Practica 4

PATIENT
(patient_id, patient_name, patient_address, patient_city, primary_phone, secondary_phone)

DOCTOR
(doctor_id, doctor_name, doctor_address, doctor_city, doctor_speciality)

APPOINTMENT
(patient_id, appointment_date, appointment_duration, contact_phone, observations, payment_card)

MEDICAL_REVIEW
(patient_id, appointment_date, doctor_id)

PRESCRIBED_MEDICATION
(patient_id, appointment_date, medication_name)

Ejercicio 2

```
SELECT patient_id, patient_name
FROM patient NATURAL JOIN appointment
WHERE contact_phone = primary_phone AND patient_id NOT IN (
   SELECT patient_id
   FROM patient NATURAL JOIN appointment
   WHERE contact_phone <> primary_phone
   )
```

Ejercicio 3

```
CREATE VIEW appointments.doctors_per_patients AS (
   SELECT patient_id, doctor_id
 FROM Patient INNER JOIN Doctor ON (patient_city = doctor_city)
 )
```

Ejercicio 4

a_

```
SELECT patient_id, count(*)
FROM doctors_per_patients
GROUP BY patient_id
```

b_

```
SELECT patient_name, patient_id
FROM Patient
WHERE patient_id NOT IN (
    SELECT patient_id
  FROM doctors_per_patients
)
```

c_

```
SELECT doctor_id
FROM doctors_per_patients
GROUP BY doctor_id
HAVING count(*) > 5
```

Ejercicio 5

```
CREATE TABLE APPOINTMENTS_PER_PATIENT (
    idApP INT(11) NOT NULL AUTO_INCREMENT,
    id_patient INT(11) NOT NULL,
    count_appointments INT(11),
    last_update DATETIME,
    user VARCHAR(16),
    PRIMARY KEY (idApP)
);
```

Ejercicio 6

1. Crear un Stored Procedure que realice los siguientes pasos dentro de una transacción:
   a. Realizar la siguiente consulta: cada *pacient* (identificado por *id_patient*), calcule la cantidad de appointments que tiene registradas. Registrar la fecha en la que se realiza esta carga y además del usuario con el se realiza.
   b. Guardar el resultado de la consulta en un cursor.
   c. Iterar el cursor e insertar los valores correspondientes en la tabla APPOINTMENTS PER PATIENT. Tenga en cuenta que last_update es la fecha en que se realiza esta carga, es decir la fecha actual, mientras que user es el usuario logueado actualmente, utilizar las correspondientes funciones para esto.

a_

```
DELIMITER //
CREATE PROCEDURE calcular_appointments ()
BEGIN
   START TRANSACTION;

   SELECT patient_id, COUNT(*) AS count_appointments, NOW() AS last_u
pdate, CURRENT_USER() AS user
   FROM Appointment
   GROUP BY patient_id;

   COMMIT;
END//
DELIMITER ;
```

b_

```
DELIMITER //

CREATE PROCEDURE calcular_appointments_cursor()
BEGIN
```

```
    -- Variables para guardar los valores del cursor
    DECLARE done INT DEFAULT FALSE;
    DECLARE v_patient_id INT;
    DECLARE v_count INT;

    -- Declarar el cursor con la consulta
    DECLARE cur_appointments CURSOR FOR
        SELECT patient_id, COUNT(*) AS count_appointments, NOW() AS last_
update, CURRENT_USER() AS user
        FROM Appointment
        GROUP BY patient_id;

    -- Handler para detectar el fin del cursor
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

    -- Iniciar transacción
    START TRANSACTION;

    -- Abrir cursor
    OPEN cur_appointments;

    -- Iterar sobre cada fila del cursor
    read_loop: LOOP
        FETCH cur_appointments INTO v_patient_id, v_count;
        IF done THEN
            LEAVE read_loop;
        END IF;

        INSERT INTO APPOINTMENTS_PER_PATIENT
        (id_patient, count_appointments, last_update, user)
        VALUES
        (v_patient_id, v_count, NOW(), CURRENT_USER());
    END LOOP;

    -- Cerrar cursor
    CLOSE cur_appointments;

    -- Confirmar transacción
```

```
    COMMIT;
END //

DELIMITER ;
```

