

Universidad Autónoma de Baja California
Facultad de ciencias químicas e Ingeniería

Plan de Ingeniero en Software y tecnologías emergentes



Bases de Datos (351)

Práctica 7

Funciones, Procedimientos Almacenados y Triggers

Docente:

Sukey Sayonara Nakasima Lopez

Participante(es):

Luis Eduardo Galindo Amaya (1274895)

viernes, 27 octubre 2023

Índice

1. Función para Calcular Años Transcurridos.....	3
1.1. Función años transcurridos.....	3
1.2. Calcular mi edad.....	3
Salida de la prueba.....	4
2. Procedimiento Almacenado para Calcular Edad y Años de Antigüedad.....	5
2.1. Procedimiento edad y años trabajados.....	5
2.2. Prueba del procedimiento.....	6
Salida de la Prueba.....	6
3. Procedimiento Almacenado para Estadísticas de Edad y Pensión.....	7
3.1. Procedimiento.....	7
3.2. Prueba de estadísticas de pensión.....	9
3.3. Cuestionario.....	9
¿Cuántos hombres y mujeres NO son candidatos al proceso de pensión?.....	9
¿Cuántos hombres y mujeres PUEDEN iniciar su proceso de pensión?.....	9
¿Cuántos hombres y mujeres DEBEN iniciar su proceso de pensión?.....	9
4. Trigger BEFORE INSERT para asignar categoría.....	10
4.1. Agregar el campo categoría.....	10
4.2. Trigger BEFORE INSERT.....	10
4.3. Prueba del trigger.....	10
Salida del trigger.....	11
5. Trigger AFTER INSERT para Generar Registro en Salaries.....	12
5.1. Permitir NULL en el campo to_date.....	12
5.2. Trigger AFTER INSERT.....	12
5.3. Prueba del AFTER INSERT.....	12
Salidas del trigger.....	13
6. Trigger BEFORE UPDATE con Auditoría de Cambios en Categoría.....	14
6.1. Crear tabla employee_category_audit.....	14
6.2. Trigger BEFORE UPDATE.....	14
6.3. Prueba de Trigger BEFORE UPDATE.....	14
Salida del comando.....	15
7. Trigger AFTER UPDATE para Incrementar Salario y Fecha.....	16
7.1. Eliminar las llaves primarias de salaries.....	16
7.2. Trigger.....	16
7.3. Probar el trigger.....	17
Capturas.....	17

1. Función para Calcular Años Transcurridos

Crear un FUNCIÓN que reciba una fecha y con base a ella, calcule los años transcurridos.

1.1. Función años transcurridos

```
DELIMITER $
CREATE FUNCTION fn_anios_transcurridos (fecha_inicial date)
RETURNS int deterministic
BEGIN
    DECLARE days_diff int;
    DECLARE anios_diff int;
    SET days_diff = datediff(now(), fecha_inicial);

    IF (
        month(fecha_inicial) > month(now())
        OR (
            month(fecha_inicial) = month(now())
            AND day(fecha_inicial) > day(now())
        )
    )
    THEN
        SET anios_diff = days_diff / 365 - 1;
    ELSE
        SET anios_diff = days_diff / 365;
    END IF;

    RETURN anios_diff;
END
$
```

1.2. Calcular mi edad

```
INSERT INTO employees
    (emp_no, birth_date, first_name, last_name, gender, hire_date)
VALUES
    (20000, "2001-08-20", "Luis", "Galindo", "M", now());

SELECT
    fn_anios_transcurridos (birth_date) AS edad
FROM employees
WHERE emp_no = 20000;
```

