***Summary***

* Some system operations need to update data scattered across multiple services. Traditional, XA/2PC-based distributed transactions aren’t a good fit for mod- ern applications. A better approach is to use the Saga pattern. A saga is sequence of local transactions that are coordinated using messaging. Each local transac- tion updates data in a single service. Because each local transaction commits its changes, if a saga must roll back due to the violation of a business rule, it must execute compensating transactions to explicitly undo changes.
* You can use either choreography or orchestration to coordinate the steps of a saga. In a choreography-based saga, a local transaction publishes events that trig- ger other participants to execute local transactions. In an orchestration-based saga, a centralized saga orchestrator sends command messages to participants telling them to execute local transactions. You can simplify development and test- ing by modeling saga orchestrators as state machines. Simple sagas can use chore- ography, but orchestration is usually a better approach for complex sagas.
* Designing saga-based business logic can be challenging because, unlike ACID transactions, sagas aren’t isolated from one another. You must often use counter- measures, which are design strategies that prevent concurrency anomalies caused by the ACD transaction model. An application may even need to use locking in order to simplify the business logic, even though that risks deadlocks.