

Sistemas transaccionales - Entrega 3

Grupo 4

Aicardy Pacheco, Juan Angel

Castillo Alvarez, Mariana

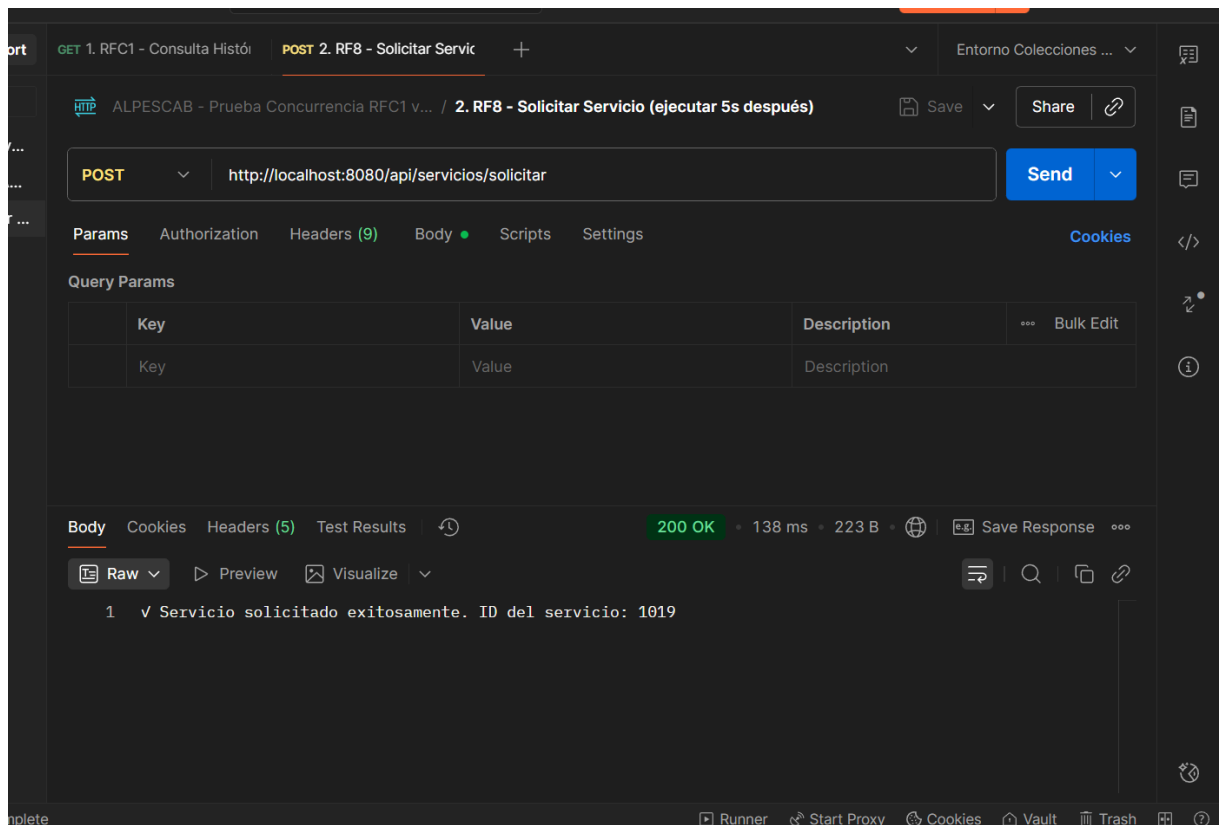
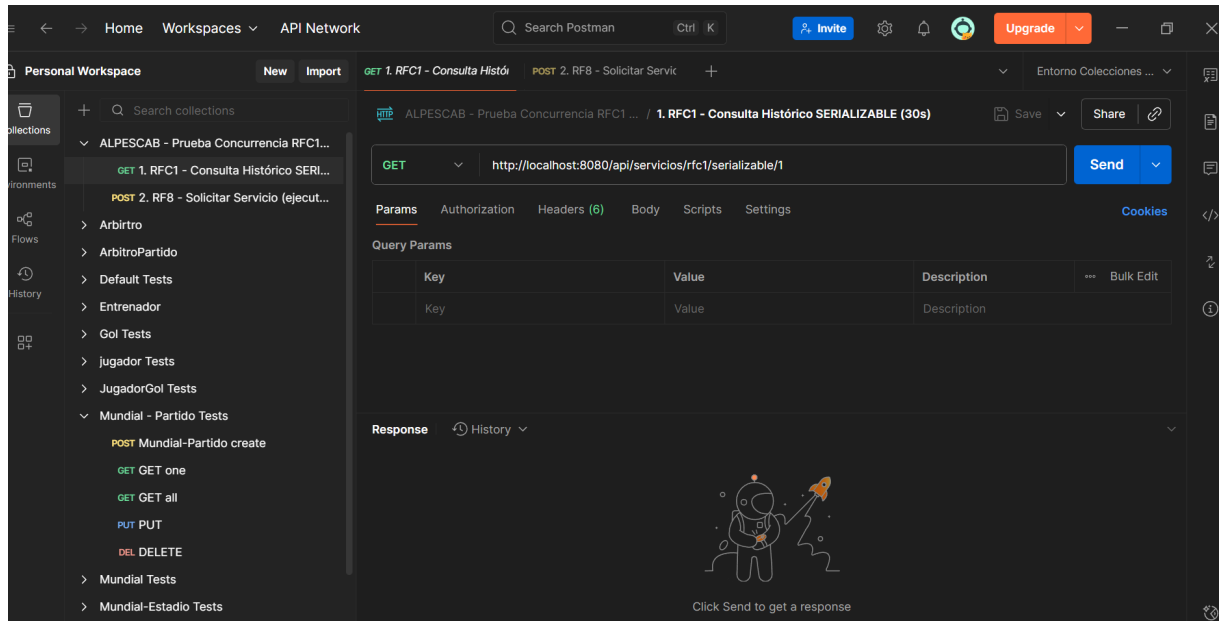
Rodriguez Paez, Maria Camila