Salida de la prueba

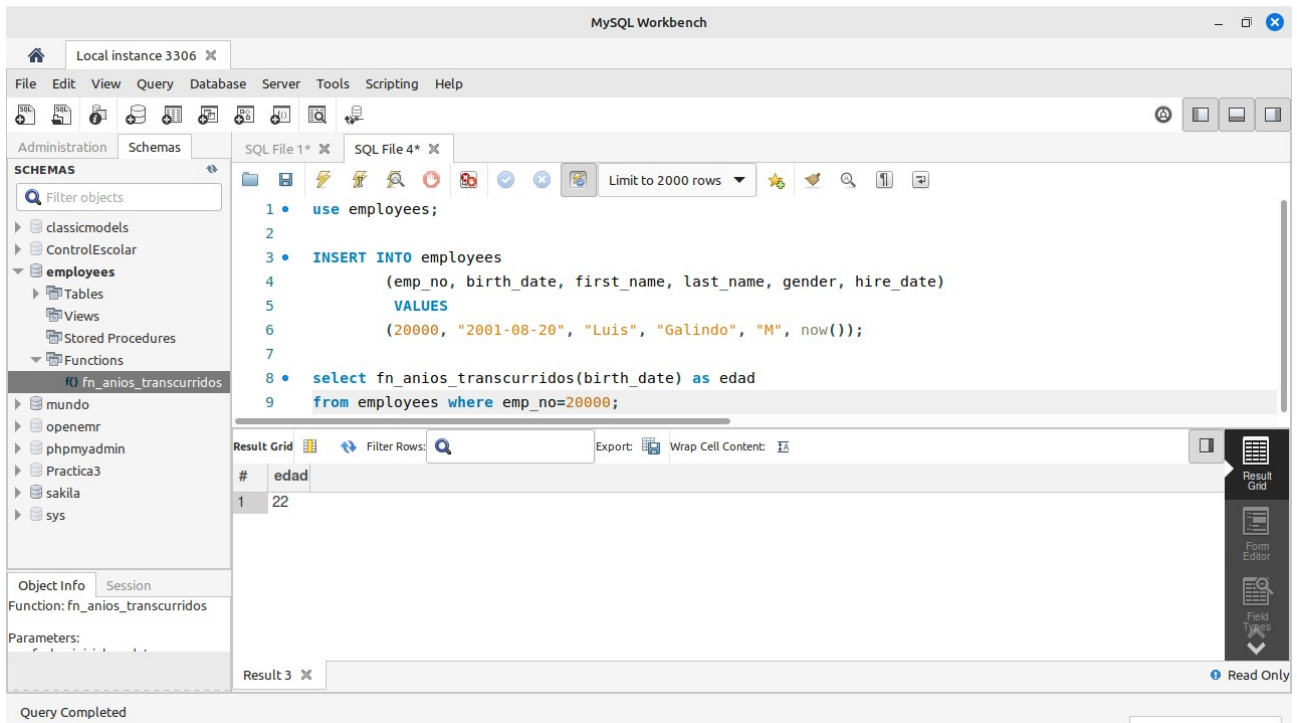


Figura 1.2.1: Mi edad es 22 años

2. Procedimiento Almacenado para Calcular Edad y Años de Antigüedad

Crear un PROCEDIMIENTO ALMACENADO en el cual con base a la tabla employees, y los campos de birth_date y hire_date calcule la edad y los años de antigüedad respectivamente, utilizando la función que creaste.

2.1. Procedimiento edad y años trabajados

```
DELIMITER $
CREATE PROCEDURE sp_edad_y_anios_trabajados ()
BEGIN
    SELECT
        emp_no AS Numero_Empleado,
        concat(first_name, ", ", last_name) AS Nombre_Empleado,
        gender AS Genero,
        fn_anios_transcurridos (birth_date) AS Edad,
        fn_anios_transcurridos (hire_date) AS Antigüedad,
        (CASE
            WHEN fn_anios_transcurridos (birth_date) BETWEEN 60 AND 64 THEN
                "Puede iniciar su proceso de pensión"
            WHEN fn_anios_transcurridos (birth_date) > 64 THEN
                "Debe iniciar su proceso de pensión"
            ELSE
                -- Antigüedad < 60
                "No es candidato para pensionarse"
        END) AS Observacion
    FROM
        employees;
END
$
```

