**PROGRAMACIÓN EN LA BASE DE DATOS TRIGGERS Y FUNCIONES**

**Parte I - Ejercicios de la base de datos SoftUni:**

-- 1

DROP PROCEDURE IF EXISTS nombreApell;

delimiter $$

CREATE PROCEDURE nombreApell()

BEGIN

SELECT FirstName 'nombre', LastName 'apellido'

FROM Employees

WHERE Salary >= 35000;

END $$

delimiter ;

CALL nombreApell();

-- 2

DROP PROCEDURE IF EXISTS nombreApell;

delimiter $$

CREATE PROCEDURE nombreApell(

IN salario DECIMAL(10, 4)

)

BEGIN

SELECT FirstName 'nombre', LastName 'apellido'

FROM Employees

WHERE Salary >= salario;

END $$

delimiter ;

CALL nombreApell(100);

-- 3

DROP PROCEDURE IF EXISTS ciudades;

delimiter $$

CREATE PROCEDURE ciudades(

IN letra VARCHAR(10)

)

BEGIN

DECLARE nombre VARCHAR(10) DEFAULT CONCAT(letra, "%");

SELECT Name 'ciudad', TownID 'ID'

FROM Towns

WHERE lower(Name) LIKE lower(nombre)

ORDER BY Name ASC;

END $$

delimiter ;

CALL ciudades('r');

-- 4

DROP PROCEDURE IF EXISTS empleadosciudad;

delimiter $$

CREATE PROCEDURE empleadosciudad(

IN nombreCiudad VARCHAR(30)

)

BEGIN

SELECT FirstName 'Nombre', LastName 'Apellido'

FROM Employees e

JOIN Addresses a ON e.AddressID = a.AddressID

JOIN Towns t ON t.TownID = a.TownID

WHERE t.Name = nombreCiudad

ORDER BY FirstName DESC;

END $$

delimiter ;

CALL empleadosciudad('Sofia');

-- 5

DROP FUNCTION IF EXISTS niveldesalario;

delimiter $$

CREATE FUNCTION niveldesalario(salarioEmpleado DECIMAL(10, 4))

RETURNS VARCHAR(10)

DETERMINISTIC

BEGIN

DECLARE nivelDeSalario VARCHAR(10);

IF(salarioEmpleado < 30000) THEN

BEGIN

SET nivelDeSalario = 'Low';

END;

ELSE IF(salarioEmpleado >= 30000 AND salarioEmpleado <= 50000) THEN

BEGIN

SET nivelDeSalario = 'Average';

END;

ELSE

BEGIN

SET nivelDeSalario = 'High';

END;

END IF;

END IF;

RETURN nivelDeSalario;

END $$

delimiter ;

SELECT niveldesalario(13500.0000);

-- 6

DROP PROCEDURE IF EXISTS empleadosconniveldesalario;

delimiter $$

CREATE PROCEDURE empleadosconniveldesalario(

IN nivelDeSalario VARCHAR(10)

)

BEGIN

SELECT FirstName,LastName

FROM Employees

WHERE niveldesalario(Salary) = nivelDeSalario;

END $$

delimiter ;

CALL empleadosconniveldesalario('Low');

-- 7

DROP FUNCTION IF EXISTS verdaderoofalso;

CREATE FUNCTION verdaderoofalso(letras varchar(50), palabra varchar(50))

RETURNS BOOL

DETERMINISTIC

RETURN palabra REGEXP (CONCAT('^[', letras, ']+$'));

SELECT verdaderoofalso('oistmiahf', 'Sofia');

SELECT verdaderoofalso('oistmiahf', 'halves');

SELECT verdaderoofalso('bobr', 'Rob');

SELECT verdaderoofalso('pppp', 'Guy');

-- 8

-- 9

**Parte II - Ejercicios sobre la base de datos bank\_db:**

-- 10

DROP PROCEDURE IF EXISTS nombreCompleto;