PUNTO 4

Escenario de concurrencia probado:

Se ejecutó RFC1 con nivel de aislamiento SERIALIZABLE, que realiza dos consultas separadas por 30 segundos. Durante ese intervalo, se ejecutó RF8 que insertó un nuevo servicio en la base de datos.

pruebas:



RFC1	Tiempo	RF8
Inicia transacción y realiza primera consulta	t1	
Espera (sleep de 30 segundos)	t2	
	t3	Inicia transacción y solicita nuevo servicio

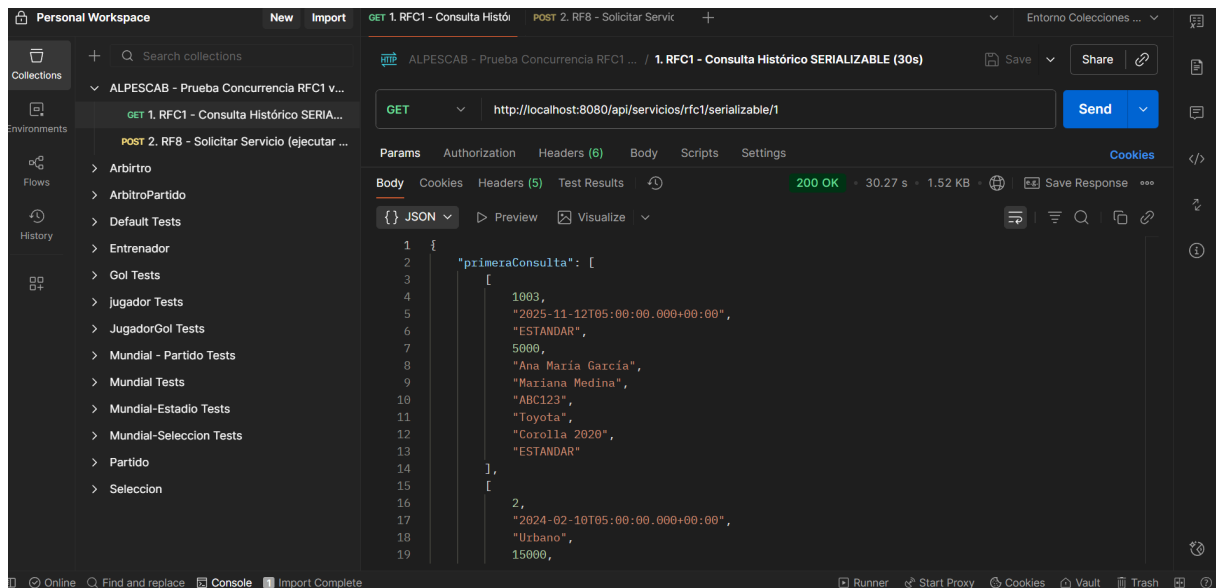
	t4	Registra el servicio y finaliza transacción
Realiza segunda consulta y reporta resultados	t5	