2.2. Prueba del procedimiento

```
call sp_edad_y_anios_trabajados;
```

Salida de la Prueba

The screenshot shows the MySQL Workbench interface. The 'Query' tab is active, displaying the following SQL code:

```
1 • use employees;
2
3 • call sp_edad_y_anios_trabajados;
```

The 'Result Grid' is visible, showing the output of the stored procedure. The results are as follows:

#	Numero_Empleado	Nombre_Empleado	Genero	Edad	Antigüedad	Observacion
1	10001	Georgi,Facello	M	70	37	Debe iniciar su proceso de pensión
2	10002	Bezalel,Simmel	F	59	37	No es candidato para pensionarse
3	10003	Parto,Bamford	M	63	37	Puede iniciar su proceso de pensión
4	10004	Chirstian,Koblick	M	70	36	Debe iniciar su proceso de pensión
5	10005	Kyoichi,Maliniak	M	69	34	Debe iniciar su proceso de pensión
6	10006	Anneke,Preusig	F	71	34	Debe iniciar su proceso de pensión
7	10007	Tzvetan,Zielinski	F	66	35	Debe iniciar su proceso de pensión
8	10008	Saniya,Kalloufi	M	66	29	Debe iniciar su proceso de pensión
9	10009	Sumant,Peac	F	72	39	Debe iniciar su proceso de pensión
10	10010	Duangkaew,Piveteau	F	60	34	Puede iniciar su proceso de pensión
11	10011	Mary,Sluis	F	69	34	Debe iniciar su proceso de pensión
12	10012	Patricio,Bridgland	M	63	30	Puede iniciar su proceso de pensión
13	10013	Eberhardt,Terki	M	60	38	Puede iniciar su proceso de pensión

The status bar at the bottom indicates 'Query Completed'.

Figura 2.2.1: Salida del procedimiento

3. Procedimiento Almacenado para Estadísticas de Edad y Pensión

Crear un PROCEDIMIENTO ALMACENADO que muestre la estadística con base a la edad y los criterios establecidos en el requerimiento 2, nos diga:

- ¿Cuántos hombres y Mujeres NO son candidatos al proceso de pensión?
- ¿Cuántos hombres y Mujeres PUEDEN iniciar su proceso de pensión?
- ¿Cuántos hombres y Mujeres DEBEN iniciar su proceso de pensión?

3.1. Procedimiento

```
DELIMITER $
CREATE PROCEDURE sp_estadistica_pension ()
BEGIN
    DECLARE per_fac int;
    -- factor que convierte una cantidad de empleados a porcentaje
    SET per_fac = 100 / (SELECT count(*) FROM employees);

    -- Calcular el número de empleados
    CREATE TEMPORARY TABLE IF NOT EXISTS Freq_Pension AS
    SELECT
        -- Nombre completo del genero
        (
            CASE WHEN gender = "M" THEN
                "MASCULINO"
            WHEN gender = "F" THEN
                "FEMENINO"
            ELSE
                "INDEFINIDO"
            END) AS Genero,

        sum(fn_anios_transcurridos (birth_date) < 60)
            AS NO_Son_Candidatos,

        sum(fn_anios_transcurridos (birth_date) BETWEEN 60 AND 65)
            AS PUEDEN_Iniciar_Proceso_Pension,

        sum(fn_anios_transcurridos (birth_date) > 65)
            AS DEBEN_Iniciar_Proceso_Pension
    FROM
        employees
    GROUP BY
        gender;

    -- Calcular porcentajes
    SELECT
        Genero,
        NO_Son_Candidatos,
        (NO_Son_Candidatos * per_fac)
            AS Per_NO_Son_Candidatos,
```

```
    PUEDEN_Iniciar_Proceso_Pension,  
    (PUEDEN_Iniciar_Proceso_Pension * per_fac)  
        AS PER_PUEDEN_Iniciar_Proceso_Pension,  
  
    DEBEN_Iniciar_Proceso_Pension,  
    (DEBEN_Iniciar_Proceso_Pension * per_fac)  
        AS PER_DEBEN_Iniciar_Proceso_Pension  
FROM  
    Freq_Pension;  
  
-- Eliminar la tabla temporal  
DROP TABLE Freq_Pension;  
END  
$
```


