la asignación más un Id secuencial y la fecha de registro.

INSTRUCCIONES PARA PROGRAMAR EL SISTEMA

Procedimientos Almacenados y TRIGGERs

1. Crear el diseño lógico de la base de datos.
   - Aplicar el diseño donde se crea una tabla por cada entidad del modelo conceptual y la entidad derivada tiene una relación de 1:1 con su tabla padre.
   - Crear una tabla empleados, porque todos son empleados;
   - Crear las tablas gerentes e ingenieros con sus atributos pero deben tener relación 1:1 con tabla empleados.
   - Crear las tablas de las demás entidades del modelo conceptual.
   - Crear el script SQL correspondiente, indicando las tablas, campos, las llaves primarias, las llaves foráneas y las relaciones entre tablas dejando bien clara la integridad referencial.
   - Adjuntar-pegar imagen con el diseño relacional.

- Adjuntar-pegar el código con todo el script SQL de la metadata de la base de datos.
- Crear una vista llamada: **v_ingenieros** para trabajar los procedimientos almacenados referentes a:

insertarEliminarIngeniero, promoverIngeniero_a_Gerente, moverIngeniero_a_Empleado.

- Crear una vista llamada: **v_gerentes** para trabajar los procedimientos almacenados referentes a:

insertarEliminarGerentes, moverGerente_a_Ingeniero, moverGerente_a_Empleado.

2. Crear un Procedimiento Almacenado para ***insertarPromoverEliminar Empleado***. Códigos de operación que debe devolver el Procedimiento Almacenado:
   - Devolver 1: Si el empleado-ingeniero no existe.
   - Devolver 9: Si hay un error de SQL (falla de la base de datos).
   - Devolver 0: Si todo se ejecutó correctamente.

3. Crear un Procedimiento Almacenado para ***InsertarMoverPromoverEliminarIngeniero***. Códigos de operación que debe devolver el Procedimiento Almacenado:

   - Devolver 1: Si el empleado no existe.
   - Devolver 9: Si hay un error de SQL (falla de la base de datos).
   - Devolver 0: Si todo se ejecutó correctamente.

4. Crear un Procedimiento Almacenado para ***InsertarMoverEliminarGerente***. Códigos de operación que debe devolver el Procedimiento Almacenado:

   - Devolver 1: Si el empleado no existe.
   - Devolver 9: Si hay un error de SQL (falla de la base de datos).
   - Devolver 0: Si todo se ejecutó correctamente.

5. Crear un Procedimiento Almacenado para ***InsertarEliminarAsignaciones_de_ Ingenieros_a_ Proyectos***. Códigos de operación que debe devolver el Procedimiento Almacenado:

   - Devolver 1: Si el empleado-ingeniero no existe.
   - Devolver 3: Si el ingeniero ya está con el máximo de proyectos: tres proyectos.
   - Devolver 9: Si hay un error de SQL (falla de la base de datos).
   - Devolver 0: Si todo se ejecutó correctamente.

6. Crear Triggers para toda Eliminación de Registros de Empleados, Ingenieros, Asignaciones de Ingenieros a Proyectos y así poder llevar los ***históricos*** que demandan los requerimientos de este documento.

7. Crear los casos de prueba para verificar la precisión de cada procedimiento almacenado y de cada trigger.

8. Para las tablas de vehículos y proyectos no crear código plpgsql ni trigger. Solo insertar los registros que se proponen para realizar los casos de prueba.

Para poder probar los diseños de tablas, históricos, vista de ingenieros, vista de gerentes, procedimientos almacenados y triggers, ingresar los siguientes datos:

Ingresar tres vehículos:

Placa	Número de Serie
AAA001	A12345
AAA002	B12345
AAA003	C12345

Ingresar cuatro proyectos:

clave de proyecto	nombre de proyecto	fecha de inicio
1	'PROY UNO'	'2019-01-02'
2	'PROY DOS'	'2019-03-01'
3	'PROY TRES'	'2019-05-01'

