

# TRANSACTIONS ASSIGNMENT

- Task A: Dirty Read (READ UNCOMMITTED vs READ COMMITTED)

Isolation Level: READ UNCOMMITTED:

```
49
50 -- Task A: Dirty Read
51 BEGIN TRANSACTION;
52
53 UPDATE Rooms
54 SET Price = 100.00
55 WHERE RoomID = 1;
56
57 SELECT * FROM Rooms WHERE RoomID = 1;
```

0 %

Results Messages

	RoomID	RoomNumber	Type	Price
1	1	101	Single	100.00

```
SQLQuery2.sql - SF-...PO-005\Install (07)) SQLQuery1.sql - SF-...PO-005\Install (06)
1 -- 1. Using READ UNCOMMITTED
2
3 SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;
4 SELECT * FROM Rooms WHERE RoomID = 1;
5
```

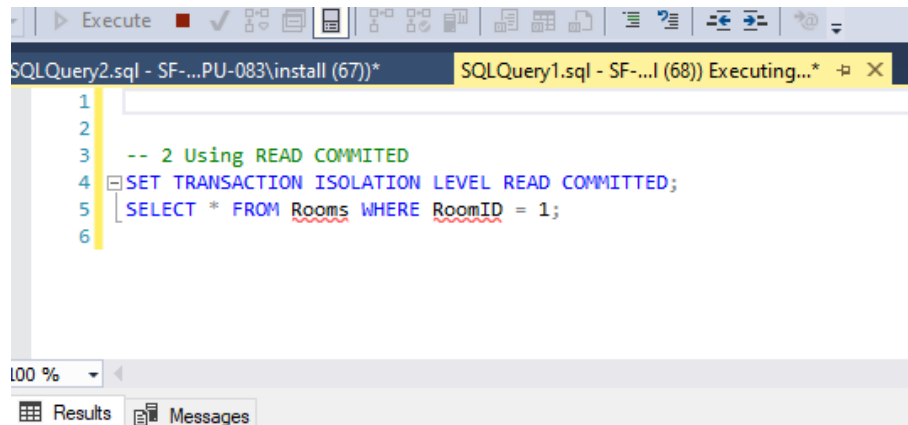
100 %

Results Messages

	RoomID	RoomNumber	Type	Price
1	1	101	Single	100.00

Since READ UNCOMMITTED allows dirty reads, **Session 2 sees the updated price** even though it is uncommitted.

Isolation Level: READ COMMITTED:



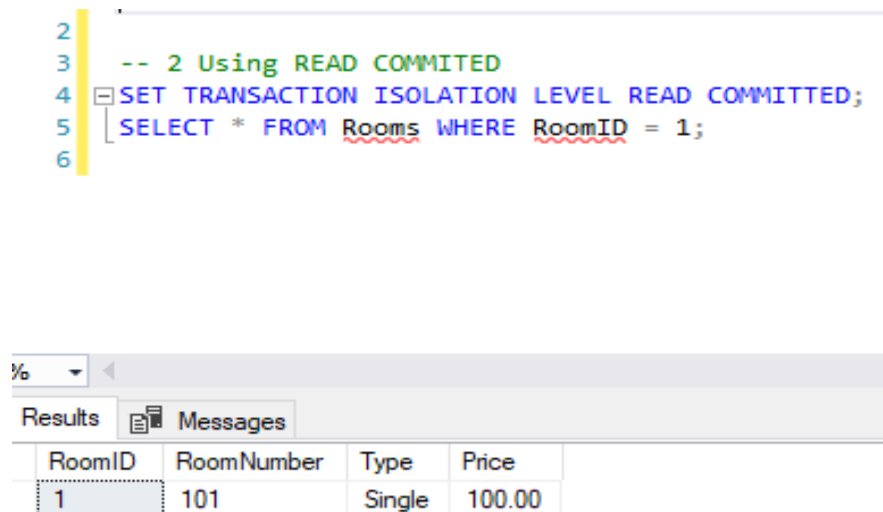
The screenshot shows a SQL Server Enterprise Manager window with two tabs. The active tab is 'SQLQuery2.sql - SF-...PU-083\install (67))\*'. The code in the window is as follows:

```
1  
2  
3  -- 2 Using READ COMMITTED  
4  SET TRANSACTION ISOLATION LEVEL READ COMMITTED;  
5  SELECT * FROM Rooms WHERE RoomID = 1;  
6
```

The bottom of the window shows a 'Results' tab and a 'Messages' tab. The zoom level is set to 100%.

