

## **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:

The screenshot shows the Postman interface with two parallel requests. Request 1 (GET) is titled "RFC1 - Consulta Histórico SERIALIZABLE" and Request 2 (POST) is titled "RF8 - Solicitar Servicio (ejecutar)". Both requests are directed to `http://localhost:8080/api/servicios/rfc1/serializable/1`. The "Params" tab is selected for both requests. A cartoon character icon is visible in the bottom right corner.

The screenshot shows the Postman interface with a single POST request titled "RF8 - Solicitar Servicio (ejecutar 5s después)". The URL is `http://localhost:8080/api/servicios/solicitar`. The "Params" tab is selected. In the response section, the status is "200 OK" with a response time of 138 ms and a body size of 223 B. The response body contains the message: "Servicio solicitado exitosamente. ID del servicio: 1019".

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

	t4	<b>Registra el servicio y finaliza transacción</b>
<b>Realiza segunda consulta y reporta resultados</b>	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**

```

1  {
2   "primeraConsulta": [
3     [
4       1003,
5       "2025-11-12T05:00:00.000+00:00",
6       "ESTANDAR",
7       5000,
8       "Ana Maria Garcia",
9       "Mariana Medina",
10      "ABC123",
11      "Toyota",
12      "Corolla 2020",
13      "ESTANDAR"
14    ],
15    [
16      2,
17      "2024-02-10T05:00:00.000+00:00",
18      "Urbano",
19      15000,

```

## Segunda Consulta

```

63  ],
64  "cantidadSegunda": 5,
65  "cantidadPrimera": 5,
66  "segundaConsulta": [
67    [
68      1003,
69      "2025-11-12T05:00:00.000+00:00",
70      "ESTANDAR",
71      5000,
72      "Ana Maria Garcia",
73      "Mariana Medina",
74      "ABC123",
75      "Toyota",
76      "Corolla 2020",
77      "ESTANDAR"
78    ],
79    [
80      2,
81      "2024-02-10T05:00:00.000+00:00",

```

## 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 with a collection named "ALPES CAB - Punto 5 READ COMMITTED". A POST request is being made to the endpoint `http://localhost:8080/api/servicios/solicitar`. The "Params" tab is selected. The response status is 200 OK, with a response time of 137 ms and a response size of 223 B. The response body contains the message: "Servicio solicitado exitosamente. ID del servicio: 1021".

The screenshot shows the Postman interface with a collection named "ALPES CAB - Punto 5 READ COMMITTED". A GET request is being made to the endpoint `http://localhost:8080/api/servicios/rfc1/readcommitted/1`. The "Params" tab is selected. The response status is 200 OK, with a response time of 30.05 s and a response size of 1.52 KB. The response body is displayed in JSON format, showing a complex nested structure of data.

**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

```
Go Run ... ← → 🔍 proyecto-sistrans-v2
SolicitudServicioService.java M sesion1.sql 1, U × sesion2.sql U
sesion1.sql > ...
2
3 -- Primera consulta
4 SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
5 SELECT COUNT(*) FROM SERVICIO WHERE id_usuarioServicio = 1;
6 COMMIT;

PROBLEMS 14 OUTPUT DEBUG CONSOLE TERMINAL PORTS QUERY RESULT SCRIPT OUTPUT

Transaction ISOLATION succeeded.

COUNT(*)
-----
12

Commit complete.

Transaction ISOLATION succeeded.

COUNT(*)
-----
13

Commit complete.
```

## RESULTADOS SEGUNDA CONSULTA

The screenshot shows a Java IDE interface with several tabs at the top: 'SolicitudServicioService.java M', 'sesion1.sql 1, U', 'sesion2.sql U' (with a circular progress icon), and 'sesion2.sql > ...'. Below the tabs, code is displayed in a dark-themed editor:

```
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
```

Below the code, there are several output sections:

- MAX(ID)**  
-----  
**1034**
- 1 row inserted.**
- 1 row inserted.**
- Commit complete.**
- COUNT(\*)**  
-----  
**13**

## CONCLUSIÓN - DIFERENCIA CON PUNTO 4

Aspecto	SERIALIZABLE	READ COMMITTED
Primera lectura	<b>5 servicios</b>	<b>5 servicios</b>
RF8 ejecuta	<b>Inserta servicio</b>	<b>Inserta servicio</b>
Segunda lectura	<b>5 servicios (NO ve RF8)</b>	<b>6 servicios (Sí ve RF8)</b>

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**.