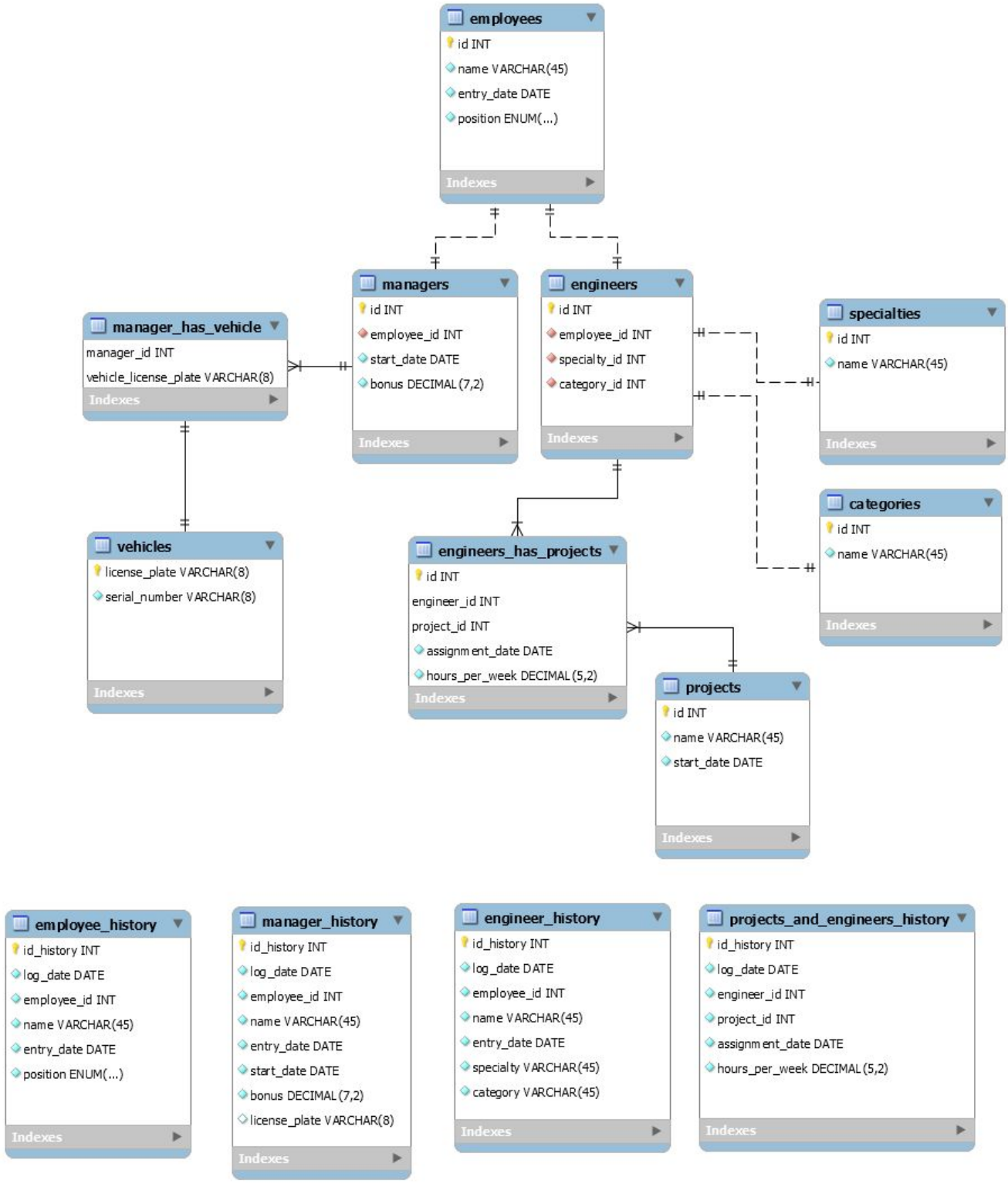
4	'PROY CUATRO'	'2019-07-01'

Especialidades de Ingenieros:

Especialidad Ingenieros
ELECTRÓNICA
MECATRÓNICA
MECÁNICA
INDUSTRIAL
SOFTWARE
SERVIDORES Y REDES

