# Isolation Levels – SQL Facilities for Concurrency and Recovery

The levels of transaction isolation in DBMS determine how the concurrently running transactions behave and, therefore, ensure data consistency with performance being even. There are four basic levels- Read Uncommitted, Read Committed, Repeatable Read, and Serializable that provide different degrees of data protection from providing fast access with possible incoherence and strict accuracy at the cost of performance. It depends upon choosing the right one based on whether the need is speed or data integrity.

**What is the Transaction Isolation Level?**

In a database management system, transaction isolation levels define the degree to which the operations in one transaction are isolated from the operations of other concurrent transactions. In other words, it defines how and when the changes made by one transaction are visible to others to assure data consistency and integrity.

As we know, to maintain consistency in a database, it follows ACID properties. Among these four properties (Atomicity, Consistency, Isolation, and Durability) Isolation determines how transaction integrity is visible to other users and systems. It means that a transaction should take place in a system in such a way that it is the only transaction that is accessing the resources in a database system.

Isolation levels define the degree to which a transaction must be isolated from the data modifications made by any other transaction in the database system.

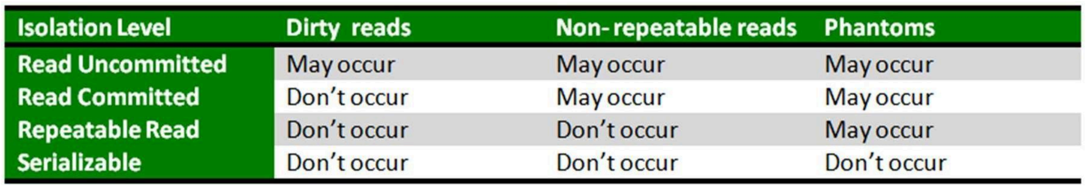
**A transaction isolation level is defined by the following phenomena:**

- Dirty Read – A Dirty read is a situation when a transaction reads data that has not yet been committed. For example, Let's say transaction 1 updates a row and leaves it uncommitted, meanwhile, Transaction 2 reads the updated row. If transaction 1 rolls back the change, transaction 2 will have read data that is considered never to have existed.
- Non Repeatable read – Non-repeatable read occurs when a transaction reads the same row twice and gets a different value each time. For example, suppose transaction T1 reads data. Due to concurrency, another transaction T2 updates the same data and commit, Now if transaction T1 rereads the same data, it will retrieve a different value.
- Phantom Read – Phantom Read occurs when two same queries are executed, but the rows retrieved by the two, are different. For example, suppose transaction T1 retrieves a set of rows that satisfy some search criteria. Now, Transaction T2 generates some new rows that match the search criteria for Transaction T1. If transaction T1 re-executes the statement that reads the rows, it gets a different set of rows this time.

**Based on these phenomena, The SQL standard defines four isolation levels:**

1. Read Uncommitted – Read Uncommitted is the lowest isolation level. In this level, one transaction may read not yet committed changes made by other transactions, thereby allowing dirty reads. At this level, transactions are not isolated from each other.
2. Read Committed – This isolation level guarantees that any data read is committed at the moment it is read. Thus it does not allow dirty read. The transaction holds a read or write lock on the current row, and thus prevents other transactions from reading, updating, or deleting it.
3. Repeatable Read – This is the most restrictive isolation level. The transaction holds read locks on all rows it references and writes locks on referenced rows for update and delete actions. Since other transactions cannot read, update or delete these rows, consequently it avoids non-repeatable read.
4. Serializable – This is the highest isolation level. A *serializable* execution is guaranteed to be serializable. Serializable execution is defined to be an execution of operations in which concurrently executing transactions appears to be serially executing.

The Table given below clearly depicts the relationship between isolation levels, read phenomena, and locks:



| Isolation Level | Dirty reads | Non-repeatable reads | Phantoms |
|---|---|---|---|
| Read Uncommitted | May occur | May occur | May occur |
| Read Committed | Don't occur | May occur | May occur |
| Repeatable Read | Don't occur | Don't occur | May occur |
| Serializable | Don't occur | Don't occur | Don't occur |

Anomaly Serializable is not the same as Serializable. That is, it is necessary, but not sufficient that a Serializable schedule should be free of all three phenomena types. Transaction isolation levels are used in database management systems (DBMS) to control the level of interaction between concurrent transactions.

