DUMPSARENA

SOA Design & Architecture Lab with Services & Microservices

SOA S90.08B

Version Demo

Total Demo Questions: 5

Total Premium Questions: 17

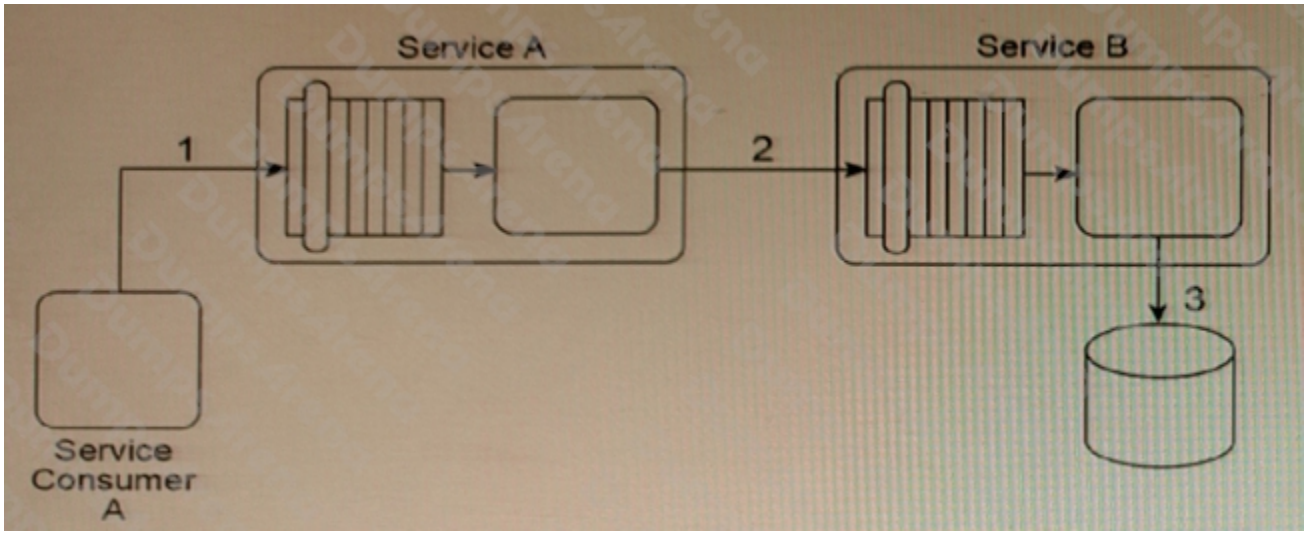
Buy Premium PDF

https://dumpsarena.co

sales@dumpsarena.co

sales@dumpsarena.co
dumpsarena.co

QUESTION NO: 1



Service A is a SOAP-based Web service with a functional context dedicated to invoice-related processing. Service B is a REST-based utility service that provides generic data access to a database.

In this service composition architecture, Service Consumer A sends a SOAP message containing an invoice XML document to Service A (1). Service A then sends the invoice XML document to Service B (2), which then writes the invoice document to a database (3).

The data model used by Service Consumer A to represent the invoice document is based on XML Schema A. The service contract of Service A is designed to accept invoice documents based on XML Schema B. The service contract for Service B is designed to accept invoice documents based on XML Schema A. The database to which Service B needs to write the invoice record only accepts entire business documents in a proprietary Comma Separated Value (CSV) format.

Due to the incompatibility of the XML schemas used by the services, the sending of the invoice document from Service Consumer A through to Service B cannot be accomplished using the services as they currently exist. Assuming that the Contract Centralization pattern is being applied and that the Logic Centralization pattern is not being applied, what steps can be taken to enable the sending of the invoice document from Service Consumer A to the database without adding logic that will increase the runtime performance requirements?

A. Service Consumer A can be redesigned to use XML Schema B so that the SOAP message it sends is compliant with the service contract of Service The Data Model Transformation pattern can then be applied to transform the SOAP message sent by Service A so that it conforms to the XML Schema A used by Service B. The Standardized Service Contract principle must then be applied to Service B and Service Consumer A so that the invoice XML document is optimized to avoid unnecessary validation.

B. The service composition can be redesigned so that Service Consumer A sends the invoice document directly to Service B after the specialized invoice processing logic from Service A is copied to Service Because Service Consumer A and Service B use XML Schema A, the need for transformation logic is avoided. This naturally applies the Service Loose Coupling principle because Service Consumer A is not required to send the invoice document In a format that is compliant with the database used by Service B.

C. Service Consumer A can be redesigned to write the invoice document directly to the database. This reduces performance requirements by avoiding the involvement of Service A and Service B. It further supports the application of the Service Loose Coupling principle by ensuring that Service Consumer A contains data access logic that couples it directly to the database.