Sección de Solución del Problema

1. Pegar aquí la imagen del diseño de tablas (diseño lógico).



2. Pegar aquí el script de la base de datos (tablas, vistas, relaciones).

```
--- Script de la base de datos (tablas, vistas, relaciones).

--- Creación de la BASE DE DATOS:
CREATE DATABASE final_project_db ENCODING 'utf8';

--- Definición de TIPOS y DOMINIOS:
CREATE TYPE enumPosition AS ENUM('GENERAL', 'ENGINEER', 'MANAGER');
CREATE DOMAIN bonus DECIMAL(7,2) CHECK(VALUE >= 30000.00 AND VALUE <= 40000.00);
CREATE DOMAIN license_plate VARCHAR(8);
CREATE DOMAIN serial_number VARCHAR(8);

--- Creación de TABLAS y RELACIONES (claves foráneas):
CREATE TABLE IF NOT EXISTS employees (
 id SERIAL PRIMARY KEY,
 name VARCHAR(45) NOT NULL,
 entry_date DATE NOT NULL,
 position enumPosition NOT NULL DEFAULT 'GENERAL'
);

CREATE TABLE IF NOT EXISTS managers (
 id SERIAL PRIMARY KEY,
 employee_id INT NOT NULL,
 start_date DATE NOT NULL,
 bonus bonus NOT NULL
);

ALTER TABLE managers
   ADD CONSTRAINT fk_managers_employees
   FOREIGN KEY (employee_id)
   REFERENCES employees (id);

CREATE TABLE IF NOT EXISTS vehicles (
 license_plate license_plate NOT NULL PRIMARY KEY,
```

```
 serial_number serial_number NOT NULL
);

CREATE TABLE IF NOT EXISTS manager_has_vehicle (
 manager_id INT NOT NULL,
 vehicle_license_plate license_plate NOT NULL
);

ALTER TABLE manager_has_vehicle
   ADD CONSTRAINT fk_manager_has_vehicle_manager
   FOREIGN KEY (manager_id)
   REFERENCES managers (id)
   ON DELETE CASCADE;

ALTER TABLE manager_has_vehicle
   ADD CONSTRAINT fk_manager_has_vehicle_vehicle
   FOREIGN KEY (vehicle_license_plate)
   REFERENCES vehicles (license_plate)
   ON DELETE CASCADE;

CREATE TABLE IF NOT EXISTS specialties (
 id SERIAL PRIMARY KEY,
 name VARCHAR(45) NOT NULL
);

CREATE TABLE IF NOT EXISTS categories (
 id SERIAL PRIMARY KEY,
 name VARCHAR(45) NOT NULL
);

CREATE TABLE IF NOT EXISTS engineers (
 id SERIAL PRIMARY KEY,
 employee_id INT NOT NULL,
 specialty_id INT NOT NULL,
 category_id INT NOT NULL
);
```

```
ALTER TABLE engineers
   ADD CONSTRAINT fk_engineers_employees
   FOREIGN KEY (employee_id)
   REFERENCES employees (id);

ALTER TABLE engineers
   ADD CONSTRAINT fk_engineer_specialty
   FOREIGN KEY (specialty_id)
   REFERENCES specialties (id);

ALTER TABLE engineers
   ADD CONSTRAINT fk_engineer_category
   FOREIGN KEY (category_id)
   REFERENCES categories (id);

CREATE TABLE IF NOT EXISTS projects (
 id SERIAL PRIMARY KEY,
 name VARCHAR(45) NOT NULL,
 start_date DATE NOT NULL
);

CREATE TABLE IF NOT EXISTS engineers_has_projects (
 id SERIAL PRIMARY KEY,
 engineer_id INT NOT NULL,
 project_id INT NOT NULL,
 assignment_date DATE NOT NULL,
 hours_per_week DECIMAL(5,2) NOT NULL
);

ALTER TABLE engineers_has_projects
   ADD CONSTRAINT fk_engineers_has_projects_engineer
   FOREIGN KEY (engineer_id)
   REFERENCES engineers (id);

ALTER TABLE engineers_has_projects
```

```
   ADD CONSTRAINT fk_engineers_has_projects_project
   FOREIGN KEY (project_id)
   REFERENCES projects (id);

CREATE TABLE IF NOT EXISTS employee_history (
 id_history SERIAL PRIMARY KEY,
 log_date DATE NOT NULL,
 employee_id INT NOT NULL,
 name VARCHAR(45) NOT NULL,
 entry_date DATE NOT NULL
);

CREATE TABLE IF NOT EXISTS manager_history (
 id_history SERIAL PRIMARY KEY,
 log_date DATE NOT NULL,
 employee_id INT NOT NULL,
 name VARCHAR(45) NOT NULL,
 entry_date DATE NOT NULL,
 start_date DATE NOT NULL,
 bonus bonus NOT NULL,
 license_plate license_plate DEFAULT NULL
);

CREATE TABLE IF NOT EXISTS engineer_history (
 id_history SERIAL PRIMARY KEY,
 log_date DATE NOT NULL,
 employee_id INT NOT NULL,
 name VARCHAR(45) NOT NULL,
 entry_date DATE NOT NULL,
 specialty VARCHAR(45) NOT NULL,
 category VARCHAR(45) NOT NULL
);

CREATE TABLE IF NOT EXISTS projects_and_engineers_history (
 id_history SERIAL PRIMARY KEY,
 log_date DATE NOT NULL,
```

```
 engineer_id INT NOT NULL,
 project_id INT NOT NULL,
 assignment_date DATE NOT NULL,
 hours_per_week DECIMAL(5,2) NOT NULL
);

--- Inserción de "DATOS INICIALES" en la base de datos:
INSERT INTO vehicles (license_plate, serial_number)
VALUES  ('AAA001', 'A12345'),
       ('AAA002', 'B12345'),
       ('AAA003', 'C12345');

INSERT INTO projects (name, start_date)
VALUES  ('PROY UNO', '2019-01-02'),
        ('PROY DOS', '2019-03-01'),
        ('PROY TRES', '2019-05-01'),
        ('PROY CUATRO', '2019-07-01');

INSERT INTO specialties (name)
VALUES  ('ELECTRÓNICA'),
        ('MECATRÓNICA'),
        ('MECÁNICA'),
        ('INDUSTRIAL'),
        ('SOFTWARE'),
        ('SERVIDORES Y REDES');

INSERT INTO categories (name)
VALUES  ('JUNIOR'),
        ('SENIOR');

--- Creación de VISTAS:
CREATE VIEW v_employees AS
  SELECT id AS employee_id, name, entry_date
 FROM employees
 WHERE position = 'GENERAL'
 ORDER BY id ASC;
```

```
CREATE VIEW v_managers AS
  SELECT employee_id, name, entry_date, start_date, bonus, license_plate
  FROM managers m
  INNER JOIN employees e
    ON e.id = m.employee_id
  LEFT JOIN manager_has_vehicle m_h_v
    ON m_h_v.manager_id = m.id
  LEFT JOIN vehicles v
    ON v.license_plate = m_h_v.vehicle_license_plate
  ORDER BY employee_id ASC;

CREATE VIEW v_engineers AS
 SELECT employee_id, emp.name, entry_date, s.name AS specialty, c.name AS category
 FROM engineers eng
 INNER JOIN employees emp
   ON emp.id = eng.employee_id
 INNER JOIN specialties s
   ON s.id = eng.specialty_id
 INNER JOIN categories c
   ON c.id = eng.category_id
 ORDER BY employee_id ASC;
```

