++++++++++++++++++++Understanding Common Terms - Transaction Management++++++++++++++++++++++++++

**Atomicity

This property is also known as all or nothing principle. According to this property, a transaction can not be completed partially,

so if a transaction gets an error at any point of the transaction, the entire transaction should be aborted and rollbacked.

Or, all the actions contained by a transaction must be completed successfully.

**Consistency

According to this property, the saved data must not damage data integrity.

This means that the modified data must provide the constraints and other requirements that are defined in the database.

**Isolation

The database transactions must complete their tasks independently from the other transactions.

This property enables us to execute the transactions concurrently on the database systems.

So, the data changes which are made up by the transactions are not visible until the transactions complete (committed) their actions.

The SQL standard describes three read phenomena, and they can be experienced when more than one transaction tries to read and write to the same resources.

**Durability

According to this property, the committed data will not be lost even with the system or power failure.

**Transactions Phenomena

**Dirty Read – A Dirty read is the situation when a transaction reads a 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.

**Legitimate Uses

Dirty reads are useful when one transaction would like to spy on another, for instance during debugging or progress monitoring.

For instance, repeatedly running COUNT(*) on a table from one transaction while another ingests data into it can show the ingestion speed/progress,

but only if dirty reads are allowed.

**Non Repeatable read – Non Repeatable read occurs when a transaction reads same row twice, and get 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.

**Legitimate Uses

Doing non-repeatable reads allows access to the freshest committed data.

This might be useful for large (or frequently repeated) aggregate reports when they can tolerate reading ephemeral(lasting for short time) constraint violations.

**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.
**Legitimate Uses
Paginated search results may benefit from including brand new items as the page turns.
Then again, inserted or deleted items can shift which items are on which pages as the user navigates.

**Isolation Levels
**Read Uncommitted – Read Uncommitted is the lowest isolation level. In this level, one transaction may read
not yet committed changes made by other transaction, thereby allowing dirty reads.
In this level, transactions are not isolated from each other.
**Read Committed – This isolation level guarantees that any data read is committed at the moment it is read.
Thus it does not allows dirty read. The transaction holds a read or write lock on the current row, and
thus prevent other transactions from reading, updating or deleting it.
**Repeatable Read – This is the most restrictive isolation level. The transaction holds read locks on all rows it references and
writes locks on all rows it inserts, updates, or deletes.
Since other transaction cannot read, update or delete these rows, consequently it avoids non-repeatable read.
**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.

**Locking
Transactional isolation is usually implemented by locking whatever is accessed in a transaction.
There are two different approaches to transactional locking:
**Pessimistic locking and optimistic locking.
The disadvantage of pessimistic locking is that a resource is locked from the time it is first accessed in a
transaction until the transaction is finished, making it inaccessible to other transactions during that time.
If most transactions simply look at the resource and never change it, an exclusive lock may be overkill as it
may cause lock contention, and optimistic locking may be a better approach.
**With pessimistic locking, locks are applied in a fail-safe way.

63 In the banking application example, an account is locked as soon as it is accessed in a transaction.
64 Attempts to use the account in other transactions while it is locked will either result in the other process being
65 delayed until the account lock is released, or that the process transaction will be rolled back.
66 The lock exists until the transaction has either been committed or rolled back.
67 **With optimistic locking, a resource is not actually locked when it is first is accessed by a transaction.
68 Instead, the state of the resource at the time when it would have been locked with the pessimistic locking approach is saved.
69 Other transactions are able to concurrently access to the resource and the possibility of conflicting changes is possible.
70 At commit time, when the resource is about to be updated in persistent storage, the state of the resource is read from storage
71 again and compared to the state that was saved when the resource was first accessed in the transaction.
72 If the two states differ, a conflicting update was made, and the transaction will be rolled back.
73
74 **There are four type of locks given in JDBC that are described below.
75 **Row and Key Locks: These type of locks are used when we update the rows.
76 **Page Locks: These type of locks are applied to a page.
77 They are used in the case, where a transaction remains in the process and is being updated, deleting, or inserting some data in a row of the table.
78 The database server locks the entire page that contains the row. The page lock can be applied once by the database server.
79 **Table locks: Table locks are applied to the table. It can be applied in two ways, i.e., shared and exclusive.
80 Shared lock lets the other transactions to read the table but not update it.
However, The exclusive lock prevents others from reading and writing the table.
81 **Database locks: The Database lock is used to prevent the read and update access from other transactions when the database is open.
82
83 **JDBC provides support 5 transaction isolation levels through Connection interface.
84 TRANSACTION_NONE: It is represented by integer value 0 does not support transactions.
85 TRANSACTION_READ_UNCOMMITTED: It is represented by integer value 1 supports transactions allowing Dirty Reads,
86 Non-Repeatable Reads and, Phantom Reads.
87 TRANSACTION_READ_COMMITTED: It is represented by integer value 2 supports transactions allowing Non-Repeatable Reads and, Phantom Reads.
88 TRANSACTION_REPEATABLE_READ: It is represented by integer value 4 supports transactions allowing only Phantom Reads.
89 TRANSACTION_SERIALIZABLE: It is represented by integer value 8 supports transactions with out allowing Dirty Reads,
90 Non-Repeatable Reads and, Phantom Reads.
91

++++++++++++++++++++++++++++ A simple Test Your Understanding ++++++++++++++++++++

1. 8:00: UserA started a query "SELECT * FROM orders", which queries all the rows of the table.
   In our scenario, this query usually takes approximately five minutes to complete, as the database must
   fully scan the table's blocks from start to end and extract the rows.
   This is called a FULL TABLE SCAN query, and is not recommended from a performance perspective.
2. 8:01: UserB updates the last row in the in the Orders table, and commits the change.
3. 8:04: UserA's query process arrives at the row modified by UserB. What will happen?

READ UNCOMMITTED: UserA will see the change made by UserB.
This isolation level is called dirty reads, which means that read data is not consistent with other parts of the table or the query,
and may not yet have been committed. This isolation level ensures the quickest performance,
as data is read directly from the table's blocks with no further processing, verifications or any other validation.
The process is quick and the data is as dirty as it can get.

READ COMMITTED: UserA will not see the change made by UserB.
This is because in the READ COMMITTED isolation level, the rows returned by a query are the rows that were committed when the
query was started. The change made by UserB was not present when the query started, and therefore will not be included in the query result.

REPEATABLE READ: UserA will not see the change made by UserB.
This is because in the REPEATABLE READ isolation level, the rows returned by a query are the rows that were committed
when the transaction was started.
The change made by UserB was not present when the transaction was started, and therefore will not be included in the query result.
This means that "All consistent reads within the same transaction read the snapshot established by the first read"
(from MySQL documentation. See http://dev.mysql.com/doc/refman/5.1/en/innodb-consistent-read.html).

SERIALIZABLE: This isolation level specifies that all transactions occur in a completely isolated fashion, meaning as if all transactions
in the system were executed serially, one after the other.
The DBMS can execute two or more transactions at the same time only if the illusion of serial execution can be maintained.
In practice, SERIALIZABLE is similar to REPEATABLE READ, but uses a different implementation for each database engine.
In Oracle, the REPEATABLE READ level is not supported and SERIALIZABLE

provides the highest isolation level.

122 **This level is similar to REPEATABLE READ, but InnoDB implicitly converts all plain SELECT statements to "SELECT … LOCK IN SHARE MODE.**