```
-- ============================================================
--  EVALUACIÓN BD2 — SOLUCIONES COMPLETAS EN MySQL
--  UTN - Técnico Universitario en Programación
--  Profesor: Eduardo Mónaco. Comisión 2. 22/04/2026
-- ============================================================

-- ============================================================
--  RESPUESTAS OPCIÓN MÚLTIPLE
--  1. c)  2. b)  3. b)  4. b)  5. b)  6. c)
-- ============================================================


-- ============================================================
--  CREACIÓN DE LA BASE DE DATOS Y TABLAS BASE
-- ============================================================

DROP DATABASE IF EXISTS tienda_cadena;
CREATE DATABASE tienda_cadena CHARACTER SET utf8mb4 COLLATE
utf8mb4_unicode_ci;
USE tienda_cadena;

CREATE TABLE Sucursales (
    sucursal_id   INT            AUTO_INCREMENT PRIMARY KEY,
    nombre        VARCHAR(100)   NOT NULL,
    ciudad        VARCHAR(100)   NOT NULL
);

CREATE TABLE Empleados (
    empleado_id   INT            AUTO_INCREMENT PRIMARY KEY,
    nombre        VARCHAR(100)   NOT NULL,
    apellido      VARCHAR(100)   NOT NULL,
    sucursal_id   INT,
    FOREIGN KEY (sucursal_id) REFERENCES Sucursales(sucursal_id)
);

CREATE TABLE Clientes (
    cliente_id    INT            AUTO_INCREMENT PRIMARY KEY,
    nombre        VARCHAR(100)   NOT NULL,
    apellido      VARCHAR(100)   NOT NULL,
    email         VARCHAR(150)   UNIQUE NOT NULL
);

CREATE TABLE Ventas (
    venta_id      INT            AUTO_INCREMENT PRIMARY KEY,
    cliente_id    INT            NOT NULL,
    empleado_id   INT            NOT NULL,
    sucursal_id   INT            NOT NULL,
    monto         DECIMAL(10,2)  NOT NULL,
    fecha         DATETIME       NOT NULL DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (cliente_id)  REFERENCES Clientes(cliente_id),
    FOREIGN KEY (empleado_id) REFERENCES Empleados(empleado_id),
    FOREIGN KEY (sucursal_id) REFERENCES Sucursales(sucursal_id)
);

-- Datos de prueba
INSERT INTO Sucursales (nombre, ciudad) VALUES
    ('Sucursal Centro',  'Buenos Aires'),
    ('Sucursal Norte',   'Rosario'),
```

```
    ('Sucursal Sur',     'Mar del Plata'),
    ('Sucursal Este',    'Córdoba');

INSERT INTO Empleados (nombre, apellido, sucursal_id) VALUES
    ('Carlos',  'García',    1),
    ('María',   'López',     1),
    ('Sergio',  'Pérez',     2),
    ('Laura',   'Rodríguez', 3),
    ('Miguel',  'Fernández', 4);

INSERT INTO Clientes (nombre, apellido, email) VALUES
    ('Ana',    'Martínez',  'ana@mail.com'),
    ('Luis',   'González',  'luis@mail.com'),
    ('Sofía',  'Torres',    'sofia@mail.com'),
    ('Pedro',  'Ramírez',   'pedro@mail.com'),
    ('Julia',  'Morales',   'julia@mail.com');

INSERT INTO Ventas (cliente_id, empleado_id, sucursal_id, monto, fecha)
VALUES
    (1, 1, 1, 30000.00, '2024-11-01'),
    (2, 2, 1, 45000.00, '2024-11-15'),
    (3, 3, 2, 60000.00, '2024-12-01'),
    (4, 4, 3, 20000.00, '2024-12-10'),
    (5, 5, 4, 15000.00, '2025-01-05'),
    (1, 1, 1, 25000.00, '2025-01-20'),
    (2, 2, 2, 55000.00, '2025-02-01'),
    (3, 3, 3, 10000.00, '2025-02-14'),
    (1, 4, 4, 80000.00, '2025-03-01'),
    (2, 5, 1, 35000.00, '2025-03-15');


-- ============================================================
--  EJERCICIO 7
--  CTE que calcula total vendido por sucursal +
--  Vista SucursalesDestacadas (total > $50.000)
--  Mostrar: sucursal_id, nombre_sucursal, ciudad, total_vendido
-- ============================================================

-- ── FORMA A — CTE nombrada + CREATE VIEW ──────────────────
-- La CTE agrega las ventas; la VIEW filtra con WHERE.