If **Session 1** has an **open transaction (not committed or rolled back)**, **Session 2** must **wait** until the transaction is completed. READ COMMITTED in **Session 2** waits for the exclusive lock to be released before reading.

After commit-



The screenshot shows the same SQL Server Enterprise Manager window as before, but now the 'Results' tab is active. The query window still shows the same code:

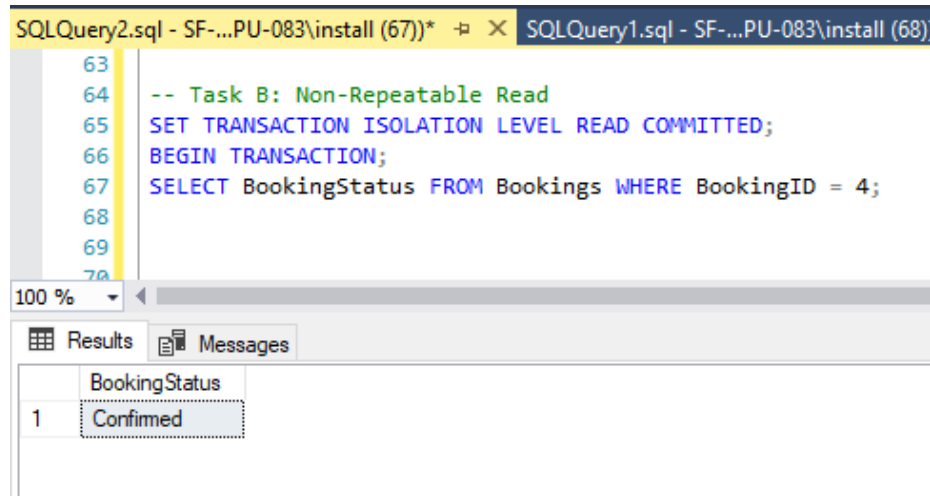
```
2  
3  -- 2 Using READ COMMITTED  
4  SET TRANSACTION ISOLATION LEVEL READ COMMITTED;  
5  SELECT * FROM Rooms WHERE RoomID = 1;  
6
```

The 'Results' tab displays the following data:

RoomID	RoomNumber	Type	Price
1	101	Single	100.00

- Task B: Non-Repeatable Read (READ COMMITTED vs REPEATABLE READ)

Isolation Level: READ COMMITTED:

