

TABLAS:

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)

USE bd1;

1) Paso 1: Crear el usuario appointments_user con todos los permisos

```
CREATE USER 'appointments_user'@'localhost';
```

```
GRANT ALL PRIVILEGES ON bd1.* TO 'appointments_user'@'localhost';
```

```
FLUSH PRIVILEGES;
```

a- Paso 2: Crear el usuario appointments_select con permisos solo de selección

```
CREATE USER 'appointments_select'@'localhost';
```

```
GRANT SELECT ON bd1.* TO 'appointments_select'@'localhost';
```

```
FLUSH PRIVILEGES;
```

b- Paso 3: Crear el usuario appointments_update con permisos de selección, inserción, actualización y eliminación a nivel de filas.

```
CREATE USER 'appointments_update'@'localhost';
```

```
GRANT SELECT, INSERT, UPDATE, DELETE ON bd1.* TO 'appointments_update'@'localhost';
```

```
FLUSH PRIVILEGES;
```

c- Paso 4: Crear el usuario appointments_schema con permisos de selección, inserción, actualización, eliminación y modificación del esquema.

```
CREATE USER 'appointments_schema'@'localhost';
```

```
GRANT SELECT, INSERT, UPDATE, DELETE, ALTER, CREATE, DROP ON bd1.* TO 'appointments_schema'@'localhost';
```

```
FLUSH PRIVILEGES;
```

```

2) USE bd1;
SELECT *
FROM PATIENT p
WHERE NOT EXISTS (
    SELECT 1
    FROM APPOINTMENT a
    WHERE a.patient_id = p.patient_id
    AND a.contact_phone = p.secondary_phone
);

```

```

SELECT *
FROM patient p
WHERE (p.secondary_phone NOT IN (
    SELECT contact_phone
    FROM appointment)
);

```

3) VISTA:

```

USE bd1;
DROP VIEW doctors_per_patients;
CREATE VIEW doctors_per_patients AS
    SELECT p.patient_id, d.doctor_id
    FROM patient p
        INNER JOIN doctor d ON p.patient_city = d.doctor_city;

```

4) a- USE bd1;

```

SELECT patient_id
FROM patient p
WHERE (
    SELECT COUNT(doctor_id)
    FROM doctor d
    WHERE p.patient_city = d.doctor_city)
= (
    SELECT COUNT(DISTINCT d.doctor_id)
    FROM medical_review mw
        INNER JOIN doctor d ON mw.doctor_id = d.doctor_id
    WHERE p.patient_id = mw.patient_id
    AND p.patient_city = d.doctor_city
)
ORDER BY patient_id
LIMIT 100;

```

b- USE bd1;

```
SELECT patient_id
FROM patient p
WHERE (
    SELECT COUNT(doctor_id)
    FROM doctors_per_patients dp      -- ESTO CAMBIA
    WHERE p.patient_id = dp.patient_id) -- ESTO CAMBIA
= (
    SELECT COUNT(DISTINCT d.doctor_id)
    FROM medical_review mw
        INNER JOIN doctor d ON mw.doctor_id = d.doctor_id
    WHERE p.patient_id = mw.patient_id
    AND p.patient_city = d.doctor_city
)
ORDER BY patient_id
LIMIT 100;
```

```
5) CREATE TABLE APPOINTMENTS_PER_PATIENT (
    idApP INT(11) PRIMARY KEY AUTO_INCREMENT, -- Clave primaria con auto-incremento
    id_patient INT(11) NOT NULL,              -- Identificador del paciente, obligatorio
    count_appointments INT(11),               -- Cantidad de citas
    last_update DATETIME,                     -- Fecha y hora de la última actualización
    user VARCHAR(16),                         -- Usuario que realizó la última modificación
);
```

No hace falta:

```
FOREIGN KEY (id_patient) REFERENCES PATIENT(patient_id) -- FK a la tabla PATIENT
```