3.2. Prueba de estadísticas de pensión

```
call sp_estadistica_pension;
```

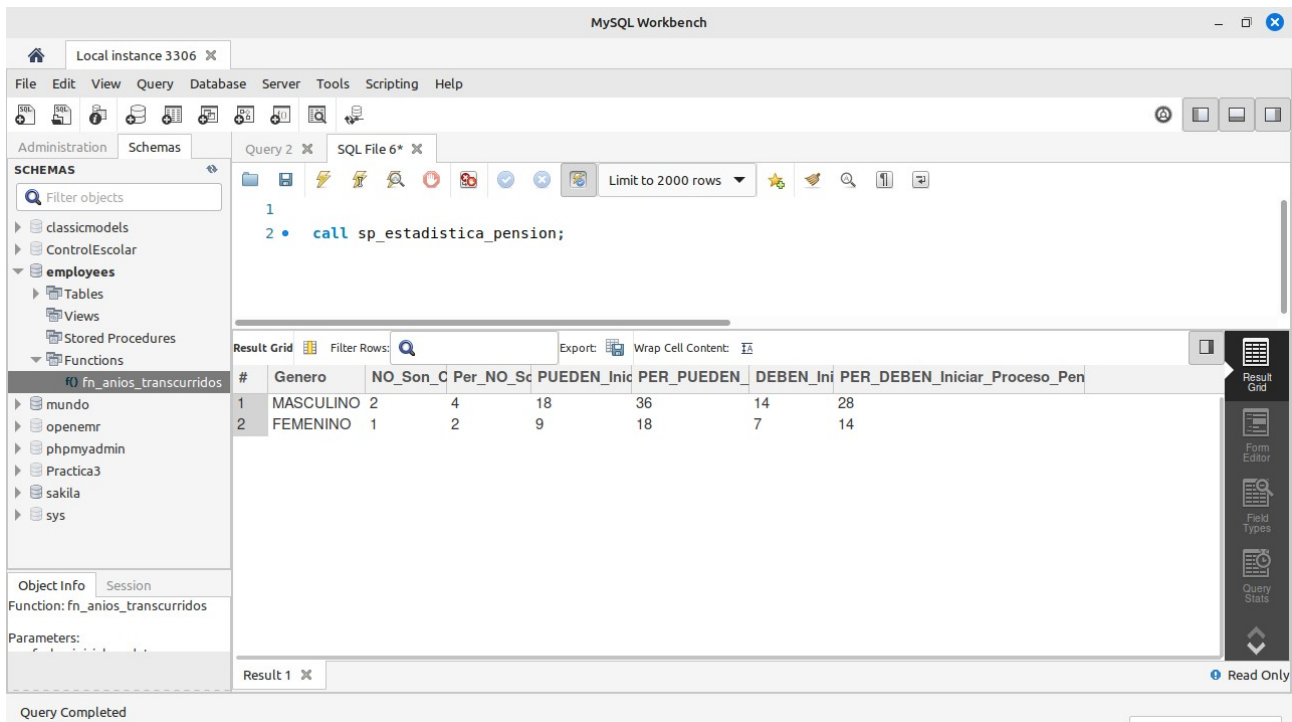


Figura 3.2.1: En la primera columna esta contabilizado le prueba

3.3. Cuestionario

¿Cuántos hombres y mujeres NO son candidatos al proceso de pensión?

- 1 hombre y 1 mujeres

¿Cuántos hombres y mujeres PUEDEN iniciar su proceso de pensión?

- 18 hombres y 9 mujeres

¿Cuántos hombres y mujeres DEBEN iniciar su proceso de pensión?

