Practica 4 BD1

Esquema final

> La clave primaria está en negrita

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 1

1. Crea un usuario para las bases de datos usando el nombre '*appointments_user'*. Asigne a estos todos los permisos sobre sus respectivas tablas. Habiendo creado este usuario evitaremos el uso de '*root'* para el resto del trabajo práctico.
   Adicionalmente, con respecto a esta base de datos:
   1. Cree un usuario sólo con permisos para realizar consultas de selección, es decir que no puedan realizar cambios en la base. Use el nombre 'appointments_select'.
   2. Cree un usuario que pueda realizar consultas de selección, inserción, actualización y eliminación a nivel de filas, pero que no puedan modificar el esquema. Use el nombre 'appointments_update'.
   3. Cree un usuario que tenga los permisos de los anteriores, pero que además pueda modificar el esquema de la base de datos. Use el nombre 'appointments_schema'.

Resolución

Inciso 1

```
GRANT SELECT, INSERT, UPDATE, DELETE, CREATE, DROP, ALTER, CREATE ROUTINE,
ALTER ROUTINE, EXECUTE
ON appointments.* TO 'appointments_user'@'localhost';
```

Inciso 1.1

```
CREATE USER 'appointments_select'@'localhost' IDENTIFIED BY 'bd1_2025';
GRANT SELECT ON appointments.* TO 'appointments_select'@'localhost';
```

Inciso 1.2

```
CREATE USER 'appointments_update'@'localhost' IDENTIFIED BY 'bd1_2025';
GRANT SELECT, INSERT, UPDATE, DELETE ON appointments.* TO
'appointments_update'@'localhost';
```

Inciso 1.3

```
CREATE USER 'appointments_schema'@'localhost' IDENTIFIED BY 'bd1_2025';
GRANT ALL PRIVILEGES ON appointments.* TO
'appointments_schema'@'localhost';
```

Ejercicio 2

Hallar aquellos pacientes que para todas sus consultas médicas siempre hayan dejado su número de teléfono primario (nunca el teléfono secundario).

Solución 1

```
SELECT p.patient_id, p.patient_name
FROM patient p INNER JOIN appointment a ON p.patient_id = a.patient_id
GROUP BY p.patient_id, p.primary_phone, p.patient_name
HAVING COUNT(*) = SUM(a.contact_phone = p.primary_phone);
```

Solución 2

```
SELECT p.patient_id, p.patient_name
FROM PATIENT p
WHERE NOT EXISTS (
    SELECT *
    FROM APPOINTMENT a
    WHERE a.patient_id = p.patient_id
      AND a.contact_phone <> p.primary_phone
);
```

Ejercicio 3

Crear una vista llamada ‘doctors_per_patients’ que muestre los id de los pacientes y los id de doctores de la ciudad donde vive el paciente.

Resolución

```
CREATE VIEW appointments.doctors_per_patients AS
SELECT p.patient_id, d.doctor_id
FROM appointments.patient p, appointments.doctor d
ON p.patient_city = d.doctor_city;
```

Ejercicio 4

Utiliza la vista generada en el ejercicio anterior para resolver las siguientes consultas:

1. Obtener la cantidad de doctores por cada paciente que tiene disponible en su ciudad
2. Obtener los nombres de los pacientes sin doctores en su ciudad
3. Obtener los doctores que comparten ciudad con más de cinco pacientes.

Resolución

Inciso 1

```
SELECT patient_id, COUNT(doctor_id) AS cantidad_doctores
FROM appointments.doctors_per_patients
GROUP BY patient_id;
```

Inciso 2

```
SELECT p.patient_name
FROM patient p
LEFT JOIN appointments.doctors_per_patients dp
  ON p.patient_id = dp.patient_id
WHERE dp.doctor_id IS NULL;
```

Inciso 3

```
SELECT doctor_id
FROM appointments.doctors_per_patients
GROUP BY doctor_id
HAVING COUNT(patient_id) > 5;
```

Ejercicio 5

Resolución

```
CREATE TABLE appointments_per_patient (
    idApP INT(11) NOT NULL AUTO_INCREMENT,
```

```
    id_patient INT(11),
    count_appointments INT(11),
    last_update DATETIME,
    user VARCHAR(16),
    PRIMARY KEY (idApP)
);
```

Ejercicio 6

Crear un Stored Procedure que realice los siguientes pasos dentro de una transacción:

1. 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.
2. Guardar el resultado de la consulta en un cursor.
3. 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.

Resolución

```
DELIMITER //

CREATE PROCEDURE appointments_patient()
BEGIN
    DECLARE aux_id INT(11);
    DECLARE aux_count INT(11);
    DECLARE aux_last DATETIME;
    DECLARE aux_user VARCHAR(16);
    DECLARE fin INT DEFAULT 0;

    DECLARE cursor_appointments CURSOR FOR
        SELECT a.patient_id,
               COUNT(*) AS count_appointments,
               NOW() AS last_update
        FROM appointment a
        GROUP BY a.patient_id;

    DECLARE CONTINUE HANDLER FOR NOT FOUND SET fin = 1;

    SET aux_user = LEFT(CURRENT_USER(), 16);

    START TRANSACTION;

    OPEN cursor_appointments;
```

```
    loop_cursor: LOOP
        FETCH cursor_appointments INTO aux_id, aux_count, aux_last;

        IF fin = 1 THEN
            LEAVE loop_cursor;
        END IF;

        INSERT INTO appointments_per_patient (id_patient,
count_appointments, last_update, user)
        VALUES (aux_id, aux_count, aux_last, aux_user);
    END LOOP;

    CLOSE cursor_appointments;

    COMMIT;
END //

DELIMITER ;
```

Ejercicio 7

1. 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.
    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)
    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.
    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.

Resolución

Inciso 1

Falso, puede ejecutarse en otros eventos también, no solo cuando se inserta en una tabla. Como en update y delete.

Inciso 2

Falso, esto es definido al crear el trigger con BEFORE o AFTER

Inciso 3

Verdadero

Inciso 4

Falso, **no se pueden usar ambos indistintamente en cualquier operación**, y dependiendo del tipo de trigger algunos no estarán disponibles.

Inciso 5

Verdadero.