DELIMITER $$

CREATE PROCEDURE nombreCompleto()

BEGIN

SELECT

CONCAT(' ', h.first\_name, h.last\_name) AS 'Nombre completo'

FROM

account\_holders h

JOIN

(SELECT DISTINCT

a.account\_holder\_id

FROM

accounts a) a ON h.id = a.account\_holder\_id

ORDER BY `Nombre completo`;

END $$

DELIMITER ;

CALL nombreCompleto();

-- 11

DROP PROCEDURE IF EXISTS genteconmasdinerototal;

DELIMITER $$

CREATE PROCEDURE genteconmasdinerototal(cantidad DECIMAL(19, 4))

BEGIN

SELECT

CONCAT(' ', h.first\_name, h.last\_name) AS 'Nombre completo'

FROM

account\_holders h

JOIN (SELECT

a.id, a.account\_holder\_id, SUM(a.balance) AS 'Dinero total'

FROM

accounts a

GROUP BY (a.account\_holder\_id)

HAVING `Dinero total` > cantidad) AS a ON h.id = a.account\_holder\_id

ORDER BY `Nombre completo`;

END $$

DELIMITER ;

CALL genteconmasdinerototal(50000);

-- 12

DROP FUNCTION IF EXISTS valorfuturodelasuma;

DELIMITER $$

CREATE FUNCTION valorfuturodelasuma(

suma DECIMAL(14, 4), interesAnual DECIMAL(14, 4), años INT)

RETURNS DECIMAL(14, 4)

DETERMINISTIC

BEGIN

RETURN suma \* POW((1 + interesAnual), años);

END $$

DELIMITER ;

SELECT valorfuturodelasuma(1000, 2, 3);

-- 13

DROP FUNCTION IF EXISTS formula\_grande;

DELIMITER $$

CREATE FUNCTION formula\_grande(

I DECIMAL(14,4),

R DECIMAL(14,4),

T DECIMAL(14,4)

)

RETURNS DECIMAL(14,4)

DETERMINISTIC

BEGIN

DECLARE FV DECIMAL(14,4);

SET FV = I \* (1 + R);

RETURN POW(FV,T);

END $$

DELIMITER ;

SELECT formula\_grande(4,3,9);

-- 14

delimiter $$

CREATE PROCEDURE calcular\_valor\_futuro\_cuenta (account\_id INT, interest\_rate DECIMAL(19,4))

BEGIN

DECLARE future\_value DECIMAL(14, 4);

DECLARE balance DECIMAL(14, 4);

SET balance := (SELECT a.balance FROM accounts AS a WHERE a.d = account\_id);

SET future\_balance := valor\_futuro(balance, interest\_rate, 5);

-- balance \* (POW((1 + interest\_rate), 5))\_;

SELECT

a.id AS account\_id,

ah.FirstName,

ah.LastName,

a.balance,

future\_value

FROM

accounts AS a

INNER JOIN account\_holders AS ah ON a.account\_holder\_id = ah\_id

AND a.id = account\_id;

END $$

delimiter ;

-- 15 // ESTÁ MAL

delimiter $$

CREATE TRIGGER info\_cuentas AFTER UPDATE ON accounts FOR EACH ROW

BEGIN

SELECT ah.first\_name 'Nombre', ah.last\_name 'Apellido', a.balance

FROM account\_holders ah

JOIN accounts a

WHERE ah.id = a.id;

END $$

delimiter ;

-- 16

-- 17

delimiter $$

DROP PROCEDURE IF EXISTS retirardinero;

CREATE PROCEDURE retirardinero(

idCuenta INT, cantidad DECIMAL(14, 4))

BEGIN

IF accounts.balance > 0 THEN

START TRANSACTION;

UPDATE accounts a

SET

a.balance = a.balance - cantidad

WHERE

a.id = account\_id;

IF (SELECT a.balance

FROM accounts a

WHERE a.id = idCuenta) < 0

THEN ROLLBACK;