- 14 hombres y 7 mujeres

4. Trigger BEFORE INSERT para asignar categoría

Crear un TRIGGER con el tiempo y evento BEFORE INSERT donde antes de insertar un nuevo empleado a la tabla employee, se deberá asignar al campo Category el valor 1.

El campo Category en la tabla employee no existe, por lo tanto, tendrás que agregarlo, las categorías irán de la 1 a la 8.

4.1. Agregar el campo categoria

El campo 'Category' en la tabla employee no existe, por lo tanto, tendrás que agregarlo, las categorías irán de la 1 a la 8.

```
ALTER TABLE employees
ADD category INT CHECK (category >= 1 AND category <= 8);
```

4.2. Trigger BEFORE INSERT

```
DELIMITER %
CREATE TRIGGER before_set_category
  BEFORE INSERT ON employees
  FOR EACH ROW
BEGIN
  SET new.category = 1;
END
%
```

4.3. Prueba del trigger

```
INSERT INTO employees
  (emp_no, birth_date, first_name, last_name, gender, hire_date)
VALUES (20000, "2001-08-30", "Luis", "Galindo", "M", now());

SELECT * FROM employees;
```

Salida del trigger

MySQL Workbench interface showing a SQL query and its results.

Query 3:

```

1 • use employees;
2
3 • delete from employees where emp_no=20000;
4
5 • select * from employees;

```

Result Grid:

#	emp_no	birth_date	first_name	last_name	gender	hire_date	category
1	20000	2001-08-30	Luis	Galindo	M	2023-10-31	1
2	10050	1958-05-21	Yinghua	Dredge	M	1990-12-25	NULL
3	10049	1961-04-24	Basil	Tramer	F	1992-05-04	NULL
4	10048	1963-07-11	Florian	Syrotiuk	M	1985-02-24	NULL
5	10047	1952-06-29	Zvonko	Nyanch...	M	1989-03-31	NULL
6	10046	1960-07-23	Lucien	Rosenb...	M	1992-06-20	NULL
7	10045	1957-08-14	Moss	Shanbh...	M	1989-09-02	NULL
8	10044	1961-09-21	Mingsen	Casley	F	1994-05-21	NULL
9	10043	1960-09-19	Yishay	Tzvieli	M	1990-10-20	NULL
10	10042	1956-02-26	Magy	Stamatiou	F	1993-03-21	NULL
11	10041	1959-08-27	Uri	Lenart	F	1989-11-12	NULL
12	10040	1959-09-13	Weiyl	Meriste	F	1993-02-14	NULL
13	10039	1959-10-01	Alejandro	Brender	M	1988-01-19	NULL

Query Completed

Figura 4.3.1: La categoría del registro es '1'

5. Trigger AFTER INSERT para Generar Registro en Salaries

Crear un TRIGGER con el tiempo y evento AFTER INSERT que después de INSERTAR un registro en la tabla employees, genere un registro en la tabla salaries, agregando en from_date la fecha actual, en to_date dejarlo NULL y en salary establecer el valor 5000.

5.1. Permitir NULL en el campo to_date

```
ALTER TABLE salaries MODIFY to_date DATE NULL;
```

5.2. Trigger AFTER INSERT

```
DELIMITER %  
CREATE TRIGGER after_add_salary  
  AFTER INSERT ON employees  
  FOR EACH ROW  
BEGIN  
  INSERT INTO salaries  
    (emp_no, salary, from_date, to_date)  
VALUES  
  (new.emp_no, 5000, now(), null);  
END  
%
```

5.3. Prueba del AFTER INSERT

```
INSERT INTO employees  
  (emp_no, birth_date, first_name, last_name, gender, hire_date)  
VALUES  
  (20000, "2001-08-30", "Luis", "Galindo", "M", now());  
  
SELECT * FROM salaries WHERE emp_no=20000;
```