WITH TotalesPorSucursal AS (
    SELECT
        s.sucursal_id,
        s.nombre   AS nombre_sucursal,
        s.ciudad,
        SUM(v.monto) AS total_vendido
    FROM Sucursales s
    JOIN Ventas v ON s.sucursal_id = v.sucursal_id
    GROUP BY s.sucursal_id, s.nombre, s.ciudad
)
SELECT * FROM TotalesPorSucursal WHERE total_vendido > 50000;

-- La vista se define directamente usando la CTE dentro de ella:
CREATE OR REPLACE VIEW SucursalesDestacadas AS
WITH TotalesPorSucursal AS (
    SELECT
        s.sucursal_id,
        s.nombre   AS nombre_sucursal,
```

```
        s.ciudad,
        SUM(v.monto) AS total_vendido
    FROM Sucursales s
    JOIN Ventas v ON s.sucursal_id = v.sucursal_id
    GROUP BY s.sucursal_id, s.nombre, s.ciudad
)
SELECT sucursal_id, nombre_sucursal, ciudad, total_vendido
FROM TotalesPorSucursal
WHERE total_vendido > 50000;

SELECT * FROM SucursalesDestacadas;


-- ── FORMA B — Subconsulta derivada (sin CTE explícita) ────────
-- Equivalente funcional: la subconsulta hace el trabajo de la CTE.
CREATE OR REPLACE VIEW SucursalesDestacadas AS
SELECT sucursal_id, nombre_sucursal, ciudad, total_vendido
FROM (
    SELECT
        s.sucursal_id,
        s.nombre   AS nombre_sucursal,
        s.ciudad,
        SUM(v.monto) AS total_vendido
    FROM Sucursales s
    JOIN Ventas v ON s.sucursal_id = v.sucursal_id
    GROUP BY s.sucursal_id, s.nombre, s.ciudad
) AS sub
WHERE total_vendido > 50000;

SELECT * FROM SucursalesDestacadas;


-- ── FORMA C — HAVING en lugar de WHERE externo ────────────────
-- HAVING filtra grupos directamente en el GROUP BY, evitando
-- una subcapa extra. Semánticamente idéntico a las formas A y B.
CREATE OR REPLACE VIEW SucursalesDestacadas AS
SELECT
    s.sucursal_id,
    s.nombre   AS nombre_sucursal,
    s.ciudad,
    SUM(v.monto) AS total_vendido
FROM Sucursales s
JOIN Ventas v ON s.sucursal_id = v.sucursal_id
GROUP BY s.sucursal_id, s.nombre, s.ciudad
HAVING SUM(v.monto) > 50000;

SELECT * FROM SucursalesDestacadas;


-- ============================================================
--  EJERCICIO 8
--  Trigger AFTER INSERT en Ventas →
--  Inserta en auditoria_ventas: venta_id, empleado_id, fecha
-- ============================================================

CREATE TABLE IF NOT EXISTS auditoria_ventas (
    auditoria_id  INT         AUTO_INCREMENT PRIMARY KEY,
    venta_id      INT         NOT NULL,
```

```
    empleado_id   INT         NOT NULL,
    fecha_operacion DATETIME  NOT NULL
);


-- ── FORMA A — Trigger con NEW y NOW() ──────────────────────
-- La forma más directa: usa los valores del registro insertado.
DROP TRIGGER IF EXISTS trg_auditoria_ventas_a;
DELIMITER $$
CREATE TRIGGER trg_auditoria_ventas_a
AFTER INSERT ON Ventas
FOR EACH ROW
BEGIN
    INSERT INTO auditoria_ventas (venta_id, empleado_id, fecha_operacion)
    VALUES (NEW.venta_id, NEW.empleado_id, NOW());
END$$
DELIMITER ;

-- Prueba:
INSERT INTO Ventas (cliente_id, empleado_id, sucursal_id, monto, fecha)
VALUES (3, 2, 1, 12000.00, NOW());
SELECT * FROM auditoria_ventas;


-- ── FORMA B — Trigger usando la columna fecha de la venta ────
-- En vez de NOW() se registra la fecha que viene en el propio
-- registro de Ventas, lo que puede ser más exacto en cargas batch.
DROP TRIGGER IF EXISTS trg_auditoria_ventas_b;
DELIMITER $$
CREATE TRIGGER trg_auditoria_ventas_b
AFTER INSERT ON Ventas
FOR EACH ROW
BEGIN
    INSERT INTO auditoria_ventas (venta_id, empleado_id, fecha_operacion)
    VALUES (NEW.venta_id, NEW.empleado_id, NEW.fecha);
END$$
DELIMITER ;


-- ── FORMA C — Trigger con INSERT … SET (sintaxis alternativa) ─
-- MySQL acepta INSERT … SET como alternativa a INSERT … VALUES.
