Práctica Nro. 4

Conceptos Aplicados usando MySQL

Publicación: 13/10/2025
Finalización: 25/10/2025

Para la resolución de este TP se necesita tener instalado una instancia de MySQL, para instalarla, ingrese a este link y descargue el instalador correspondiente a su SO.

PREPARACIÓN BD

Un hospital posee una base de datos para almacenar información sobre las atenciones que se realizan para sus pacientes, además de los doctores que los atendieron en cada atención y los medicamentos que le fueron recetados.
El esquema con el que cuentan es el siguiente:

APPOINTMENTS(patient_id, patient_name, patient_address, patient_city, primary_phone, secondary_phone, doctor_id, doctor_name, doctor_address, doctor_city, doctor_speciality, appointment_date, appointment_duration, observations, payment_card, contact_phone, medication_name)

Clave candidata del esquema APPOINTMENTS:

CC: (patient_id, doctor_id, appointment_date, medication_name)

Dependencias funcionales válidas en el esquema APPOINTMENTS:

DF1: patient_id -> patient_name, patient_address, patient_city, primary_phone, secondary_phone
DF2: doctor_id-> doctor_name, doctor_address, doctor_city, doctor_speciality
DF3: patient_id, appointment_date -> appointment_duration, contact_phone, observations, payment_card

Dependencias multivaluadas válidas en el esquema APPOINTMENTS:

DM1: patient_id, appointment_date ->> doctor_id
DM2: patient_id, appointment_date ->> medication_name

Luego de haber aplicado el proceso de normalización quedan las siguientes particiones en 4FN:

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)

Y la siguiente Clave Primaria:

CP = (patient_id, doctor_id, appointment_date, medication_name)

Se proveen dos archivos separados con lo necesario para la creación de las tablas e inserción de datos. Ambos archivos se encuentran en el comprimido **apppointments.sql.zip** adjunto a esta práctica.

Para crear los esquemas y cargar los datos, hacerlo desde línea de comando. Para esto, descomprimir los archivos y ejecutar desde la terminal el siguiente comando para acceder a la terminal mysql:

mysql -h localhost -u <nombre-de-usuario> -p

dentro de la terminal mysql, crear ambos esquemas:

mysql> create database appointments;
mysql> exit;

nuevamente en la terminal, ejecutar los scripts que contienen ambos archivos: *'appointments.sql'* creará las tablas, mientras que *'insert_appointments.sql'* crea una serie de tuplas de ejemplo para poder realizar las consultas.

mysql appointments -h localhost -u root -p < ruta_del_archivo

donde ruta_del_archivo es el path al archivo provisto.
Nota: Quizá deba ingresar la contraseña del usuario por cada comando que ejecute en nombre de este. Puede utilizar el usuario root.

EJERCICIOS

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:
   a. 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'.
   b. 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'.

   c. 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'.

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

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.

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

   a. Obtener la cantidad de doctores por cada paciente que tiene disponible en su ciudad
   b. Obtener los nombres de los pacientes sin doctores en su ciudad
   c. Obtener los doctores que comparten ciudad con más de cinco pacientes.

5. Escribe y ejecute la sentencia correspondiente para crear la siguiente tabla:

   APPOINTMENTS_PER_PATIENT
   idApP: int(11) PK AI
   id_patient: int(11)
   count_appointments: int(11)
   last_update: datetime
   user: varchar(16)

6. 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.

7. Indique si las siguientes afirmaciones sobre triggers son verdaderas o falsas. Justifique las falsas.
   a. Un trigger se ejecuta únicamente cuando se inserta una fila en una tabla.

Falso. Un disparador(trigger) es un objeto con nombre dentro de una base de datos el cual se asocia con una tabla y se activa cuando ocurre en ésta una operación INSERT, UPDATE y DELETE.

   b. 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. {BEFORE | AFTER} indica cuándo se activa el trigger, antes o después de la operación.

   c. Todo trigger debe asociarse a una tabla en concreto.

Verdadero. Un trigger es un objeto de la BD que se asocia a una tabla y funciona como un disparador de una accion ante un evento.

d. 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.

Falso. NEW se usa en INSERT y UPDATE y OLD se usa en UPDATE y DELETE. No se pueden usar los dos “independientemente” de la operación.

e. FOR EACH ROW en un trigger se usa para indicar que el trigger se ejecutará una vez por cada fila afectada por la operación.