Salidas del trigger

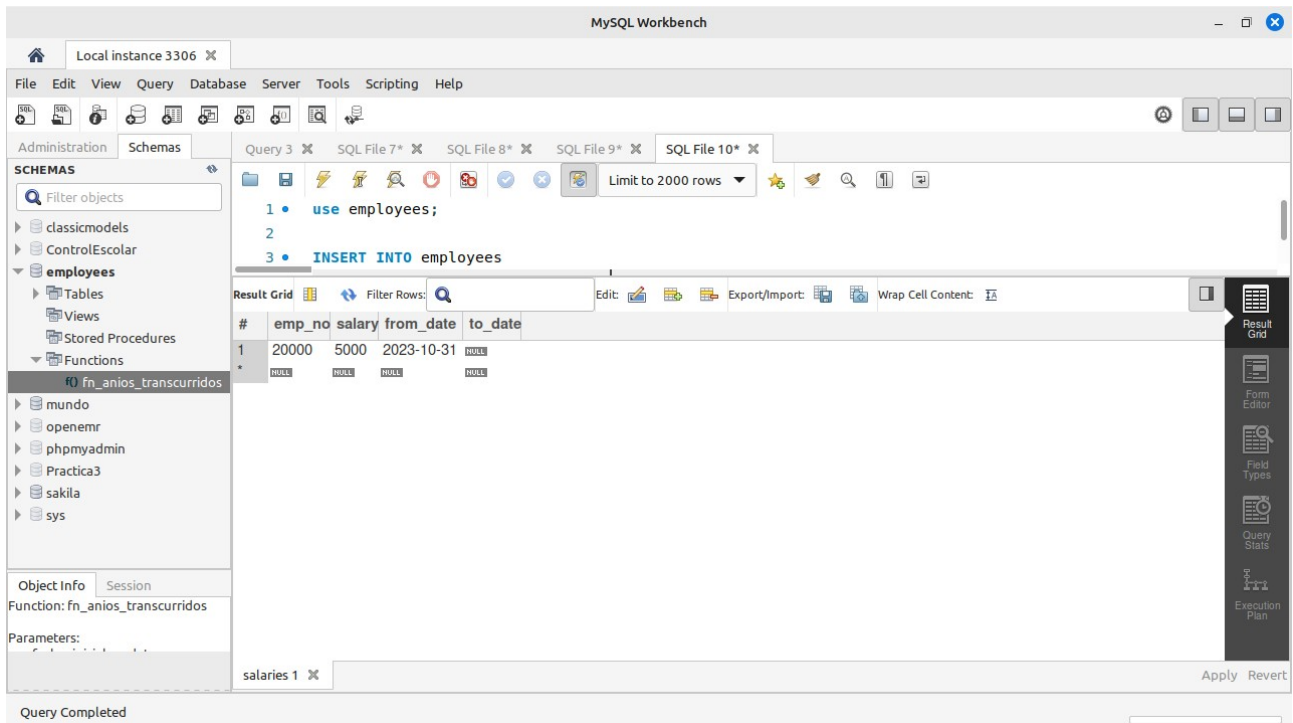


Figura 5.3.1: Salario de \$5000

6. Trigger BEFORE UPDATE con Auditoría de Cambios en Categoría

Crear un TRIGGER con el tiempo y evento BEFORE UPDATE donde antes de ACTUALIZAR un registro en la tabla employees, genere un registro en la tabla employee_category_audit, agregando para dicho empleado la categoría nueva, la categoría vieja y la fecha en la que se realizó dicho cambio. Esta tabla **NO EXISTE** así que tendrás que crearla.