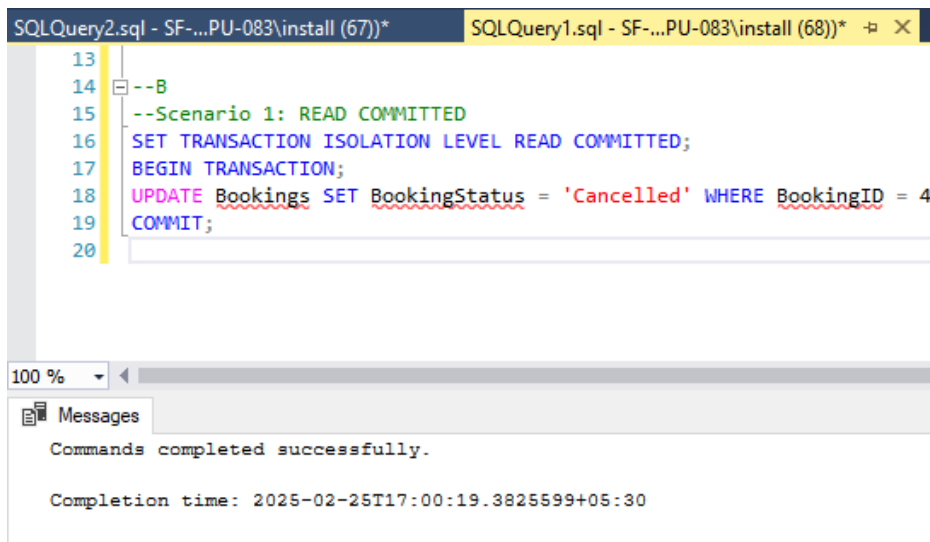


The screenshot shows a SQL Server Enterprise Manager interface. The top pane displays a query window with the following SQL code:

```
63  
64 -- Task B: Non-Repeatable Read  
65 SET TRANSACTION ISOLATION LEVEL READ COMMITTED;  
66 BEGIN TRANSACTION;  
67 SELECT BookingStatus FROM Bookings WHERE BookingID = 4;  
68  
69  
70
```

The bottom pane shows the 'Results' tab with a single row of data:

BookingStatus
1 Confirmed



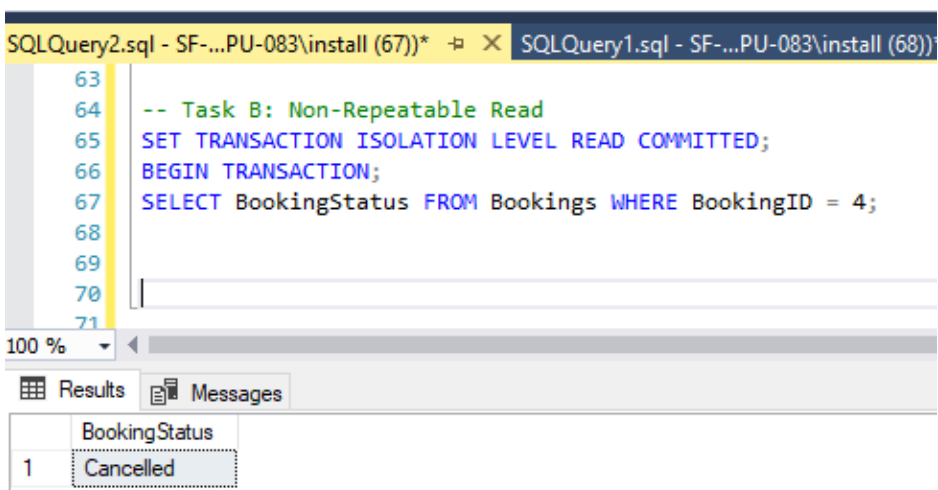
The screenshot shows a SQL Server Enterprise Manager interface. The top pane displays a query window with the following SQL code:

```
13  
14 --B  
15 --Scenario 1: READ COMMITTED  
16 SET TRANSACTION ISOLATION LEVEL READ COMMITTED;  
17 BEGIN TRANSACTION;  
18 UPDATE Bookings SET BookingStatus = 'Cancelled' WHERE BookingID = 4;  
19 COMMIT;  
20
```

The bottom pane shows the 'Messages' tab with the following text:

Commands completed successfully.

Completion time: 2025-02-25T17:00:19.3825599+05:30



The screenshot shows a SQL Server Enterprise Manager interface. The top pane displays a query window with the following SQL code:

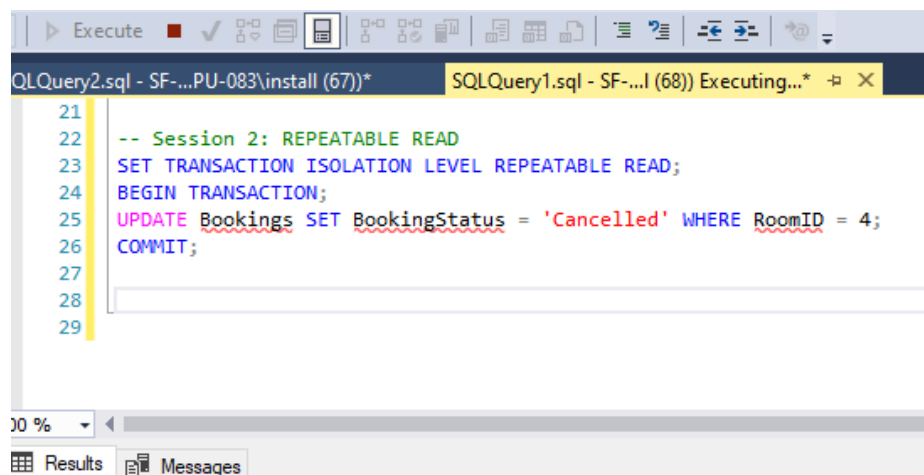
```
63  
64 -- Task B: Non-Repeatable Read  
65 SET TRANSACTION ISOLATION LEVEL READ COMMITTED;  
66 BEGIN TRANSACTION;  
67 SELECT BookingStatus FROM Bookings WHERE BookingID = 4;  
68  
69  
70  
71
```