6) STORED PROCEDURE:

```
USE bd1;
DELIMITER //
CREATE PROCEDURE storedProcedure() BEGIN
    DECLARE fin INT DEFAULT 0;
    DECLARE id_paciente INT(11);
    DECLARE cant INT(11);

    -- Definir el cursor para calcular el número de citas para cada paciente
    DECLARE contar CURSOR FOR
        SELECT a.patient_id, COUNT(a.patient_id) -- AS cantAppointments -- var contadora
        FROM appointment a
        GROUP BY a.patient_id;

    DECLARE CONTINUE HANDLER FOR NOT FOUND SET fin = 1; -- Manejar fin del cursor

    START TRANSACTION;      -- Iniciar la transacción

    OPEN contar;             -- Abrir el cursor

    loop_cursor: LOOP        -- Iterar sobre cada fila en el cursor
        FETCH NEXT FROM contar INTO id_paciente, cant;

        IF fin = 1 THEN
            LEAVE loop_cursor;
        END IF;

        INSERT INTO appointments_per_patient(id_patient, count_appointments, last_update,
user)                                -- columna del appointments_per_patient
        VALUES (id_paciente, cant, NOW(), CURRENT_USER());
    END LOOP;

    CLOSE contar;           -- Cerrar el cursor

    COMMIT;                 -- Confirmar la transacción
END;
// DELIMITER ;
```

7) TRIGGER: -- no se puede modificar, si o si borrar y desp volver a crear

USE bd1;

DROP TRIGGER IF EXISTS addAppointmentPorCliente;

DELIMITER //

CREATE TRIGGER addAppointmentPorCliente

AFTER INSERT ON appointment FOR EACH ROW

BEGIN

UPDATE appointments_per_patient

SET count_appointments = (count_appointments + 1),

last_update = NOW(),

user = CURRENT_USER()

WHERE id_patient = NEW.patient_id;

END;

// DELIMITER;

VERIFICAR: DROP TRIGGER IF EXISTS addAppointmentPorCliente;

DELIMITER //

CREATE TRIGGER addAppointmentPorCliente

AFTER INSERT ON appointment FOR EACH ROW

BEGIN

DECLARE existe;

SELECT 1

INTO existe

FROM appointments_per_patient

WHERE id_patient = NEW.patient_id;

IF existe = 1

UPDATE appointments_per_patient

SET count_appointments = (count_appointments + 1),

last_update = NOW(),

user = CURRENT_USER()

WHERE id_patient = NEW.patient_id;

ELSE

INSERT INTO appointments_per_patient (id_patient, count_appointments,
last_update, user)

VALUES (NEW.patient_id, 1, NOW(),CURRENT_USER());

END IF;

END;

// DELIMITER ;

8) USE bd1;

DROP PROCEDURE IF EXISTS agregarAppointment;

DELIMITER //

CREATE PROCEDURE agregarAppointment (IN idPaciente INT, IN idDoctor INT, IN durCita INT,
IN telContacto VARCHAR(45), IN nombreMedicacion VARCHAR(30))

BEGIN

DECLARE diaCita DATETIME; -- Declarar variable local

SET diaCita = NOW();

START TRANSACTION;

INSERT INTO appointment (patient_id, appointment_date, appointment_duration,
contact_phone, observations, payment_card)

VALUES (idPaciente, diaCita, durCita, telContacto, NULL, NULL);

INSERT INTO medical_review (patient_id, appointment_date, doctor_id)

VALUES (idPaciente, diaCita, idDoctor);

INSERT INTO prescribed_medication (patient_id, appointment_date, medication_name)

VALUES (idPaciente, diaCita, nombreMedicacion);

COMMIT;

END;

// DELIMITER ;

9) USE bd1;

CALL agregarAppointment (10004427, 1003, 30, "+54 15 2913 9963", "Paracetamol");

//

SELECT * FROM appointment WHERE patient_id = 10004427;

SELECT * FROM medical_review WHERE patient_id = 10004427;