6.1. Crear tabla employee_category_audit

```
CREATE TABLE if not exists employee_category_audit (  
    emp_no int not null,  
    change_date date,  
    old_category int,  
    new_Category int,  
  
    CONSTRAINT FK_emp_no FOREIGN KEY (emp_no)  
    references employees(emp_no)  
);
```

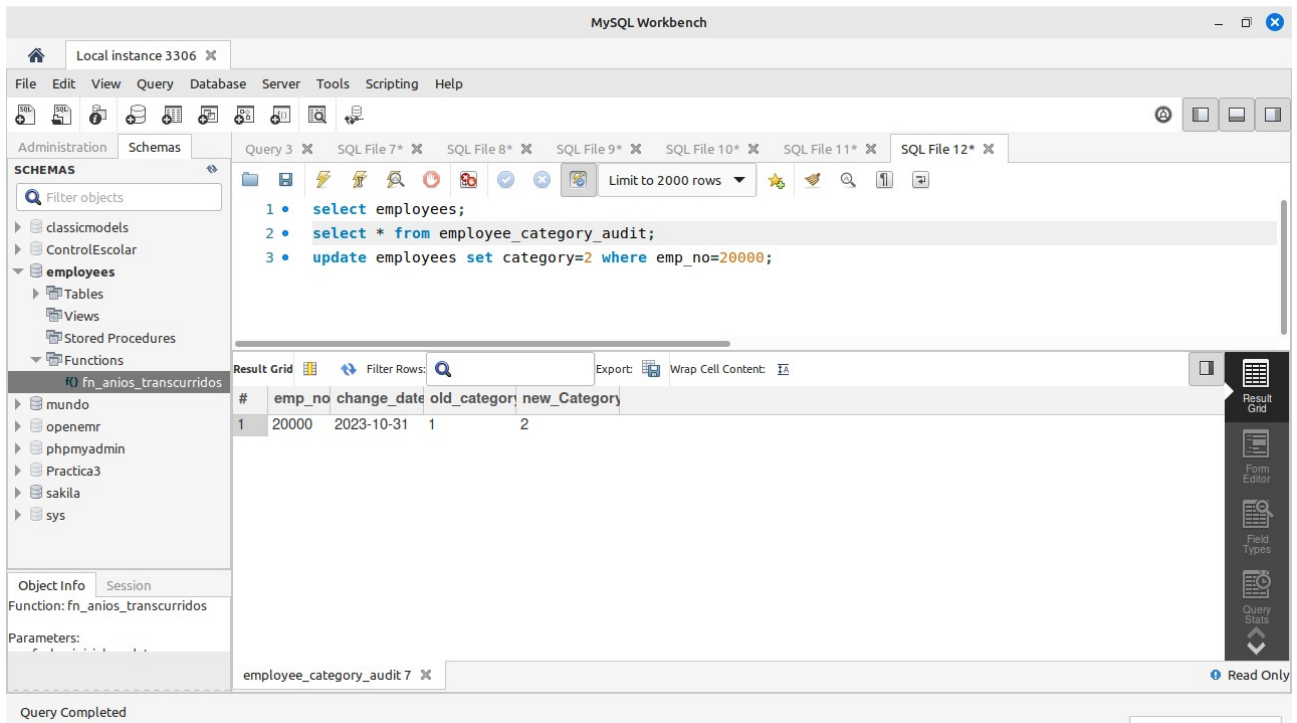
6.2. Trigger BEFORE UPDATE

```
DELIMITER %  
CREATE TRIGGER after_category_update  
    BEFORE UPDATE ON employees  
    FOR EACH ROW  
BEGIN  
    IF old.category <> new.category THEN  
        INSERT INTO employee_category_audit  
            (emp_no, change_date, old_category, new_Category)  
        VALUES  
            (new.emp_no, now(), old.category, new.category);  
    END IF;  
END  
%
```

6.3. Prueba de Trigger BEFORE UPDATE

```
UPDATE employees SET category=2 WHERE emp_no=20000;  
SELECT * FROM employee_category_audit;
```

Salida del comando



The screenshot shows the MySQL Workbench interface. The left sidebar displays the 'SCHEMAS' tree with 'employees' expanded. The central query editor contains three SQL statements: a select statement for 'employees', a select statement for 'employee_category_audit', and an update statement for 'employees' setting 'category=2' where 'emp_no=20000'. The 'Result Grid' at the bottom shows the output of the update query, displaying a single row with columns: '#', 'emp_no', 'change_date', 'old_category', and 'new_Category'. The row contains the values: 1, 20000, 2023-10-31, 1, and 2. The status bar at the bottom indicates 'Query Completed'.