Descripción de lo sucedido:

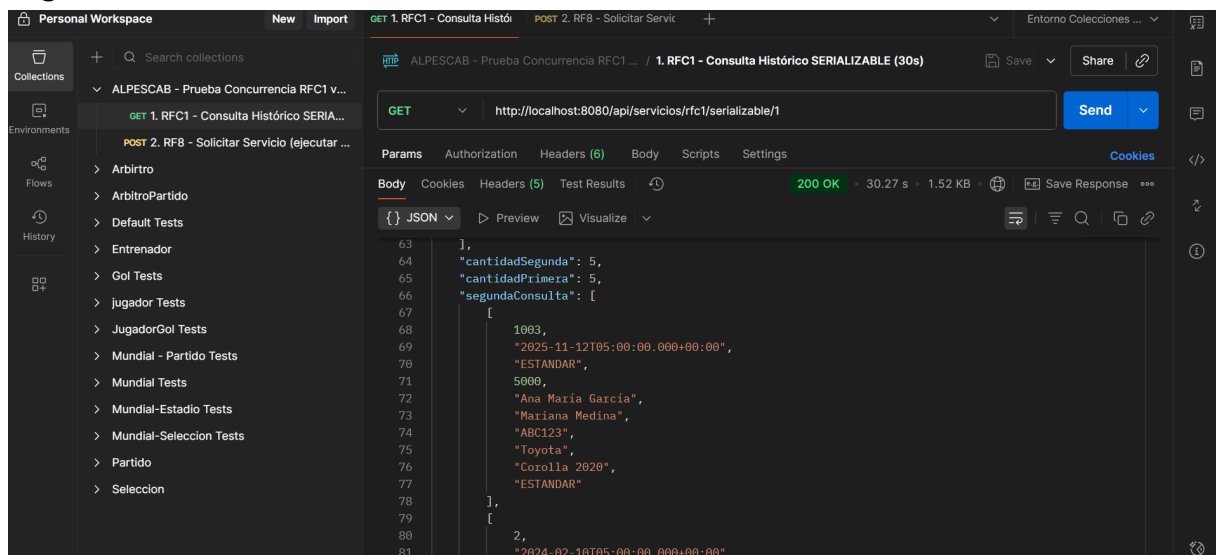
No, el componente que implementa RFC1 no necesita esperar a que termine la ejecución de RF8 para registrar la orden de servicio, ni RF8 necesita esperar a que termine RFC1 para registrar una nueva orden. Bajo el nivel de aislamiento **SERIALIZABLE**, cada transacción (RFC1 y RF8) opera como si fuera la única ejecutándose; es decir, cada una ve una "foto" de los datos confirmados antes de que comenzó su propia transacción.

RESULTADO PRESENTADO POR RFC1

Primera consulta



Segunda Consulta



Conclusión:

El nivel de aislamiento **SERIALIZABLE** garantiza que una transacción vea una "fotografía" consistente de los datos desde el inicio hasta el fin de la transacción, aislando completamente los cambios concurrentes realizados por otras transacciones. Esto previene anomalías como lecturas no repetibles y lecturas fantasma.

PUNTO 5

pruebas

The screenshot shows the Postman interface for a POST request. The request is named "2. RF8 - Solicitar Serv" and is sent to the URL "http://localhost:8080/api/servicios/solicitar". The response is a 200 OK status with a response time of 137 ms and a body size of 223 B. The response body is displayed in the "Body" tab, showing a single line of text: "1 ✓ Servicio solicitado exitosamente. ID del servicio: 1021".