DROP TRIGGER IF EXISTS trg_auditoria_ventas_c;
DELIMITER $$
CREATE TRIGGER trg_auditoria_ventas_c
AFTER INSERT ON Ventas
FOR EACH ROW
BEGIN
    INSERT INTO auditoria_ventas
    SET venta_id       = NEW.venta_id,
        empleado_id    = NEW.empleado_id,
        fecha_operacion = NOW();
END$$
DELIMITER ;


-- ── FORMA D — Trigger con bloque condicional (validación extra) ─
-- Agrega lógica: solo audita si el monto es mayor a 0
-- (demuestra el uso de IF dentro del trigger).
```

```
DROP TRIGGER IF EXISTS trg_auditoria_ventas_d;
DELIMITER $$
CREATE TRIGGER trg_auditoria_ventas_d
AFTER INSERT ON Ventas
FOR EACH ROW
BEGIN
    IF NEW.monto > 0 THEN
        INSERT INTO auditoria_ventas (venta_id, empleado_id,
fecha_operacion)
        VALUES (NEW.venta_id, NEW.empleado_id, NOW());
    END IF;
END$$
DELIMITER ;


-- ============================================================
--  EJERCICIO 9
--  Índice sobre cliente_id en Ventas +
--  Justificación del tipo elegido
-- ============================================================

-- ── FORMA A — Índice simple (no único) ──────────────────────
-- Es el más apropiado: cliente_id NO es único en Ventas
-- (un cliente puede tener muchas ventas). Un índice UNIQUE
-- rechazaría el segundo registro del mismo cliente.
CREATE INDEX idx_ventas_cliente_id
ON Ventas (cliente_id);

/*
  JUSTIFICACIÓN (Forma A):
  Se elige un índice no único (CREATE INDEX) porque cliente_id es
  una clave foránea: un mismo cliente puede aparecer en múltiples
  ventas. El índice acelera JOINs con la tabla Clientes y los
  GROUP BY / WHERE que filtran por cliente. No se usa UNIQUE porque
  eso impediría registrar más de una venta por cliente.
*/


-- ── FORMA B — Índice compuesto (cliente_id + fecha) ─────────
-- Útil cuando las consultas frecuentes filtran por cliente Y
-- por rango de fechas. El optimizador puede resolverlas con
-- solo este índice (index-only scan parcial).
CREATE INDEX idx_ventas_cliente_fecha
ON Ventas (cliente_id, fecha);

/*
  JUSTIFICACIÓN (Forma B):
  Índice compuesto: cliente_id como columna líder (alta selectividad
  en filtros por cliente) + fecha para cubrir consultas del tipo
  "ventas del cliente X en el último mes". Mejora tanto GROUP BY
  como ORDER BY cuando ambas columnas están presentes en el WHERE.
*/


-- ── FORMA C — Índice con USING BTREE explícito ──────────────
-- MySQL usa B-Tree por defecto, pero declararlo explícitamente
-- documenta la intención y deja en claro que se optimiza para
-- búsquedas por rango/igualdad, no para texto completo.
```

```
CREATE INDEX idx_ventas_cliente_btree
ON Ventas (cliente_id)
USING BTREE;

/*
  JUSTIFICACIÓN (Forma C):
  Se especifica USING BTREE (estructura por defecto de InnoDB)
  porque las consultas sobre cliente_id usan operadores de igualdad
  (=) y rango (<, >, BETWEEN). Un índice HASH sería más rápido en
  igualdad exacta pero no soporta rangos ni ORDER BY; B-Tree cubre
  todos esos casos.
*/

-- Ver índices creados sobre la tabla:
SHOW INDEX FROM Ventas;

-- Verificar que el optimizador usa el índice:
EXPLAIN SELECT * FROM Ventas WHERE cliente_id = 1;


-- ============================================================
--  EJERCICIO 10
--  Función dias_desde_ultima_compra(cliente_id) +
--  Vista ClientesInactivos (sin compras en últimos 90 días)
--  Mostrar: nombre, apellido, email, dias_sin_comprar
-- ============================================================

-- ── FORMA A — DATEDIFF + MAX(fecha) ──────────────────────────
DROP FUNCTION IF EXISTS dias_desde_ultima_compra;
DELIMITER $$
CREATE FUNCTION dias_desde_ultima_compra(p_cliente_id INT)
RETURNS INT
READS SQL DATA
DETERMINISTIC
BEGIN
    DECLARE ultima_fecha DATE;
    SELECT MAX(fecha) INTO ultima_fecha
    FROM Ventas
    WHERE cliente_id = p_cliente_id;

    IF ultima_fecha IS NULL THEN
        RETURN NULL;   -- el cliente nunca compró
    END IF;

    RETURN DATEDIFF(CURDATE(), ultima_fecha);
END$$
DELIMITER ;

-- Vista usando la función:
CREATE OR REPLACE VIEW ClientesInactivos AS
SELECT
    c.nombre,
    c.apellido,
    c.email,
    dias_desde_ultima_compra(c.cliente_id) AS dias_sin_comprar
FROM Clientes c
WHERE dias_desde_ultima_compra(c.cliente_id) > 90
   OR dias_desde_ultima_compra(c.cliente_id) IS NULL;
```

```
SELECT * FROM ClientesInactivos;


-- ── FORMA B — TIMESTAMPDIFF en lugar de DATEDIFF ──────────────
-- TIMESTAMPDIFF(DAY, ...) considera horas exactas; más preciso
-- cuando las fechas incluyen componente horario.