Verdadero. Corresponde a la definición.

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).

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).

10. Ejecutar el stored procedure del punto 9 con los siguientes datos:
    patient_id: 10004427
    doctor_id: 1003
    appointment_duration: 30
    contact_phone: +54 15 2913 9963
    appointment_address: ‘Hospital Italiano’
    medication_name: ‘Paracetamol’

11. Considerando la siguiente consulta: Analice su plan de ejecución mediante el uso de la sentencia EXPLAIN.

```sql
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'
```

Análisis:

```
mysql> 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	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	d	NULL	ALL	PRIMARY	NULL	NULL	NULL	100	10.00	Using where
1	SIMPLE	p	NULL	ALL	PRIMARY	NULL	NULL	NULL	1000	10.00	Using where; Using join buffer (hash join)
1	SIMPLE	mr	NULL	ref	PRIMARY,doctor_id	doctor_id	8	appointments.d.doctor_id,appointments.p.patient_id	1	100.00	Using index
1	SIMPLE	a	NULL	eq_ref	PRIMARY	PRIMARY	9	appointments.p.patient_id,appointments.mr.appointment_date	1	100.00	Using index

4 rows in set, 1 warning (0.01 sec)

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

De la tabla de EXPLAIN, los más importantes son:

- type: indica el tipo de acceso (full scan, uso de índice, etc.).
- key: el índice que realmente se está usando para la búsqueda.
- rows: número estimado de filas que MySQL tendrá que leer.
- Extra: indica si se aplican filtros adicionales (Using where), Using index, o join buffers (Using join buffer (hash join)).

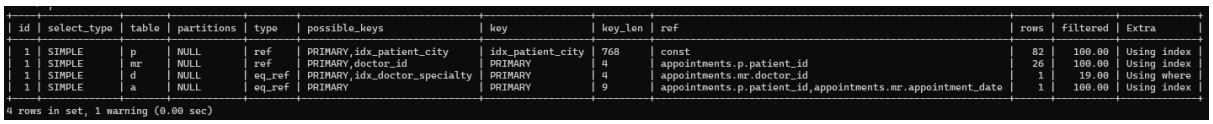
b. Observe en particular el atributo type ¿cómo se están aplicando los JOIN entre las tablas involucradas?

- d (doctor) - ALL: se hace un full table scan, no se usa índice para filtrar por doctor_specialty. Ineficiente si hay muchos doctores.
- p (patient) - ALL: se hace full table scan, se filtra por patient_city. MySQL usa un join buffer (hash join) para combinarlo con d.
- mr (medical_review) - ref: usa el índice doctor_id para buscar filas coincidentes.
- a (appointment) - eq_ref: búsqueda eficiente por PRIMARY KEY (patient_id + appointment_date).

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

- Crear índices sobre las columnas que se usan en los filtros.
- Considerar índices compuestos si se van a hacer frecuentemente joins y filtros juntos.
- Cambiar la consulta a JOIN explícito.

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



id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	p	NULL	ref	PRIMARY,idx_patient_city	idx_patient_city	768	const	82	100.00	Using index
1	SIMPLE	mr	NULL	ref	PRIMARY,doctor_id	PRIMARY	4	appointments.p.patient_id	26	100.00	Using index
1	SIMPLE	d	NULL	eq_ref	PRIMARY,idx_doctor_specialty	PRIMARY	4	appointments.mr.doctor_id	1	19.00	Using where
1	SIMPLE	a	NULL	eq_ref	PRIMARY	PRIMARY	9	appointments.p.patient_id,appointments.mr.appointment_date	1	100.00	Using index

4 rows in set, 1 warning (0.00 sec)

Los índices son mucho más eficientes porque transforman un "escaneo completo de tabla" (que lee todas las filas de todas las tablas) en una "búsqueda dirigida" usando árboles. En la consulta, sin índices MySQL se tendría que examinar todas las filas en doctor, patient, appointment y medical_review, comparando cada una con d.doctor_specialty = 'Cardiology' y p.patient_city = 'Rosario'. Con idx_doctor_specialty y idx_patient_city, MySQL usa el índice para saltar directamente a las pocas filas reduciendo el tiempo de ejecución lineal a logarítmico, lo que significa leer del disco/memoria solo bloques específicos en lugar de toda la tabla, logrando "Using index".