3. Pegar aquí todo el código plpgsql y triggers.

```
--- Creación de PROCEDIMIENTOS ALMACENADOS y TRIGGERS:

--- Procedimiento almacenado y Trigger para la inserción de EMPLEADOS (generales):
CREATE OR REPLACE FUNCTION funcInsertEmployee() RETURNS TRIGGER AS $$
BEGIN
  INSERT INTO employees (name, entry_date, position) VALUES(NEW.name, NEW.entry_date,
'GENERAL');
  RETURN NEW;
END
$$ LANGUAGE 'plpgsql';

CREATE TRIGGER triggerInsertEmployee INSTEAD OF INSERT
  ON v_employees
  FOR EACH ROW
  EXECUTE PROCEDURE funcInsertEmployee();

CREATE OR REPLACE FUNCTION insertEmployee(_name VARCHAR(45), _entry_date DATE) RETURNS
INT AS $$
BEGIN
 BEGIN
   INSERT INTO v_employees (name, entry_date) VALUES(_name, _entry_date);
 EXCEPTION
   WHEN others THEN
     RETURN 9;
 END;

 RETURN 0;
END
$$ LANGUAGE 'plpgsql';

--- Procedimiento almacenado y Trigger para la eliminación de EMPLEADOS (generales):
CREATE OR REPLACE FUNCTION funcDeleteEmployee() RETURNS TRIGGER AS $$
```

```
BEGIN
 INSERT INTO employee_history (log_date, employee_id, name, entry_date)
   SELECT NOW()::DATE AS log_date, *
   FROM v_employees
   WHERE employee_id = OLD.employee_id FOR UPDATE;

 DELETE FROM employees WHERE id = OLD.employee_id;
  RETURN OLD;
END
$$ LANGUAGE 'plpgsql';

CREATE TRIGGER triggerDeleteEmployee INSTEAD OF DELETE
  ON v_employees
  FOR EACH ROW
  EXECUTE PROCEDURE funcDeleteEmployee();

CREATE OR REPLACE FUNCTION deleteEmployee(_employee_id INT) RETURNS INT AS $$
BEGIN
 BEGIN
   IF NOT EXISTS(SELECT * FROM v_employees WHERE employee_id = _employee_id FOR
UPDATE) THEN
     RETURN 1;
   END IF;

   DELETE FROM v_employees WHERE employee_id = _employee_id;
 EXCEPTION
   WHEN others THEN
     RETURN 9;
 END;

 RETURN 0;
END
$$ LANGUAGE 'plpgsql';

--- Procedimiento almacenado y Trigger para la inserción de GERENTES:
CREATE OR REPLACE FUNCTION funcInsertManager() RETURNS TRIGGER AS $$
```

```
DECLARE
  var_employee_id INT := 0;
BEGIN
  IF NEW.entry_date >= NEW.start_date THEN
    RAISE EXCEPTION 'La fecha de inicio de gerencia debe ser posterior o el mismo día
que la fecha de ingreso como empleado.';
  END IF;

 IF NEW.employee_id IS NULL THEN
   INSERT INTO employees (name, entry_date, position)
     VALUES (NEW.name, NEW.entry_date, 'MANAGER')
     RETURNING id INTO var_employee_id;
 ELSE
   var_employee_id := NEW.employee_id;
 END IF;

  INSERT INTO managers (employee_id, start_date, bonus)
    VALUES(var_employee_id, NEW.start_date, NEW.bonus);

  BEGIN
    IF EXISTS(SELECT * FROM v_managers WHERE license_plate = NEW.license_plate) THEN
      RAISE EXCEPTION 'El vehículo ya ha sido previamente asignado a un gerente.';
    END IF;

    INSERT INTO manager_has_vehicle (manager_id, vehicle_license_plate)
      SELECT id, NEW.license_plate AS license_plate
      FROM managers
      WHERE employee_id = var_employee_id FOR UPDATE;
  EXCEPTION
    WHEN undefined_column THEN
    WHEN not_null_violation THEN
  END;

  RETURN NEW;
END
$$ LANGUAGE 'plpgsql';
```

```
CREATE TRIGGER triggerInsertManager INSTEAD OF INSERT
  ON v_managers
  FOR EACH ROW
  EXECUTE PROCEDURE funcInsertManager();

CREATE OR REPLACE FUNCTION insertManager(_name VARCHAR(45), _entry_date DATE,
_start_date DATE, _bonus bonus, _license_plate license_plate) RETURNS INT AS $$
BEGIN
 BEGIN
   INSERT INTO v_managers (name, entry_date, start_date, bonus, license_plate)
     VALUES(_name, _entry_date, _start_date, _bonus, _license_plate);
 EXCEPTION
   WHEN others THEN
     RETURN 9;
 END;

 RETURN 0;
END
$$ LANGUAGE 'plpgsql';

--- Procedimiento almacenado y Trigger para la eliminación de GERENTES:
CREATE OR REPLACE FUNCTION funcDeleteManager() RETURNS TRIGGER AS $$
BEGIN
  INSERT INTO manager_history (log_date, employee_id, name, entry_date, start_date,
bonus, license_plate)
    SELECT NOW()::DATE AS log_date, *
    FROM v_managers
    WHERE employee_id = OLD.employee_id;

  DELETE FROM managers WHERE employee_id = OLD.employee_id;
  UPDATE employees SET position = 'GENERAL' WHERE id = OLD.employee_id;
  RETURN OLD;
END
$$ LANGUAGE 'plpgsql';
```

```
CREATE TRIGGER triggerDeleteManager INSTEAD OF DELETE
  ON v_managers
  FOR EACH ROW
  EXECUTE PROCEDURE funcDeleteManager();

CREATE OR REPLACE FUNCTION deleteManager(_employee_id INT) RETURNS INT AS $$
BEGIN
 BEGIN
   IF NOT EXISTS(SELECT * FROM v_managers WHERE employee_id = _employee_id) THEN
     RETURN 1;
   END IF;

   DELETE FROM v_managers WHERE employee_id = _employee_id;
 EXCEPTION
   WHEN others THEN
     RETURN 9;