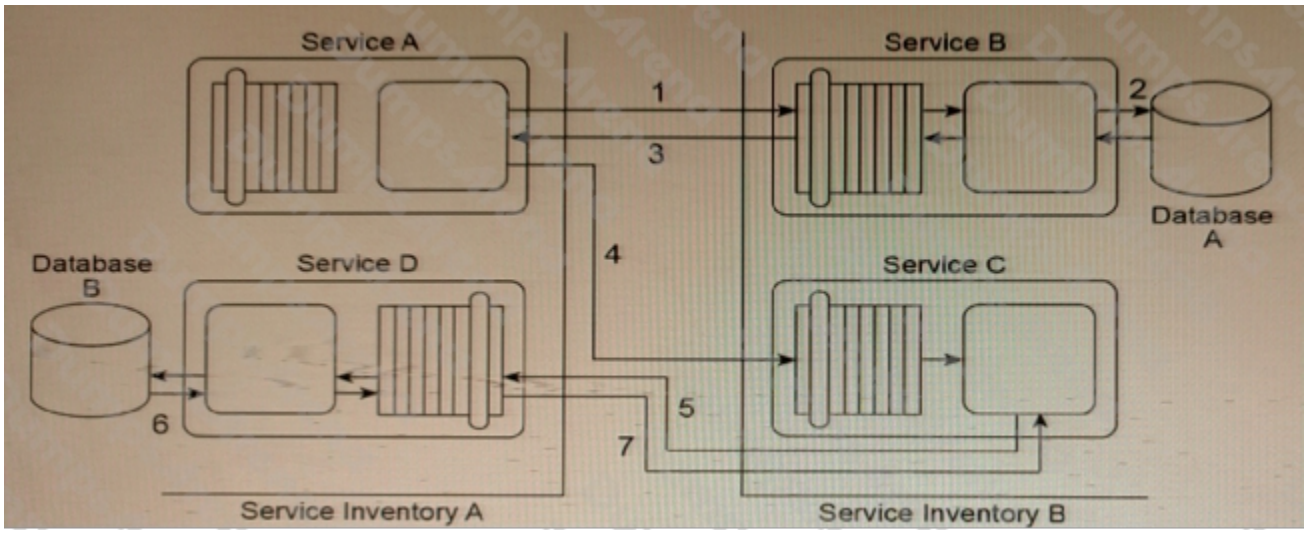
D. The service composition can be redesigned so that Service Consumer A sends the invoice document directly to Service B. Because Service Consumer A and Service B use XML Schema A, the need for transformation logic is avoided. This naturally applies the Logic Centralization pattern because Service Consumer A is not required to send the invoice document In a format that is compliant with the database used by Service B.

ANSWER: A

Explanation:

The recommended solution is to use the Data Model Transformation pattern to transform the invoice XML document from Schema B to Schema A before passing it to Service B. This solution maintains the separation of concerns and allows each service to work with its own specific XML schema. Additionally, the Standardized Service Contract principle should be applied to Service B and Service Consumer A to ensure that unnecessary validation is avoided, thus optimizing the invoice XML document. This solution avoids adding logic that will increase the runtime performance requirements.

QUESTION NO: 2



Service A sends a message to Service B (1). After Service B writes the message contents to Database A (2), it issues a response message back to Service A (3). Service A then sends a message to Service C (4). Upon receiving this message, Service C sends a message to Service D (5), which then writes the message contents to Database B (6) and issues a response message back to Service C (7).

Service A and Service D are located in Service Inventory A. Service B and Service C are located in Service Inventory B.

You are told that In this service composition architecture, all four services are exchanging invoice-related data in an XML format. However, the services in Service Inventory A are standardized to use a different XML schema for invoice data than the services in Service Inventory B. Also, Database A can only accept data in the Comma Separated Value (CSV) format and therefore cannot accept XML-formatted data. Database B only accepts XML-formatted data. However, it is a legacy database that uses a proprietary XML schema to represent invoice data that is different from the XML schema used by services in Service Inventory A or Service Inventory B.

What steps can be taken to enable the planned data exchange between these four services?

A. The Data Model Transformation pattern can be applied so that data model transformation logic is positioned between Service A and Service B, between Service C and Service D, and between the Service D logic and Database B. The Data

Format Transformation pattern can be applied so that data format transformation logic is positioned between Service A and Service C, and between the Service B logic and Database A.

B. The Protocol Bridging pattern can be applied so that protocol conversion logic is positioned between the Service B logic and Database A. The Data Format Transformation pattern can be applied so that data format transformation logic is positioned between Service A and Service B, between Service A and Service C, between Service C and Service D, and between the Service D logic and Database B.

