Transaction Isolation Levels (ODBC)

01/19/2017 • 4 minutes to read • +1

Transaction isolation levels are a measure of the extent to which transaction isolation succeeds. In particular, transaction isolation levels are defined by the presence or absence of the following phenomena:

- **Dirty Reads** A *dirty read* occurs when a transaction reads data that has not yet been committed. For example, suppose transaction 1 updates a row. Transaction 2 reads the updated row before transaction 1 commits the update. If transaction 1 rolls back the change, transaction 2 will have read data that is considered never to have existed.

- **Nonrepeatable Reads** A *nonrepeatable read* occurs when a transaction reads the same row twice but gets different data each time. For example, suppose transaction 1 reads a row. Transaction 2 updates or deletes that row and commits the update or delete. If transaction 1 rereads the row, it retrieves different row values or discovers that the row has been deleted.

- **Phantoms** A *phantom* is a row that matches the search criteria but is not initially seen. For example, suppose transaction 1 reads a set of rows that satisfy some search criteria. Transaction 2 generates a new row (through either an update or an insert) that matches the search criteria for transaction 1. If transaction 1 reexecutes the statement that reads the rows, it gets a different set of rows.

The four transaction isolation levels (as defined by SQL-92) are defined in terms of these phenomena. In the following table, an "X" marks each phenomenon that can occur.

Transaction isolation level	Dirty reads	Nonrepeatable reads	Phantoms
Read uncommitted	X	X	X
Read committed	--	X	X
Repeatable read	--	--	X
Serializable	--	--	--

The following table describes simple ways that a DBMS might implement the transaction isolation levels.

ⓘ **Important**

Most DBMSs use more complex schemes than these to increase concurrency. These examples are provided for illustration purposes only. In particular, ODBC does not prescribe how particular DBMSs isolate transactions from each other.

Transaction isolation	Possible implementation
Read uncommitted	Transactions are not isolated from each other. If the DBMS supports other transaction isolation levels, it ignores whatever mechanism it uses to implement those levels. So that they do not adversely affect other transactions, transactions running at the Read Uncommitted level are usually read-only.
Read committed	The transaction waits until rows write-locked by other transactions are unlocked; this prevents it from reading any "dirty" data. The transaction holds a read lock (if it only reads the row) or write lock (if it updates or deletes the row) on the current row to prevent other transactions from updating or deleting it. The transaction releases read locks when it moves off the current row. It holds write locks until it is committed or rolled back.
Repeatable read	The transaction waits until rows write-locked by other transactions are unlocked; this prevents it from reading any "dirty" data. The transaction holds read locks on all rows it returns to the application and write locks on all rows it inserts, updates, or deletes. For example, if the transaction includes the SQL statement **SELECT * FROM Orders**, the transaction read-locks rows as the application fetches them. If the transaction includes the SQL statement **DELETE FROM Orders WHERE Status = 'CLOSED'**, the transaction write-locks rows as it deletes them. Because other transactions cannot update or delete these rows, the current transaction avoids any nonrepeatable reads. The transaction releases its locks when it is committed or rolled back.

Transaction isolation	Possible implementation
Serializable	The transaction waits until rows write-locked by other transactions are unlocked; this prevents it from reading any "dirty" data. The transaction holds a read lock (if it only reads rows) or write lock (if it can update or delete rows) on the range of rows it affects. For example, if the transaction includes the SQL statement **SELECT * FROM Orders**, the range is the entire Orders table; the transaction read-locks the table and does not allow any new rows to be inserted into it. If the transaction includes the SQL statement **DELETE FROM Orders WHERE Status = 'CLOSED'**, the range is all rows with a Status of "CLOSED"; the transaction write-locks all rows in the Orders table with a Status of "CLOSED" and does not allow any rows to be inserted or updated such that the resulting row has a Status of "CLOSED". Because other transactions cannot update or delete the rows in the range, the current transaction avoids any nonrepeatable reads. Because other transactions cannot insert any rows in the range, the current transaction avoids any phantoms. The transaction releases its lock when it is committed or rolled back.

It is important to note that the transaction isolation level does not affect a transaction's ability to see its own changes; transactions can always see any changes they make. For example, a transaction might consist of two **UPDATE** statements, the first of which raises the pay of all employees by 10 percent and the second of which sets the pay of any employees over some maximum amount to that amount. This succeeds as a single transaction only because the second **UPDATE** statement can see the results of the first.

Is this page helpful?

Yes No

Recommended content

Understanding isolation levels - SQL Server
Understanding isolation levels

Concurrency Control - SQL Server
Concurrency Control

SET TRANSACTION ISOLATION

Clustered and Nonclustered

LEVEL (Transact-SQL) - SQL Server

SET TRANSACTION ISOLATION LEVEL (Transact-SQL)

Indexes Described - SQL Server

Clustered and Nonclustered Indexes Described

Show more