 END;

 RETURN 0;
END
$$ LANGUAGE 'plpgsql';

--- Procedimiento almacenado y Trigger para la inserción de INGENIEROS:
CREATE OR REPLACE FUNCTION funcInsertEngineer() RETURNS TRIGGER AS $$
DECLARE
  var_employee_id INT := 0;
 var_specialty_id INT := 0;
 var_category_id INT := 0;
BEGIN

 IF NEW.employee_id IS NULL THEN
   INSERT INTO employees (name, entry_date, position)
     VALUES (NEW.name, NEW.entry_date, 'ENGINEER')
     RETURNING id INTO var_employee_id;
 ELSE
   var_employee_id := NEW.employee_id;
```

```
 END IF;

 SELECT id INTO var_specialty_id FROM specialties
 WHERE name = NEW.specialty;

 IF var_specialty_id IS NULL THEN
   RAISE EXCEPTION 'La especialidad ingresada es inválida.';
 END IF;

 SELECT id INTO var_category_id FROM categories
 WHERE name = NEW.category;

 IF var_category_id IS NULL THEN
   RAISE EXCEPTION 'La categoría ingresada es inválida.';
 END IF;

 INSERT INTO engineers (employee_id, specialty_id, category_id)
    VALUES(var_employee_id, var_specialty_id, var_category_id);
  RETURN NEW;
END
$$ LANGUAGE 'plpgsql';

CREATE TRIGGER triggerInsertEngineer INSTEAD OF INSERT
  ON v_engineers
  FOR EACH ROW
  EXECUTE PROCEDURE funcInsertEngineer();

CREATE OR REPLACE FUNCTION insertEngineer(_name VARCHAR(45), _entry_date DATE,
_specialty VARCHAR(45), _category VARCHAR(45)) RETURNS INT AS $$
BEGIN
 BEGIN
   INSERT INTO v_engineers (name, entry_date, specialty, category)
     VALUES(_name, _entry_date, _specialty, _category);
 EXCEPTION
   WHEN others THEN
     RETURN 9;
```

```
 END;

 RETURN 0;
END
$$ LANGUAGE 'plpgsql';

--- Procedimiento almacenado para ASIGNAR INGENIERO A PROYECTO:
CREATE OR REPLACE FUNCTION assignEngineerToProject(_employee_id INT, _project_id INT,
_assignment_date DATE, _hours_per_week DECIMAL(5,2)) RETURNS INT AS $$
DECLARE
 var_engineer_id INT := 0;
 var_projects_counter INT := 0;
 var_hours_per_week DECIMAL(5, 2) := 0.0;
BEGIN
 BEGIN
   IF NOT EXISTS(SELECT * FROM v_engineers WHERE employee_id = _employee_id FOR
UPDATE) THEN
     RETURN 1;
   END IF;

   SELECT id INTO var_engineer_id FROM engineers WHERE employee_id = _employee_id;

   IF EXISTS(SELECT * FROM engineers_has_projects WHERE engineer_id = var_engineer_id
AND project_id = _project_id) THEN
     RAISE EXCEPTION 'No es posible asignar un ingeniero al mismo proyecto más de una
vez.';
   END IF;

   SELECT COUNT(DISTINCT id) INTO var_projects_counter
   FROM engineers_has_projects
   WHERE engineer_id = var_engineer_id;

   SELECT sum(hours_per_week) INTO var_hours_per_week
   FROM engineers_has_projects
   WHERE engineer_id = var_engineer_id;
```

```
   IF (var_projects_counter + 1) > 3 THEN
     RAISE EXCEPTION 'Un ingeniero no puede ser asignado a más de tres proyectos a la
vez.';
   END IF;

   IF (var_hours_per_week + _hours_per_week) > 40 THEN
     RAISE EXCEPTION 'La suma total de las horas por semana asignadas al ingeniero en
los diversos proyectos no debe ser mayor a 40.';
   END IF;

   INSERT INTO engineers_has_projects (engineer_id, project_id, assignment_date,
hours_per_week)
     VALUES(var_engineer_id, _project_id, _assignment_date, _hours_per_week);
 EXCEPTION
   WHEN others THEN
     RETURN 9;
 END;

 RETURN 0;
END
$$ LANGUAGE 'plpgsql';

--- Procedimiento almacenado para QUITAR INGENIERO A PROYECTO:
CREATE OR REPLACE FUNCTION removeEngineerToProject(_employee_id INT, _project_id INT)
RETURNS INT AS $$
DECLARE
 var_engineer_id INT := 0;
 var_relationship_id INT := NULL;
BEGIN
 BEGIN
   IF NOT EXISTS(SELECT * FROM v_engineers WHERE employee_id = _employee_id FOR
UPDATE) THEN
     RETURN 1;
   END IF;

   SELECT id INTO var_engineer_id FROM engineers WHERE employee_id = _employee_id;
```

```
   SELECT id INTO var_relationship_id
   FROM engineers_has_projects
   WHERE engineer_id = var_engineer_id AND project_id = _project_id FOR UPDATE;

   IF var_relationship_id IS NULL THEN
     RETURN 1;
   END IF;

   INSERT INTO projects_and_engineers_history (log_date, engineer_id, project_id,
assignment_date, hours_per_week)
     SELECT NOW()::DATE AS log_date, engineer_id, project_id, assignment_date,
hours_per_week
     FROM engineers_has_projects
     WHERE engineer_id = var_engineer_id AND project_id = _project_id;

   DELETE FROM engineers_has_projects WHERE id = var_relationship_id;
 EXCEPTION
   WHEN others THEN
     RETURN 9;
 END;

 RETURN 0;
END
$$ LANGUAGE 'plpgsql';

--- Procedimiento almacenado y Trigger para la eliminación de GERENTES:
CREATE OR REPLACE FUNCTION funcDeleteEngineer() RETURNS TRIGGER AS $$