C. The Data Model Transformation pattern can be applied so that data model transformation logic is positioned between Service A and Service B, between Service A and Service C, between Service C and Service D, and between the Service D logic and Database B. The Data Format Transformation pattern can be applied so that data format transformation logic is positioned between the Service B logic and Database A.

D. The Protocol Bridging pattern can be applied so that protocol conversion logic is positioned between Service A and Service B, between Service A and Service C, and between Service C and Service The Data Format Transformation pattern can be applied so that data format transformation logic is positioned between the Service B logic and Database A and between the Service D logic and Database B.

ANSWER: C

Explanation:

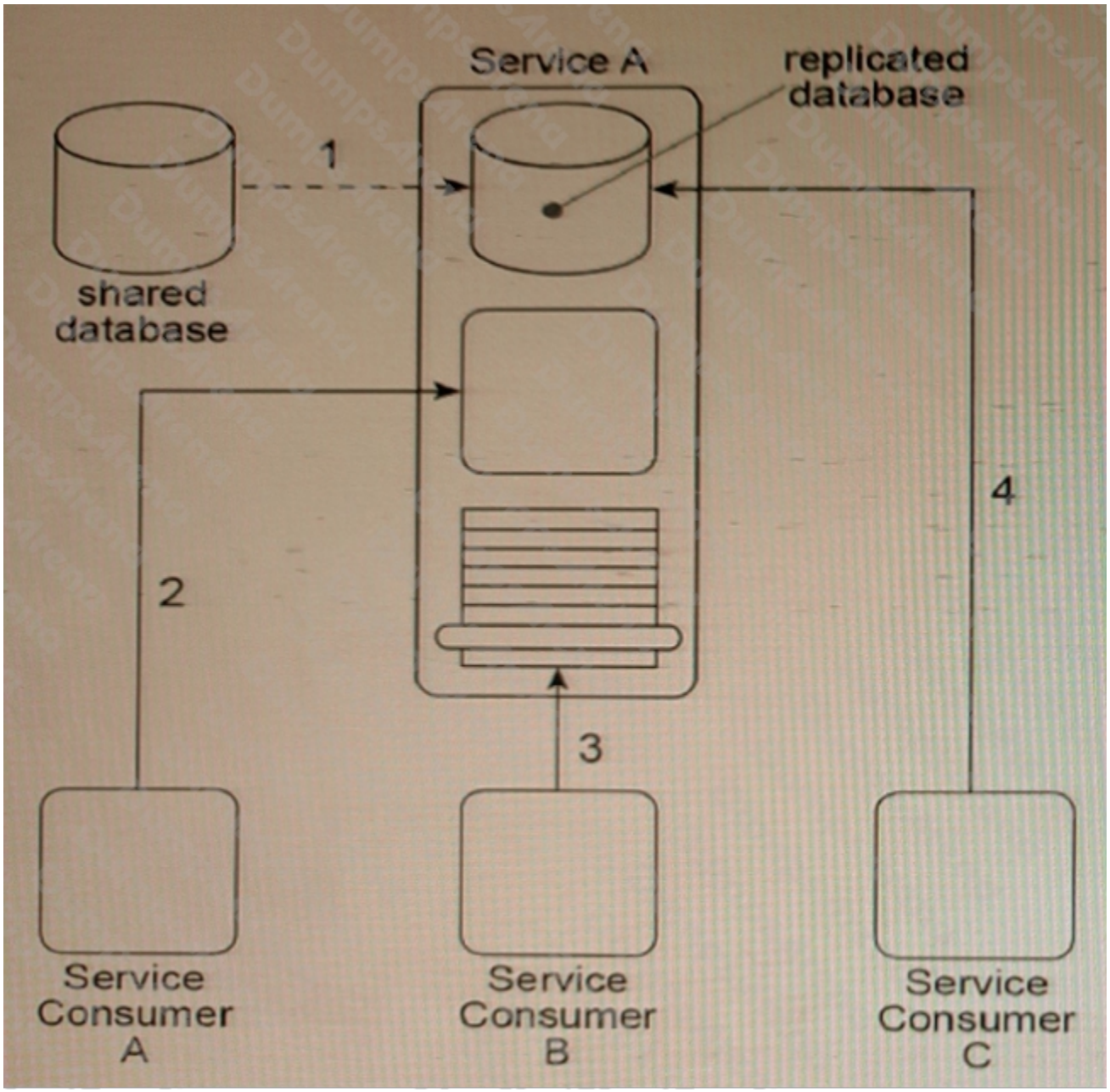
This solution addresses the two main challenges in the service composition architecture: the different XML schema used by services in Service Inventory A and Service Inventory B, and the incompatible data formats of the two databases.