The bottom pane shows the 'Results' tab with a single row of data:

BookingStatus
1 Cancelled

The **READ COMMITTED** isolation level ensures that **Session 1** can read **committed data**. Once **Session 2** commits the change (status update to 'Cancelled'), **Session 1** will see the new status immediately in its next read.

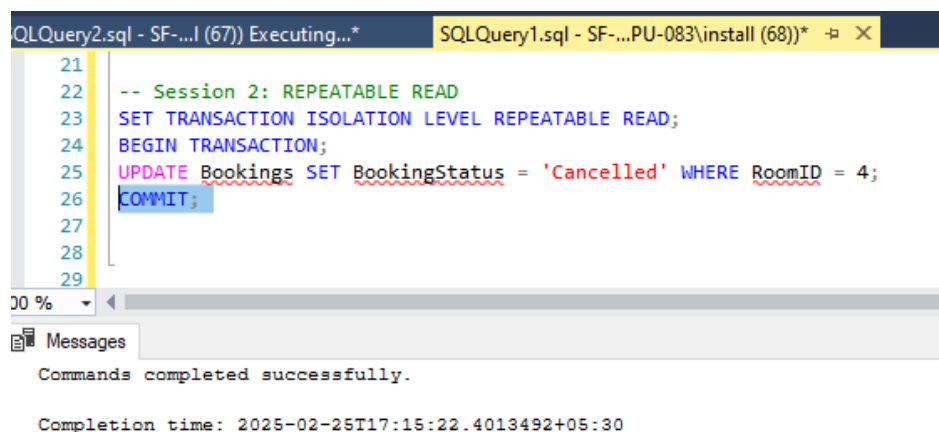
Isolation Level: REPEATABLE READ:



```
21
22 -- Session 2: REPEATABLE READ
23 SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;
24 BEGIN TRANSACTION;
25 UPDATE Bookings SET BookingStatus = 'Cancelled' WHERE RoomID = 4;
26 COMMIT;
27
28
29
```

The reason **Session 2**'s update statement is still running (blocked) is due to the fact that **REPEATABLE READ** isolation level ensures **consistent reads** by locking the data for the duration of **Session 1**'s transaction. Once **Session 1** commits, **Session 2** will be able to acquire the lock and proceed with its update.

After commit in session 1-



```
21
22 -- Session 2: REPEATABLE READ
23 SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;
24 BEGIN TRANSACTION;
25 UPDATE Bookings SET BookingStatus = 'Cancelled' WHERE RoomID = 4;
26 COMMIT;
27
28
29
```

Commands completed successfully.

Completion time: 2025-02-25T17:15:22.4013492+05:30

- Task C: Phantom Read (REPEATABLE READ vs SERIALIZABLE)

Isolation Level: REPEATABLE READ:

```
80
81  -- Task C: Phantom Read
82
83  SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;
84  BEGIN TRANSACTION;
85
86  SELECT COUNT(*) FROM Rooms WHERE Type = 'Suite';
87
```