DECLARE
 var_engineer_id INT := NULL;
 tmp_row record;
BEGIN
  INSERT INTO engineer_history (log_date, employee_id, name, entry_date, specialty,
category)
    SELECT NOW()::DATE AS log_date, *
    FROM v_engineers
```

```
    WHERE employee_id = OLD.employee_id FOR UPDATE;
  SELECT id INTO var_engineer_id FROM engineers WHERE employee_id = OLD.employee_id
FOR UPDATE;
  FOR tmp_row IN (SELECT project_id FROM engineers_has_projects WHERE engineer_id =
var_engineer_id) LOOP
   PERFORM removeEngineerToProject(OLD.employee_id, tmp_row.project_id);
 END LOOP;

  DELETE FROM engineers WHERE employee_id = OLD.employee_id;
  UPDATE employees SET position = 'GENERAL' WHERE id = OLD.employee_id;
  RETURN OLD;
END
$$ LANGUAGE 'plpgsql';

CREATE TRIGGER triggerDeleteEngineer INSTEAD OF DELETE
  ON v_engineers
  FOR EACH ROW
  EXECUTE PROCEDURE funcDeleteEngineer();

CREATE OR REPLACE FUNCTION deleteEngineer(_employee_id INT) RETURNS INT AS $$
BEGIN
 BEGIN
   IF NOT EXISTS(SELECT * FROM v_engineers WHERE employee_id = _employee_id FOR
UPDATE) THEN
     RETURN 1;
   END IF;

   DELETE FROM v_engineers WHERE employee_id = _employee_id;
 EXCEPTION
   WHEN others THEN
     RETURN 9;
 END;

 RETURN 0;
END
$$ LANGUAGE 'plpgsql';
```

```
--- Procedimiento almacenado para la PROMOCIÓN DE UN EMPLEADO A INGENIERO:
CREATE OR REPLACE FUNCTION promoteEmployeeToEngineer(_employee_id INT, _specialty
VARCHAR(45), _category VARCHAR(45)) RETURNS INT AS $$
BEGIN
 BEGIN
   IF NOT EXISTS(SELECT * FROM v_employees WHERE employee_id = _employee_id FOR
UPDATE) THEN
     RETURN 1;
   END IF;

   UPDATE employees SET position = 'ENGINEER' WHERE id = _employee_id;

   INSERT INTO v_engineers (employee_id, specialty, category)
     VALUES (_employee_id, _specialty, _category);
 EXCEPTION
   WHEN others THEN
     RETURN 9;
 END;

 RETURN 0;
END
$$ LANGUAGE 'plpgsql';

--- Procedimiento almacenado para la PROMOCIÓN DE UN EMPLEADO A GERENTE:
CREATE OR REPLACE FUNCTION promoteEmployeeToManager(_employee_id INT, _start_date
DATE, _bonus bonus, _license_plate license_plate) RETURNS INT AS $$
BEGIN
 BEGIN
   IF NOT EXISTS(SELECT * FROM v_employees WHERE employee_id = _employee_id FOR
UPDATE) THEN
     RETURN 1;
   END IF;

   UPDATE employees SET position = 'MANAGER' WHERE id = _employee_id;
```

```
   INSERT INTO v_managers (employee_id, start_date, bonus, license_plate)
     VALUES(_employee_id, _start_date, _bonus, _license_plate);
 EXCEPTION
   WHEN others THEN
     RETURN 9;
 END;

 RETURN 0;
END
$$ LANGUAGE 'plpgsql';

--- Procedimiento almacenado para la PROMOCIÓN DE UN INGENIERO A GERENTE:
CREATE OR REPLACE FUNCTION promoteEngineerToManager(_employee_id INT, _start_date
DATE, _bonus bonus, _license_plate license_plate) RETURNS INT AS $$
BEGIN
 BEGIN
   IF NOT EXISTS(SELECT * FROM v_engineers WHERE employee_id = _employee_id FOR
UPDATE) THEN
     RETURN 1;
   END IF;

   PERFORM deleteEngineer(_employee_id);

   PERFORM promoteEmployeeToManager(_employee_id, _start_date, _bonus,
_license_plate);
 EXCEPTION
   WHEN others THEN
     RETURN 9;
 END;

 RETURN 0;
END
$$ LANGUAGE 'plpgsql';

--- Procedimiento almacenado para MOVER UN INGENIERO A EMPLEADO:
CREATE OR REPLACE FUNCTION moveEngineerToEmployee(_employee_id INT) RETURNS INT AS $$
```

```
BEGIN
 BEGIN
   IF NOT EXISTS(SELECT * FROM v_engineers WHERE employee_id = _employee_id FOR
UPDATE) THEN
     RETURN 1;
   END IF;

   PERFORM deleteEngineer(_employee_id);
 EXCEPTION
   WHEN others THEN
     RETURN 9;
 END;

 RETURN 0;
END
$$ LANGUAGE 'plpgsql';

--- Procedimiento almacenado para MOVER UN GERENTE A EMPLEADO:
CREATE OR REPLACE FUNCTION moveManagerToEmployee(_employee_id INT) RETURNS INT AS $$
BEGIN
 BEGIN
   IF NOT EXISTS(SELECT * FROM v_managers WHERE employee_id = _employee_id) THEN
     RETURN 1;
   END IF;

   PERFORM deleteManager(_employee_id);
 EXCEPTION
   WHEN others THEN
     RETURN 9;
 END;

 RETURN 0;
END
$$ LANGUAGE 'plpgsql';

--- Procedimiento almacenado para MOVER UN GERENTE A INGENIERO:
```

```
CREATE OR REPLACE FUNCTION moveManagerToEngineer(_employee_id INT, _specialty
VARCHAR(45), _category VARCHAR(45)) RETURNS INT AS $$
BEGIN
 BEGIN
   IF NOT EXISTS(SELECT * FROM v_managers WHERE employee_id = _employee_id) THEN
     RETURN 1;
   END IF;

   PERFORM deleteManager(_employee_id);

   PERFORM promoteEmployeeToEngineer(_employee_id, _specialty, _category);
 EXCEPTION
   WHEN others THEN
     RETURN 9;
 END;

 RETURN 0;
END
$$ LANGUAGE 'plpgsql';
```