SELECT * FROM prescribed_medication WHERE patient_id = 10004427;

10) EXPLAIN

```
SELECT count(a.patient_id)
FROM appointment a, patient p, doctor d, medical_review mr
WHERE a.patient_id = p.patient_id
      AND a.patient_id = mr.patient_id
      AND a.appointment_date = mr.appointment_date
      AND mr.doctor_id = d.doctor_id
      AND d.doctor_specialty = "Cardiology"
      AND p.patient_city = "Rosario"
```

	id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
►	1	SIMPLE	p	ALL	PRIMARY	HULL	HULL	HULL	1000	Using where
	1	SIMPLE	a	ref	PRIMARY	PRIMARY	4	bd1.p.patient_id	9	Using index
	1	SIMPLE	mr	ref	PRIMARY,doctor_id	PRIMARY	9	bd1.p.patient_id,bd1.a.appointment_date	1	Using index
	1	SIMPLE	d	eq_ref	PRIMARY	PRIMARY	4	bd1.mr.doctor_id	1	Using where

a- ¿Qué atributos del plan de ejecución encuentra relevantes para evaluar la performance de la consulta?

- **type**: tipo de acceso realizado sobre las filas.
- **key**: índice seleccionado para realizar la operación. La elección correcta de un índice indica una mejor performance.
- **rows**: cantidad de filas sobre las que opera la consulta. Cuantas + filas haya, + lenta será.

ADICIONAL -> filtered: indica el porcentaje de filas que se espera que se filtren. Cuanto > sea el porcentaje, se considera que la consulta es + eficiente.

b- Observe en particular el atributo type ¿cómo se están aplicando los JOIN entre las tablas involucradas?

ALL (p): indica que las tablas se escanean por completo. Es el menos eficiente.

EQ_REF (d): indica que se está usando un índice que coincide entre ambas tablas para unirlos.

REF (a, mr): indica que se está usando un índice para encontrar una fila específica.

Comparando los 3 tipos, **eq_ref** realiza el JOIN de manera más eficiente puesto que no evalúa las tablas completas.

c- Según lo que observó en los puntos anteriores, ¿qué mejoras se pueden realizar para optimizar la consulta?

Se pueden agregar índices en la tabla p, que es la que se une con el tipo ALL. De esta manera va a hacer una consulta + eficiente.

También podría agregar índice en la tabla d, para usarlo en vez del where.

d- Aplice las mejoras propuestas y vuelva a analizar el plan de ejecución. ¿Qué cambios observa?

```
CREATE INDEX p_index ON patient(patient_city);
```

	id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
▶	1	SIMPLE	p	ref	PRIMARY,p_index	p_index	768	const	82	Using where; Using index
	1	SIMPLE	a	ref	PRIMARY	PRIMARY	4	bd1.p.patient_id	9	Using index
	1	SIMPLE	mr	ref	PRIMARY,doctor_id	PRIMARY	9	bd1.p.patient_id,bd1.a.appointment_date	1	Using index
	1	SIMPLE	d	eq_ref	PRIMARY	PRIMARY	4	bd1.mr.doctor_id	1	Using where

```
DROP INDEX p_index ON patient;
```

POR QUE ACÁ NO MEJORA????? Si además le agrego el índice al doctor:

```
CREATE INDEX d_index ON doctor(doctor_specialty);
```

Result Grid Filter Rows: Export: Wrap Cell Content:										
	id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
▶	1	SIMPLE	p	ref	PRIMARY,p_index	p_index	768	const	82	Using where; Using index
	1	SIMPLE	a	ref	PRIMARY	PRIMARY	4	bd1.p.patient_id	9	Using index
	1	SIMPLE	mr	ref	PRIMARY,doctor_id	PRIMARY	9	bd1.p.patient_id,bd1.a.appointment_date	1	Using index
	1	SIMPLE	d	eq_ref	PRIMARY,d_index	PRIMARY	4	bd1.mr.doctor_id	1	Using where