DROP FUNCTION IF EXISTS dias_desde_ultima_compra;
DELIMITER $$
CREATE FUNCTION dias_desde_ultima_compra(p_cliente_id INT)
RETURNS INT
READS SQL DATA
DETERMINISTIC
BEGIN
    DECLARE ultima_fecha DATETIME;
    SELECT MAX(fecha) INTO ultima_fecha
    FROM Ventas
    WHERE cliente_id = p_cliente_id;

    IF ultima_fecha IS NULL THEN
        RETURN NULL;
    END IF;

    RETURN TIMESTAMPDIFF(DAY, ultima_fecha, NOW());
END$$
DELIMITER ;

CREATE OR REPLACE VIEW ClientesInactivos AS
SELECT
    c.nombre,
    c.apellido,
    c.email,
    dias_desde_ultima_compra(c.cliente_id) AS dias_sin_comprar
FROM Clientes c
WHERE dias_desde_ultima_compra(c.cliente_id) > 90
   OR dias_desde_ultima_compra(c.cliente_id) IS NULL;

SELECT * FROM ClientesInactivos;


-- ── FORMA C — Vista con CTE + JOIN (sin llamar la función) ────
-- Alternativa que no depende de la función: la lógica se incrusta
-- directamente en la vista mediante una CTE y un LEFT JOIN.
-- Útil para entender cómo funciona la función internamente.
DROP FUNCTION IF EXISTS dias_desde_ultima_compra;
DELIMITER $$
CREATE FUNCTION dias_desde_ultima_compra(p_cliente_id INT)
RETURNS INT
READS SQL DATA
DETERMINISTIC
BEGIN
    RETURN (
        SELECT DATEDIFF(CURDATE(), MAX(fecha))
        FROM Ventas
        WHERE cliente_id = p_cliente_id
    );
END$$
DELIMITER ;
```

```sql
CREATE OR REPLACE VIEW ClientesInactivos AS
WITH UltimasCompras AS (
    SELECT
        cliente_id,
        MAX(fecha)                    AS ultima_compra,
        DATEDIFF(CURDATE(), MAX(fecha)) AS dias_sin_comprar
    FROM Ventas
    GROUP BY cliente_id
)
SELECT
    c.nombre,
    c.apellido,
    c.email,
    COALESCE(uc.dias_sin_comprar, NULL) AS dias_sin_comprar
FROM Clientes c
LEFT JOIN UltimasCompras uc ON c.cliente_id = uc.cliente_id
WHERE uc.dias_sin_comprar > 90
   OR uc.ultima_compra IS NULL;   -- incluye clientes que nunca compraron

SELECT * FROM ClientesInactivos;


-- ── FORMA D — Vista con subconsulta correlacionada ──────────
-- Sin CTE ni función: subconsulta correlacionada calcula los días
-- directamente en el SELECT y el WHERE los filtra.
CREATE OR REPLACE VIEW ClientesInactivos AS
SELECT
    c.nombre,
    c.apellido,
    c.email,
    (
        SELECT DATEDIFF(CURDATE(), MAX(v.fecha))
        FROM Ventas v
        WHERE v.cliente_id = c.cliente_id
    ) AS dias_sin_comprar
FROM Clientes c
WHERE (
        SELECT DATEDIFF(CURDATE(), MAX(v.fecha))
        FROM Ventas v
        WHERE v.cliente_id = c.cliente_id
      ) > 90
   OR NOT EXISTS (
        SELECT 1 FROM Ventas v WHERE v.cliente_id = c.cliente_id
      );

SELECT * FROM ClientesInactivos;


-- ============================================================
--  FIN DEL SCRIPT
-- ============================================================
```