4. Pegar aquí cada operación que llame a un procedimiento almacenado:
    a. insertar-eliminar un empleado/ingeniero/gerente.

```
------ ***** OPERACIONES QUE LLAMAN A PROCEDIMIENTOS ALMACENADOS: *****

--- INSERTAR UN EMPLEADO:

/*
Puede realizarse de las siguientes dos formas (la segunda mediante procedimiento
almacenado es
```

```
la encargada de retornar los valores numéricos dependiendo del estado de la consulta).

INSERT INTO v_employees (name, entry_date) VALUES(...);
SELECT insertEmployee(name, entry_date);
*/

SELECT insertEmployee('Michael Serrato', '2019-05-06');

--- ELIMINAR UN EMPLEADO:

/*
Puede realizarse de las siguientes dos formas (la segunda mediante procedimiento
almacenado es
la encargada de retornar los valores numéricos dependiendo del estado de la consulta).

DELETE FROM v_employees WHERE employee_id = ...;
SELECT deleteEmployee(employee_id);
*/

SELECT deleteEmployee(1);

--- INSERTAR UN GERENTE:

/*
Las dos formas mostradas anteriormente para la inserción y borrado de los empleados
generales aplica de
igual forma para la vista de los gerentes:

INSERT INTO v_managers (name, entry_date, start_date, bonus, license_plate)
VALUES(...);
SELECT insertManager(name, entry_date, start_date, bonus, license_plate);
*/

SELECT insertManager('Brandon Serrato', '2019-05-07', '2019-05-15', 38000, 'AAA002');

--- ELIMINAR UN GERENTE:
```

```
/*
DELETE FROM v_managers WHERE employee_id = ...;
SELECT deleteManager(employee_id);
*/

SELECT deleteManager(2);

--- INSERTAR UN INGENIERO:

/*
Las dos formas mostradas anteriormente para la inserción y borrado de los empleados
generales aplica de
igual forma para la vista de los gerentes:

INSERT INTO v_engineers (name, entry_date, specialty, category) VALUES(...);
SELECT insertEngineer(name, entry_date, specialty, category);
*/

SELECT insertEngineer('Pedro Sanchez', '2019-05-08', 'SERVIDORES Y REDES', 'SENIOR');

--- ELIMINAR UN INGENIERO:

/*
DELETE FROM v_engineers WHERE employee_id = ...;
SELECT deleteEngineer(employee_id);
*/

SELECT deleteEngineer(3);
```

