**What about transactions, when to use them, when not to sue them? How can transection create performance issues?**

**Ans->**A transaction in MySQL is a **sequential group of statements**, queries, or operations such as select, insert, update or delete to perform as a one single work unit that can be committed or rolled back. If the transaction makes multiple modifications into the database, two things happen:

* Either all modification is successful when the transaction is committed.
* Or, all modifications are undone when the transaction is rollback.

**When to use transactions:**

* Use transactions when you need to ensure data consistency and integrity, especially in scenarios involving multiple related database operations that should be treated as a single unit.
* For critical operations, such as financial transactions, where any failure should result in a rollback to maintain data accuracy.

**When not to use transactions:**

1)Read-Only Operations

2)Non-Critical Operations: For non-critical operations where data consistency is not a primary concern, using transactions might be unnecessary. For example, logging and auditing operations may not require transactions.

3)Data Warehousing and Reporting: In data warehousing or reporting scenarios where data is frequently loaded in bulk and not modified concurrently, transactions may not be necessary

**performance issues with transactions:**

* **Lock Contention**: Transactions often require acquiring locks on database resources (rows, tables, etc.) to maintain data integrity. In high-concurrency environments, when multiple transactions need to access the same database resources simultaneously, contention for locks can occur, causing waiting and potentially slowing down the overall system's throughput and response time.
* **Transaction Rollbacks**: When a transaction encounters an error or needs to be rolled back due to business logic, the database may need to undo changes made by the transaction. This process can be resource-intensive, especially if the transaction modified a large number of records.
* **Long-running transactions** can create blocking situations where other transactions are forced to wait, leading to reduced throughput and response time.

What is Deadlock, what is the cause, and how can we avoid it?

**Deadlock**:

* A deadlock occurs when two or more transactions are blocked, each waiting for a resource that is held by another transaction in the deadlock chain.

**Causes of deadlock:**

* Deadlocks are caused by a combination of shared locks and exclusive locks being held by different transactions in a way that creates a cycle of dependencies.

**Avoiding deadlocks in MySQL:**

* Design applications to access resources in a consistent order to reduce the likelihood of circular dependencies.
* Keep transactions short and minimize the time locks are held to reduce the chances of conflicts.
* Use proper indexing and query optimization to minimize the time spent acquiring locks.
* Set a reasonable value for the innodb\_lock\_wait\_timeout variable to specify how long a transaction should wait for a lock before throwing an error and rolling back.
* Use the SHOW ENGINE INNODB STATUS command to identify and resolve deadlock issues.