91. What is the 'Database per Service' pattern?

Each microservice owns its own database schema, inaccessible to other services. Ensures loose coupling and independent scaling.

92. Why is sharing databases between microservices a bad idea?

It creates tight coupling, makes schema evolution difficult, and affects independent scalability and deployment.

93. What are the challenges of database per service pattern?

Data consistency, no cross-service joins, and need for inter-service API calls or event communication.

94. What is eventual consistency?

A consistency model where data updates are propagated asynchronously and the system becomes consistent over time.

95. What is CQRS (Command Query Responsibility Segregation)?

It separates write (command) and read (query) logic into distinct models to improve scalability and simplify code.

96. Benefits of CQRS?

- Independent scalability for reads/writes
- Tailored data models
- Clear separation of concerns

97. Drawbacks of CQRS?

- Complexity
- Eventual consistency
- Harder debugging and testing

98. When should you use CQRS?

Use it in complex systems with heavy read/write loads and when using event-driven or distributed architectures.

99. What is Event Sourcing?

Instead of saving current state, store all events leading to that state. Replaying events reconstructs the state.

100. Benefits of Event Sourcing?

- Full audit trail
- Easy debugging
- Replayable events
- Messaging integration

101. Drawbacks of Event Sourcing?

- Harder schema evolution
- Querying is non-trivial
- Increased complexity

102. Difference between CQRS and Event Sourcing?

CQRS: Read/Write split
Event Sourcing: Persist domain events
Can be combined in event-driven systems.

103. How does CQRS work with Event Sourcing?

Commands emit events, events are stored and projected to update read models, maintaining separate concerns.

104. What is a read model in CQRS?

A query-optimized, denormalized view of data, often maintained via event projections.

105. What is an event store?

A database specialized for storing domain events in order. Tools include EventStoreDB and Kafka.

106. What is snapshotting in Event Sourcing?

Saving periodic state snapshots to speed up recovery by avoiding full event replay.

107. How do you maintain consistency across services with separate databases?

Use event-driven choreography or orchestration patterns like SAGA for transaction consistency.

108. What are compensating transactions?

Actions that undo the effects of previous steps in a failed distributed transaction.

109. What is a materialized view in microservices?

A precomputed, read-optimized structure for fast access, used often in CQRS query sides.

110. What is denormalization and why is it used in microservices?

Duplicating data to optimize read performance and avoid expensive joins across services.