**The four standard isolation levels are:**

1. Read Uncommitted: This is the lowest level of isolation where a transaction can see uncommitted changes made by other transactions. This can result in dirty reads, non-repeatable reads, and phantom reads.
2. Read Committed: In this isolation level, a transaction can only see changes made by other committed transactions. This eliminates dirty reads but can still result in non-repeatable reads and phantom reads.
3. Repeatable Read: This isolation level guarantees that a transaction will see the same data throughout its duration, even if other transactions commit changes to

the data. However, phantom reads are still possible.

4. Serializable: This is the highest isolation level where a transaction is executed as if it were the only transaction in the system. All transactions must be executed sequentially, which ensures that there are no dirty reads, non-repeatable reads, or phantom reads.

The choice of isolation level depends on the specific requirements of the application. Higher isolation levels offer stronger data consistency but can also result in longer lock times and increased contention, leading to decreased concurrency and performance. Lower isolation levels provide more concurrency but can result in data inconsistencies. In addition to the standard isolation levels, some DBMS may also support additional custom isolation levels or features such as snapshot isolation and multi-version concurrency control that provide alternative solutions to the problems addressed by the standard isolation levels.

**Advantages of Transaction Isolation Levels**

- Improved concurrency: Transaction isolation levels can improve concurrency by allowing multiple transactions to run concurrently without interfering with each other.
- Control over data consistency: Isolation levels provide control over the level of data consistency required by a particular application.
- Reduced data anomalies: The use of isolation levels can reduce data anomalies such as dirty reads, non-repeatable reads, and phantom reads.
- Flexibility: The use of different isolation levels provides flexibility in designing applications that require different levels of data consistency.

**Disadvantages of Transaction Isolation Levels**

- Increased overhead: The use of isolation levels can increase overhead because the database management system must perform additional checks and acquire more locks.
- Decreased concurrency: Some isolation levels, such as Serializable, can decrease concurrency by requiring transactions to acquire more locks, which can lead to blocking.
- Limited support: Not all database management systems support all isolation levels, which can limit the portability of applications across different systems.
- Complexity: The use of different isolation levels can add complexity to the design of database applications, making them more difficult to implement and maintain.

## SQL Facilities for Concurrency and Recovery.

Concurrency control means that multiple transactions can be executed at the same time and then the interleaved logs occur. But there may be changes in transaction results so maintain the order of execution of those transactions.

During recovery, it would be very difficult for the recovery system to backtrack all the logs and then start recovering.

Recovery with concurrent transactions can be done in the following four ways.

1. Interaction with concurrency control
2. Transaction rollback
3. Checkpoints
4. Restart recovery

**Interaction with concurrency control :**

In this scheme, the recovery scheme depends greatly on the concurrency control scheme that is used. So, to rollback a failed transaction, we must undo the updates performed by the transaction.

**Transaction rollback :**

- In this scheme, we rollback a failed transaction by using the log.
- The system scans the log backward a failed transaction, for every log record found in the log the system restores the data item.

**Checkpoints :**

- Checkpoints is a process of saving a snapshot of the applications state so that it can restart from that point in case of failure.
- Checkpoint is a point of time at which a record is written onto the database form the buffers.
- Checkpoint shortens the recovery process.
- When it reaches the checkpoint, then the transaction will be updated into the database, and till that point, the entire log file will be removed from the file. Then the log file is updated with the new step of transaction till the next checkpoint and so on.
- The checkpoint is used to declare the point before which the DBMS was in the consistent state, and all the transactions were committed.

To ease this situation, 'Checkpoints' Concept is used by the most DBMS.

- In this scheme, we used checkpoints to reduce the number of log records that the system must scan when it recovers from a crash.
- In a concurrent transaction processing system, we require that the checkpoint log record be of the form <checkpoint L>, where 'L' is a list of transactions active at

the time of the checkpoint.

- A fuzzy checkpoint is a checkpoint where transactions are allowed to perform updates even while buffer blocks are being written out.

**Restart recovery:**

- When the system recovers from a crash, it constructs two lists.
- The undo-list consists of transactions to be undone, and the redo-list consists of transaction to be redone.
- The system constructs the two lists as follows: Initially, they are both empty. The system scans the log backward, examining each record, until it finds the first <checkpoint> record.