ACID Transactions:

- **Atomicity**: Atomicity ensures that a transaction is treated as a single, indivisible unit. It means that all the operations within a transaction are either fully completed or fully rolled back in case of any failure. There's no in-between state.

- **Consistency**: Consistency ensures that a transaction brings the database from one consistent state to another. It means that a transaction cannot violate the integrity constraints defined on the database. If a transaction starts with a valid database state, it should end with a valid state.

- **Isolation**: Isolation guarantees that concurrent transactions do not interfere with each other. Even when multiple transactions are executed simultaneously, the final result should be as if they were executed one after the other. This property prevents issues like data corruption or inconsistent results due to concurrent access.

- **Durability**: Durability ensures that once a transaction is committed, its effects are permanent, even in the face of system failures (e.g., power outage or hardware crashes). The changes made by committed transactions are stored securely and persistently in the database.

Distributed Transactions in Distributed Systems:

In distributed systems, transactions involve multiple components distributed across various nodes or servers. Ensuring their reliability and integrity is essential.

Two-Phase Commit (2PC): Ensures atomicity and consistency in distributed transactions. It guarantees that either all participants commit or none do, ensuring consistency.

Process:

1. Prepare Phase:
   - Coordinator asks participants if they are ready to commit.
   - Participants respond with "Yes" or "No."
   - If all respond "Yes," the coordinator proceeds to the commit phase.
   - If any participant responds "No," the coordinator aborts the transaction.
2. Commit Phase:
   - Coordinator sends a commit message to all participants.
   - Participants, upon receiving the commit message, execute the transaction.

Three-Phase Commit (3PC): Addresses some limitations of 2PC, especially in cases of network partitions. It adds a phase to mitigate situations where participants cannot confirm or abort.

Process:

1. Can Commit Phase:
   - Coordinator asks participants if they can commit.
   - Participants respond with "Yes," "No," or "Prepared."
2. Pre-Commit Phase:
   - If all participants respond "Yes" in the can commit phase, the coordinator proceeds.
   - Coordinator sends a pre-commit message to all participants.
   - Participants, upon receiving the pre-commit message, get ready to commit.
3. Commit Phase:
   - Coordinator sends a commit message to all participants.
   - Participants, upon receiving the commit message, execute the transaction.

Saga Pattern : The Saga pattern offers two coordination models: Orchestrator, which uses a central coordinator, and Choreography, which relies on decentralized communication between services. The choice depends on the specific needs and architecture of a distributed system.

Orchestrator:
- **Role**: Centralized coordinator that manages the sequence of steps in a distributed transaction.
- **Functionality**:
  1. Initiates the saga by sending a request to the first service.
  2. Monitors the progress of each step in the saga.
  3. Makes compensating calls to undo completed steps if a failure occurs.
  4. Ultimately, signals the outcome of the saga (commit or rollback) to maintain data consistency.

Choreography:
- **Role**: A decentralized approach where individual services communicate and coordinate with each other.
- **Functionality**:
  1. Each service knows its part in the saga and communicates directly with others.
  2. Services send events to each other to trigger their respective actions.
  3. No centralized coordinator; the coordination logic is distributed across services.
  4. Services must be designed to listen for events and handle them accordingly.
  5. Typically relies on message queues or event-driven architecture for communication.

Advantages of Orchestrator:
- Centralized control and monitoring.
- Easier to implement complex compensation logic.
- Explicit and clear definition of the saga flow.

Advantages of Choreography:
- Decentralized, less single-point-of-failure.
- More flexibility and autonomy for individual services.
- Scalable and suitable for loosely coupled systems.

Considerations:
- Orchestrator can become a bottleneck or single point of failure.
- Choreography may be harder to visualize and track, making it complex to implement.
- Choice between the two depends on system requirements and complexity.

Optimistic & Pessimistic Locks:

Pessimistic Locks: Ensures exclusive access to data and prevents concurrent modifications. When a transaction wants to access data, it acquires a lock, preventing other transactions from accessing that data until it releases the lock. It ensures data integrity but can reduce concurrency.

Optimistic Locks: Maximizes concurrency and assumes data conflicts are rare. When a transaction reads data, it records a version or timestamp. When the transaction updates the data, it checks if the version or timestamp has changed since the read. If unchanged, the update proceeds; otherwise, the transaction may retry or abort. It enhances concurrency but may require conflict resolution.

BLOB Storage (Binary Large Object Storage)

It is designed to store binary data, often of variable and large size, such as images, videos, audio files, documents, and more.

Why it's Needed:

- Efficient Storage: BLOB Storage is optimised for handling large binary data efficiently, allowing systems to store and retrieve data without performance bottlenecks.
- Data Preservation: It ensures the preservation and integrity of binary data, which is crucial for applications that rely on accurate and complete storage of files.
- Scalability: BLOB Storage solutions are scalable, accommodating the growing volume of binary data generated by modern applications.
- Accessibility: BLOB Storage provides secure and reliable access to binary data, making it accessible to applications and users.

Examples:

1. **Azure Blob Storage:** Microsoft Azure offers BLOB Storage as part of its cloud services, enabling users to store and manage unstructured data.
2. **Amazon S3 (Simple Storage Service):** Amazon Web Services provides a highly scalable and durable object storage service suitable for a wide range of binary data.
3. **Google Cloud Storage:** Google's cloud platform offers BLOB Storage with features like versioning and data lifecycle management.
4. **Database BLOB Columns:** Many relational databases, such as MySQL and PostgreSQL, include BLOB columns to store binary data within database tables.
5. **Content Delivery Networks (CDNs):** CDNs like Cloudflare and Akamai leverage BLOB Storage to distribute and serve media content efficiently.