Dirty Reads –  
When a transaction is allowed to read a row that has been modified by an another transaction which is not committed yet that time *Dirty Reads* occurred. It is mainly occurred because of multiple transaction at a time which is not committed.

**Example –**

**Table –** Record

|  |  |  |
| --- | --- | --- |
| **ID** | **Cus\_Name** | **Balance** |
| 1 | S Adam | 100 |
| 2 | Zee Young | 150 |

**Transaction –**  
Transfer 10 from S Adam account to Zee Young account:

**Input:**

BEGIN TRY

BEGIN TRANSACTION

UPDATE Table SET Balance = Balance - 10 WHERE ID=1;

UPDATE Table SET Balance = Balance + 10 WHERE ID='C';

COMMIT TRANSACTION

PRINT 'Committed'

END TRY

BEGIN CATCH

ROLLBACK TRANSACTION

PRINT 'Not Committed'

END CATCH

**Output:**

Not committed

By executing the above query the output will be “Not committed” because there is an error, there is no ID=C. So at that time if we want to execute another transaction with that row that time **Dirty Reads** occurs. There is no partial commitment if both the UPDATE query succeed only then the output will be “Committed”.

Before Execution:

**Table –** Record

|  |  |  |
| --- | --- | --- |
| **ID** | **Cus\_Name** | **Balance** |
| 1 | S Adam | 100 |
| 2 | Zee Young | 150 |

After Execution:

**Input:**

BEGIN TRY

BEGIN TRANSACTION

UPDATE Table SET Balance = Balance - 10 WHERE ID=1;

UPDATE Table SET Balance = Balance + 10 WHERE ID='C';

COMMIT TRANSACTION

PRINT 'Committed'

END TRY

BEGIN CATCH

ROLLBACK TRANSACTION

PRINT 'Not Committed'

END CATCH

**Output:**

(1row affected)

(0row affected)

Not Committed

Note that if we put a valid ID first transaction result will come out as committed or 1 row effected but 2nd one will not be affected.

**Explanation –**  
If we have a ticket booking system and One Customer is trying to book a ticket at that time available number of the ticket is 10, before completing the payment, the Second Customer wants to book a ticket that time this 2nd transaction will show the second customer that the number of the available tickets is 9. The twist is here if the first customer does not have sufficient fund in his debit card or in his wallet then the 1st transaction will Rollback, that time 9 seat available which is read by the 2nd transaction is *Dirty Read*.

**Example:**  
**Available ticket:** For 1st customer

* **1st Step –**
* **Input:**
* -- Transaction 1

Select \*from Bus\_ticket;

**Output:**

|  |  |  |
| --- | --- | --- |
| **ID** | **Bus\_Name** | **Available\_Seat** |
| 1 | KA0017 | 10 |

* **2nd Step –**  
  Booking time for 1st customer
* **Input:**
* --Transaction 1
* BEING Transaction
* UPDATE Bus\_Ticket set Available\_Seat=9
* WHERE ID=1
* --Payment for Transaction 1
* Waitfor Delay '00.00.30'

Rollback transaction

**Available ticket:** For 2nd customer while 1st customer is paying (waiting at most 30s) for the ticket.

* **3rd Step –**
* **Input:**
* -- Transaction 1
* set transaction isolation level read uncommitted

Select \*from Bus\_ticket where ID=1;

**Output:**

|  |  |  |
| --- | --- | --- |
| **ID** | **Bus\_Name** | **Available\_Seat** |
| 1 | KA0017 | 9 |

Note that during the payment of 1st customer 2nd transaction read it 9 seat is available if some how 1st transaction Rollback then available seat 9 that is Dirty read data. After rollback of the 1st transaction available seat is 10 again. 2nd and 3rd step happening at the same time.

Actually available seat after rollback of transaction 1:

|  |  |  |
| --- | --- | --- |
| **ID** | **Bus\_Name** | **Available\_Seat** |
| 1 | KA0017 | 10 |

# Non-Repeatable Read

The Non-Repeatable Read Concurrency Problem happens in SQL Server when one transaction reads the same data twice while another transaction updates that data in between the first and second read of the first transaction.

In order to solve the **Non-Repeatable Read Problem in SQL Server**, we need to use either **Repeatable Read Transaction Isolation Level** or any other higher isolation level

# Phantom reads

Getting a phantom row

A Phantom read occurs when one user is repeating a read operation on the same records, but has new records in the results set because another user (or transaction) has INSERTED a new row between the reads of user1.