GET 1. RFC1 - READ COMMIT | **POST 2. RF8 - Solicitar Serv** +

ALPES CAB - Punto 5 READ COMMITTED / 2. RF8 - Solicitar Servicio (ejecutar 5s después)

POST http://localhost:8080/api/servicios/solicitar Send

Params Authorization Headers (9) Body • Scripts Settings Cookies

Query Params

Key	Value	Description
Key	Value	Description

Body Cookies Headers (5) Test Results 200 OK • 137 ms • 223 B Save Response

Raw Preview Visualize

1 ✓ Servicio solicitado exitosamente. ID del servicio: 1021

The screenshot shows the Postman interface for a GET request. The request is named "1. RFC1 - READ COMMITTED (30s)" and is sent to the URL "http://localhost:8080/api/servicios/rfc1/readcommitted/1". The response is a 200 OK status with a response time of 30.05 s and a body size of 1.52 KB. The response body is displayed in the "Body" tab, showing a JSON object with the following structure:

```
{  "cantidadSegunda": 5,  "cantidadPrimera": 5,  "segundaConsulta": [    {      "id": 1003,      "fecha": "2025-11-12T05:00:00.000+00:00",      "estado": "ESTANDAR",      "cantidad": 5000,      "nombre": "Ana María García",      "apellido": "Mariana Medina"    }  ]}
```

DESDE POSTMAN NO FUNCIONA PQ SON LA MISMA SESIÓN, TOCA HACERLO DESDE VS CODE COMO EN EL TALLER CON DOS CONEXIONES DIFERENTES

LÍNEA DEL TIEMPO - PUNTO 5 (READ COMMITTED)

RFC1 (Serializable)	Tiempo	RF8 (Read Committed)
Inicia transacción y realiza primera consulta	t1	
Espera (sleep de 30 segundos)	t2	
	t3	Inicia transacción y solicita nuevo servicio
	t4	Registra el servicio y finaliza transacción
Realiza segunda consulta y reporta resultados	t5	

DESCRIPCIÓN DE LO SUCEDIDO

¿RFC1 debió esperar a que terminara RF8?

-NO. RFC1 NO tuvo que esperar a que terminara RF8. Ambas transacciones se ejecutaron de manera concurrente e independiente.

En el nivel de aislamiento READ COMMITTED:

RFC1 y RF8 pueden ejecutarse al mismo tiempo sin bloquearse

Cada transacción lee los datos confirmados (COMMITTED) más recientes

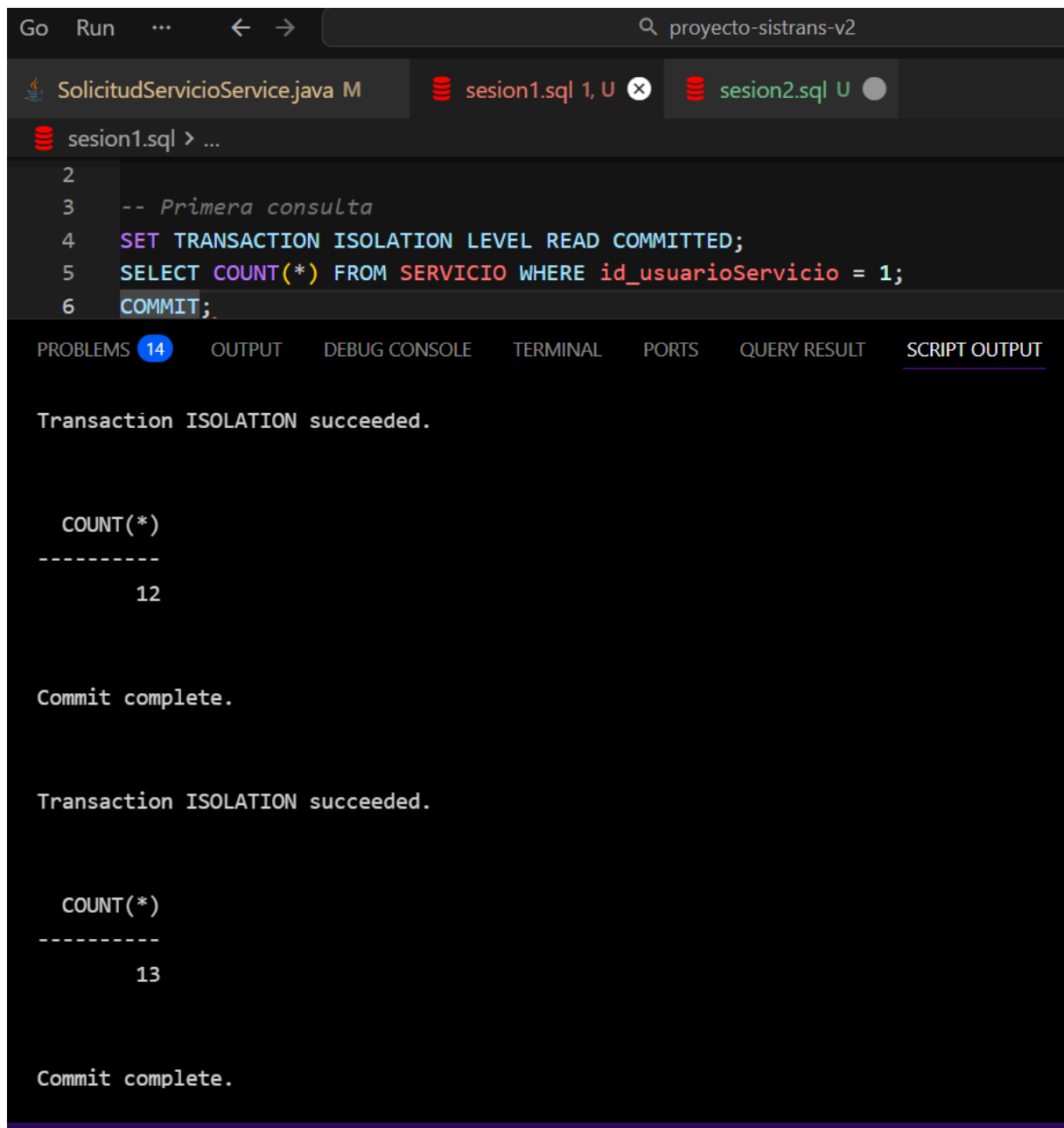
No hay bloqueos entre lecturas (SELECT) e inserciones (INSERT)