ELSE

COMMIT;

END IF;

END IF;

END $$

delimiter ;

CALL retirardinero(13, 10);

SELECT

a.id 'ID Cuenta', a.balance

FROM

accounts a

WHERE

a.id = 1;

-- 18

delimiter $$

DROP PROCEDURE IF EXISTS depositardinero;

CREATE PROCEDURE depositardinero(

idCuenta INT, cantidad DECIMAL(14, 4))

BEGIN

START TRANSACTION;

UPDATE accounts a

SET

a.balance = a.balance + cantidad

WHERE

a.id = account\_id;

IF (SELECT a.balance

FROM accounts a

WHERE a.id = idCuenta) < 0

THEN ROLLBACK;

ELSE

COMMIT;

END IF;

END $$

delimiter ;

CALL depositardinero(13, 10);

SELECT

a.id 'ID Cuenta', a.balance

FROM

accounts a

WHERE

a.id = 1;

-- 19

delimiter $$

DROP PROCEDURE IF EXISTS transferirdinero;

CREATE PROCEDURE transferirdinero(

cuentaOrigen INT, cuentaDestino INT, cantidad DECIMAL(19, 4))

BEGIN

IF (SELECT a.balance

FROM accounts a

WHERE a.id = cuentaOrigen) >= cantidad

THEN

START TRANSACTION;

UPDATE accounts a

SET

a.balance = a.balance + cantidad

WHERE

a.id = cuentaDestino;

UPDATE accounts a

SET

a.balance = a.balance - cantidad

WHERE

a.id = cuentaOrigen;

IF (SELECT a.balance

FROM accounts a

WHERE a.id = cuentaOrigen) < 0

THEN ROLLBACK;

ELSE

COMMIT;

END IF;

END IF;

END $$

delimiter ;

CALL transferirdinero(10, 12, 10);

SELECT

a.id 'ID de cuenta', a.balance 'Balance'

FROM

accounts a

WHERE

a.id IN (10, 12);

-- 20

-- 21

DROP TABLE IF EXISTS tabla\_logs;

CREATE TABLE tabla\_logs (

log\_id INT PRIMARY KEY AUTO\_INCREMENT,

account\_id INT NOT NULL,

old\_sum DECIMAL(14, 4) NOT NULL,

new\_sum DECIMAL(14, 4) NOT NULL);

DELIMITER $$

DROP TRIGGER IF EXISTS registrarlog;

CREATE TRIGGER registrarlog

AFTER UPDATE ON accounts

FOR EACH ROW

BEGIN

IF OLD.balance <> NEW.balance THEN

INSERT INTO tabla\_logs

(account\_id, old\_sum, new\_sum)

VALUES (OLD.id, OLD.balance, NEW.balance);

END IF;

END $$

DELIMITER ;

CALL transferirdinero(10, 12, 10);

SELECT \* FROM tabla\_logs;

-- 22

DROP TABLE IF EXISTS emails;

CREATE TABLE emails (

id INT PRIMARY KEY AUTO\_INCREMENT,

receptor INT NOT NULL,

asunto VARCHAR(100) NOT NULL,

mensaje VARCHAR(1000) NOT NULL);

DELIMITER $$

DROP TRIGGER IF EXISTS crearemail;

CREATE TRIGGER crearemail

AFTER INSERT ON tabla\_logs

FOR EACH ROW

BEGIN

INSERT INTO emails

(receptor, asunto, mensaje)

VALUES (

NEW.account\_id,

CONCAT('Cambio en el balance de la cuenta: ', NEW.account\_id),

CONCAT('A fecha ', DATE\_FORMAT(NOW(), '%d %M %Y'), ' el balance de su cuenta ', NEW.account\_id, ' a cambiado de ' , ROUND(NEW.old\_sum, 2), ' a ', ROUND(NEW.new\_sum, 2), '.'));

END $$

DELIMITER ;

SELECT \* FROM emails;