b. promover un empleado a ingeniero;

```
-- PROMOVER UN EMPLEADO A INGENIERO:
SELECT insertEmployee('Juanito Perez', '2019-05-11');
SELECT promoteEmployeeToEngineer(4, 'SOFTWARE', 'JUNIOR');
```

c. promover un ingeniero a gerente;

```
-- PROMOVER UN INGENIERO A GERENTE:
SELECT insertEngineer('Fernando Gutierrez', '2019-05-15', 'MECATRÓNICA', 'SENIOR');
SELECT promoteEngineerToManager(6, '2019-05-16', 40000, NULL);
```

d. mover un ingeniero a empleado;

```
-- MOVER UN INGENIERO A EMPLEADO:
SELECT insertEngineer('Pedro Fernandez', '2019-05-15', 'INDUSTRIAL', 'JUNIOR');
SELECT moveEngineerToEmployee(7);
SELECT * FROM employees;
```

e. mover un gerente a empleado o a ingeniero;

```
-- MOVER UN GERENTE A EMPLEADO:
SELECT insertManager('Francisco Herrera', '2019-04-01', '2019-04-30', 32000,
'AAA002');
SELECT moveManagerToEmployee(8);
SELECT * FROM employees;

-- MOVER UN GERENTE A INGENIERO:
SELECT insertManager('Jorge Martinez', '2019-04-22', '2019-05-03', 32000, 'AAA002');
SELECT moveManagerToEngineer(9, 'ELECTRÓNICA', 'JUNIOR');
SELECT * FROM employees;
```

f. asignar-quitar ingenieros a proyectos,

```
-- ASIGNAR INGENIERO A PROYECTO:
SELECT insertEngineer('Roberto Morales', '2019-05-20', 'SOFTWARE', 'JUNIOR');
SELECT assignEngineerToProject(10, 2, '2019-05-22', 20); /* Return: 0 (Todo bien) */
SELECT assignEngineerToProject(10, 1, '2019-05-23', 15); /* Return: 0 (Todo bien) */
SELECT assignEngineerToProject(10, 3, '2019-05-24', 10); /* Return: 9 (Excede el
número máximo de horas) */
SELECT assignEngineerToProject(10, 1, '2019-05-25', 5); /* Return: 9 (No puede ser
asignado al mismo proyecto más de una vez) */

-- QUITAR INGENIERO A PROYECTO:
SELECT removeEngineerToProject(10, 2); /* Return: 0 (Todo bien) */
SELECT * FROM projects_and_engineers_history;
SELECT removeEngineerToProject(10, 1); /* Return: 0 (Todo bien) */
SELECT * FROM projects_and_engineers_history;
```

F I N