RFC1 hizo su primera lectura, luego RF8 insertó y confirmó el servicio, y finalmente

RFC1 hizo su segunda lectura en una nueva transacción, por eso pudo ver el cambio.

RESULTADO PRESENTADO POR RFC1

RESULTADOS PRIMERA CONSULTA



The screenshot shows an IDE window with a dark theme. At the top, there's a search bar with 'proyecto-sistrans-v2'. Below it, a tab bar shows 'SolicitudServicioService.java M', 'sesion1.sql 1, U', and 'sesion2.sql U'. The 'sesion1.sql' tab is active, showing a SQL script with line numbers 2 to 6. The script sets the transaction isolation level to READ COMMITTED and runs a query to count records in the 'SERVICIO' table where 'id_usuarioServicio' is 1. Below the script, the 'SCRIPT OUTPUT' tab is selected, displaying the execution results. The output shows two successful transactions, each with a 'COUNT(*)' result of 12 and 13 respectively, followed by 'Commit complete.'.

```
2
3  -- Primera consulta
4  SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
5  SELECT COUNT(*) FROM SERVICIO WHERE id_usuarioServicio = 1;
6  COMMIT;
```

Transaction ISOLATION succeeded.

COUNT(*)

12

Commit complete.

Transaction ISOLATION succeeded.

COUNT(*)

13

Commit complete.

RESULTADOS SEGUNDA CONSULTA

```
SolicitudServicioService.java M  sesion1.sql 1, U  sesion2.sql U ●
sesion2.sql > ...
22 VALUES ('Carrera 7', -74.05, 4.68, 1, 1025, 1);
23
24 INSERT INTO PUNTO (direccion, longitud, latitud, orden, id_servicio,
25 VALUES ('Calle 26', -74.09, 4.64, 2, 1025, 1);
26
```

PROBLEMS 14 OUTPUT DEBUG CONSOLE TERMINAL PORTS QUERY RESULT SCRIPT OUTPUT

```
MAX(ID)
-----
1034

1 row inserted.

1 row inserted.

Commit complete.

COUNT(*)
-----
13
```

CONCLUSIÓN - DIFERENCIA CON PUNTO 4

Aspecto	SERIALIZABLE	READ COMMITTED
Primera lectura	5 servicios	5 servicios
RF8 ejecuta	Inserta servicio	Inserta servicio
Segunda lectura	5 servicios (NO ve RF8)	6 servicios (SÍ ve RF8)

Fenómeno	Previene lecturas no repetibles	Permite lecturas no repetibles
-----------------	--	---------------------------------------

READ COMMITTED permite que se vean los cambios confirmados por otras transacciones concurrentes, causando lecturas no repetibles (non-repeatable reads). Esto demuestra menor aislamiento que **SERIALIZABLE**.