100 %

Results Messages

	(No column name)
1	3

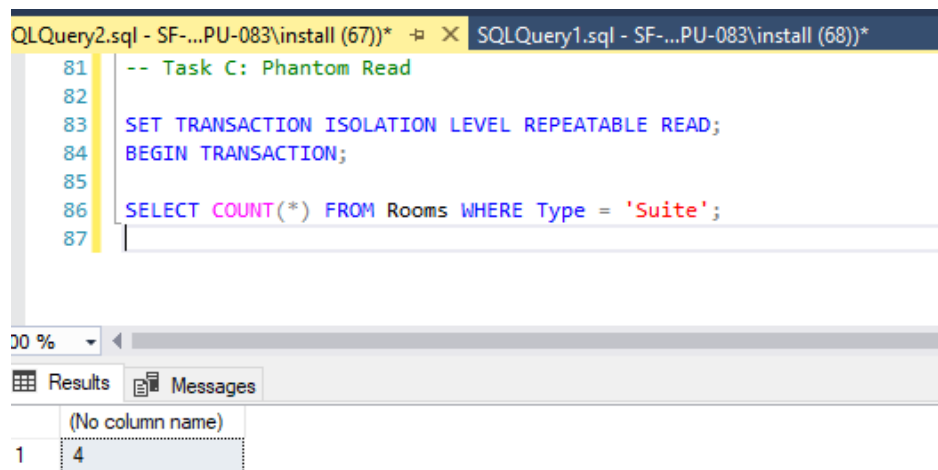
```
SQLQuery2.sql - SF-...PU-083\install (67))*  SQLQuery1.sql - SF-...PU-083\install (68))*
30
31
32  -- C
33  -- Session 1: REPEATABLE READ
34  SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;
35  BEGIN TRANSACTION;
36
37  INSERT INTO Rooms (RoomNumber, Type, Price) VALUES ('111', 'Suite', 150);
38  COMMIT;
```

100 %

Messages

Commands completed successfully.

Completion time: 2025-02-25T17:22:32.0027143+05:30



SQLQuery2.sql - SF-...PU-083\install (67))\* SQLQuery1.sql - SF-...PU-083\install (68))\*

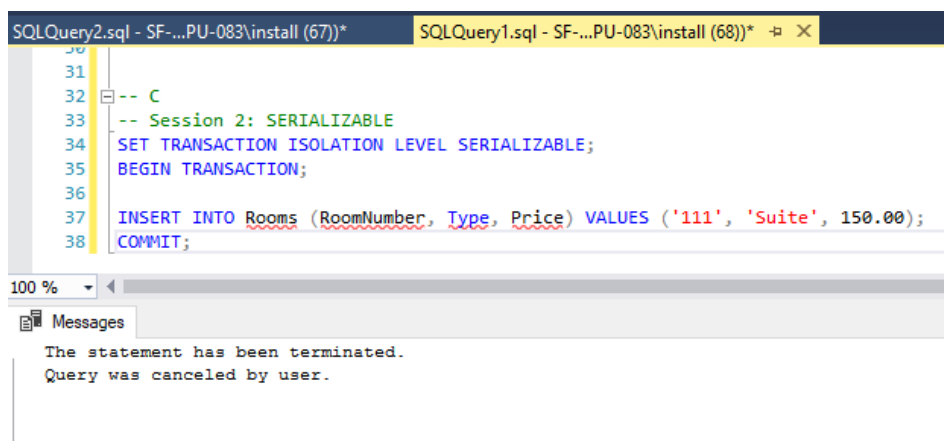
```
81 -- Task C: Phantom Read
82
83 SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;
84 BEGIN TRANSACTION;
85
86 SELECT COUNT(*) FROM Rooms WHERE Type = 'Suite';
87
```

100 %

Results Messages

	(No column name)
1	4

Isolation Level: SERIALIZABLE:



SQLQuery2.sql - SF-...PU-083\install (67))\* SQLQuery1.sql - SF-...PU-083\install (68))\*

```
30
31
32 -- C
33 -- Session 2: SERIALIZABLE
34 SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
35 BEGIN TRANSACTION;
36
37 INSERT INTO Rooms (RoomNumber, Type, Price) VALUES ('111', 'Suite', 150.00);
38 COMMIT;
```

100 %

Messages

The statement has been terminated.  
Query was canceled by user.

**Session 2** is **blocked** from inserting the new row into the **'Suite'** category because the **SERIALIZABLE** lock held by **Session 1** prevents any changes to rows that could affect the result of the query in **Session 1**.