#	emp_no	change_date	old_category	new_Category
1	20000	2023-10-31	1	2

Figura 6.3.1: Categoría actualizada

7. Trigger AFTER UPDATE para Incrementar Salario y Fecha

Crear un TRIGGER con el tiempo y evento AFTER UPDATE donde después de ACTUALIZAR el campo Category de la tabla employees, genere un registro en la tabla salaries, donde para dicho empleado deberás realizar lo siguiente:

7.1. Eliminar las llaves primarias de salaries

```
ALTER TABLE salaries DROP CONSTRAINT salaries_ibfk_1;  
ALTER TABLE salaries DROP CONSTRAINT `PRIMARY`;
```

7.2. Trigger

```
DELIMITER %  
CREATE TRIGGER after_set_category  
  AFTER UPDATE ON employees  
  FOR EACH ROW  
BEGIN  
  DECLARE old_salary INT;  
  DECLARE salary_fac FLOAT;  
  DECLARE application_date date;  
  
  -- ultimo salario  
  SET old_salary = (SELECT salary FROM salaries  
    WHERE old.emp_no AND to_date IS NULL);  
  
  -- factor de porcentaje  
  SET salary_fac = (old_salary / 100);  
  
  -- si es sabado o domingo  
  SET application_date = (CASE  
    WHEN DAYOFWEEK (now()) = 1 THEN  
      date_add (now(), interval 1 day) -- domingo  
    WHEN DAYOFWEEK (now()) = 7 THEN  
      date_add (now(), interval 2 day) -- sabado  
    ELSE  
      now()  
  );  
END);  
  
IF old.category <> new.category THEN  
  -- Actualizar fecha del salario  
  UPDATE salaries SET to_date = now()  
  WHERE emp_no = old.emp_no AND to_date IS NULL;  
  
  -- insertar el nuevo salario  
  INSERT INTO salaries (emp_no, from_date, to_date, salary)  
    VALUES (new.emp_no, application_date, NULL, (  
    -- calcular nuevo salario  
    CASE WHEN new.category = 1 THEN  
      old_salary + salary_fac * 5 -- + 5%  
    WHEN new.category BETWEEN 2 AND 4 THEN  
      old_salary + salary_fac * 10 -- +10%  
    WHEN new.category BETWEEN 5 AND 7 THEN
```



```

        old_salary + salary_fac * 30 -- +30%
    WHEN new.category = 8 THEN
        old_salary + salary_fac * 60 -- +60%
    END));
END IF;
END
%
```

7.3. Probar el trigger

```

-- crear al empleado
INSERT INTO employees
    (emp_no, birth_date, first_name, last_name, gender, hire_date, category)
VALUES
    (20000, "2001-08-30", "Luis", "Galindo", "M", now(), 0);

-- Agregar un salario
INSERT INTO salaries
    (emp_no, salary, from_date, to_date)
VALUES
    (20000, 1000, now(), NULL);

-- Actualizar datos
UPDATE employees SET category=3 WHERE emp_no=20000;
```

Capturas

The screenshot shows the MySQL Workbench interface. The 'Schemas' pane on the left displays the database structure, including tables like 'employees' and 'salaries'. The main query editor shows a sequence of SQL commands: using the 'employees' table, updating its 'category' to 4 for 'emp_no=20000', and selecting all records from 'salaries' where 'emp_no=20000'. The 'Result Grid' at the bottom displays the output of the last query, showing two rows of salary data for employee 20000.

#	emp_no	salary	from_date	to_date
1	20000	5000	2023-10-31	2023-11-01
2	20000	5500	2023-11-01	NULL

Figura 7.3.1: Salida del trigger