c_

```
DELIMITER //

CREATE PROCEDURE calcular_appointments_cursor()
BEGIN
    -- Variables para guardar los valores del cursor
    DECLARE done INT DEFAULT FALSE;
    DECLARE v_patient_id INT;
    DECLARE v_count INT;

    -- Declarar el cursor con la consulta
    DECLARE cur_appointments CURSOR FOR
        SELECT patient_id, COUNT(*) AS count_appointments
        FROM Appointment
        GROUP BY patient_id;

    -- Handler para detectar el fin del cursor
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

    -- Iniciar transacción
    START TRANSACTION;

    -- Abrir cursor
    OPEN cur_appointments;

    -- Iterar sobre cada fila del cursor
    read_loop: LOOP
        FETCH cur_appointments INTO v_patient_id, v_count;
        IF done THEN
            LEAVE read_loop;
        END IF;
```

```
        INSERT INTO APPOINTMENTS_PER_PATIENT
        (id_patient, count_appointments, last_update, user)
        VALUES
        (v_patient_id, v_count, NOW(), CURRENT_USER());
    END LOOP;

    -- Cerrar cursor
    CLOSE cur_appointments;

    -- Confirmar transacción
    COMMIT;
END //

DELIMITER ;
```

Ejercicio 7

Indique si las siguientes afirmaciones sobre triggers son verdaderas o falsas. Justifique las falsas.

1. Un trigger se ejecuta únicamente cuando se inserta una fila en una tabla. ❌ Se ejecuta cuando se hace un INSERT, DELETE o UPDATE.
2. Un trigger puede ejecutarse antes o después de la operación, esto es definido automáticamente según el tipo de la operación (UPDATE, INSERT o DELETE) ❌ Falso. No se define automaticamente, se puede elegir manualmente.
3. Todo trigger debe asociarse a una tabla en concreto. ✅
4. NEW y OLD son palabras clave que permiten acceder a los valores de las filas afectadas y se pueden usar ambos independientemente de la operación utilizada. ❌ NEW y OLD con update, NEW con insert y OLD con delete.
5. FOR EACH ROW en un trigger se usa para indicar que el trigger se ejecutará una vez por cada fila afectada por la operación. ✅

Ejercicio 8

Crear un Trigger de modo que al insertar un dato en la tabla Appointment, se actualice la cantidad de appointments del paciente, la fecha de actualización y el usuario responsable de la misma (actualiza la tabla APPOINTMENTS PER PATIENT).

```
DELIMITER //
CREATE TRIGGER update_cant_app
BEFORE INSERT ON appointment
FOR EACH ROW
BEGIN
   UPDATE appointments_per_patient
   SET count_appointments = count_appointments + 1,
         last_update = NOW(),
         user = CURRENT_USER()
   WHERE (NEW.patient_id = id_patient);
END;//
DELIMITER ;
```

Ejercicio 9

Crear un stored procedure que sirva para agregar un *appointment*, junto el registro de un doctor que lo atendió (*medical_review*) y un medicamento que se le recetó (*prescribed_medication*), dentro de una sola transacción. El stored procedure debe recibir
los siguientes parámetros: patient_id, doctor_id, appointment_duration, contact_phone, appointment_address, medication_name. El appointment_date será la fecha actual. Los
atributos restantes deben ser obtenidos de la tabla Patient (o dejarse en NULL).

No anda. Consultar. Afecta a 0 lineas.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `add_appointment`(
   IN p_patient_id INT,
   IN p_doctor_id INT,
   IN p_appointment_duration INT,
```

```
    IN p_contact_phone VARCHAR(45),
    IN p_appointment_address VARCHAR(255),
    IN p_medication_name VARCHAR(30)
)
BEGIN
    DECLARE v_now DATETIME;
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        -- Si algo falla, se hace rollback
        ROLLBACK;
    END;

    SET v_now = NOW();

    START TRANSACTION;

    -- 1. Insertar el turno
    INSERT INTO appointment (
        patient_id, appointment_date, appointment_duration, contact_phone, o
bservations, payment_card
    ) VALUES (
        p_patient_id, v_now, p_appointment_duration, p_contact_phone, p_app
ointment_address, NULL
    );

    -- 2. Insertar la revisión médica (usa la misma fecha)
    INSERT INTO medical_review (
        patient_id, appointment_date, doctor_id
    ) VALUES (
        p_patient_id, v_now, p_doctor_id
    );

    -- 3. Insertar el medicamento prescripto (usa la misma fecha)
    INSERT INTO prescribed_medication (
        patient_id, appointment_date, medication_name
    ) VALUES (
        p_patient_id, v_now, p_medication_name
    );
```

```
    -- 4. Confirmar
    COMMIT;
END
```

Ejercicio 10

Afecta a 0 lineas.