By applying the Data Model Transformation pattern, data model transformation logic can be inserted to map the invoice-related data between the different XML schemas used by the services in Service Inventory A and Service Inventory B. This can be done at the appropriate points in the message flow: between Service A and Service B, between Service A and Service C, between Service C and Service D, and between the Service D logic and Database B.

By applying the Data Format Transformation pattern, data format transformation logic can be inserted to convert the XML-formatted data used by the services to the CSV format required by Database A, and to convert the proprietary XML schema used by Database B to the XML schema used by the services. This can be done between the Service B logic and Database A.

The Protocol Bridging pattern is not necessary in this case because all services are already communicating using the same protocol (presumably HTTP or a similar protocol).

QUESTION NO: 3



Service A is a utility service that provides generic data access logic to a database containing data that is periodically replicated from a shared database (1). Because the Standardized Service Contract principle was applied to the design of Service A, its service contract has been fully standardized.

The service architecture of Service A Is being accessed by three service consumers. Service Consumer A accesses a component that is part of the Service A Implementation by Invoking it directly (2). Service Consumer B invokes Service A by accessing Its service contract (3). Service Consumer C directly accesses the replicated database that Is part of the Service A Implementation (4).

You've been told that the reason Service Consumers A and C bypass the published Service A service contract is because, for security reasons, they are not allowed to access a subset of the capabilities in the API that comprises the Service A service contract. How can the Service A architecture be changed to enforce these security restrictions while avoiding negative forms of coupling?

A. The Contract Centralization pattern can be applied to force all service consumers to access the Service A architecture via its published service contract. This will prevent negative forms of coupling that could lead to problems when the database is replaced. The Service Abstraction principle can then be applied to hide underlying service architecture details so that future service consumers cannot be designed to access any part of the underlying service implementation.

B. The Contract Centralization pattern can be applied to force service consumers to access the Service A architecture via its published service contract only. The Service Loose Coupling principle can then be applied to ensure that the centralized service contract does not contain any content that is dependent on or derived from the underlying service implementation.

C. The Contract Centralization pattern can be applied to force service consumers to access the Service A architecture via its published service contract only. The Concurrent Contracts pattern can be applied to Service A in order to establish one or more alternative service contracts. This allows service consumers with different levels of authorization to access different types of service logic via Service A's published service contracts.

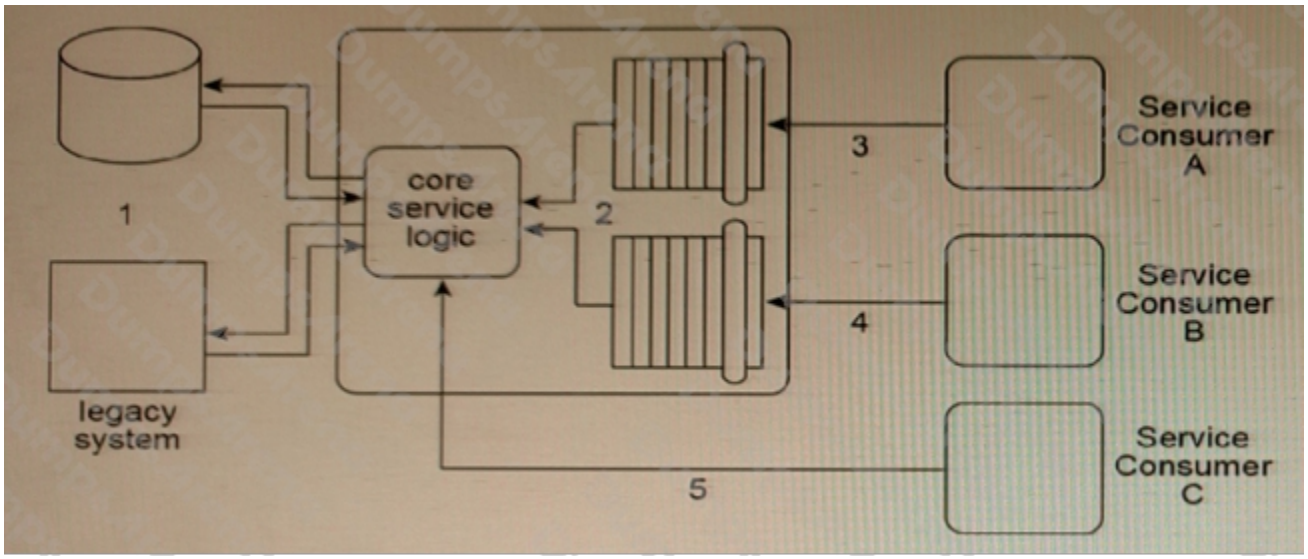
D. The Contract Centralization pattern can be applied to force service consumers to access the Service A architecture via its published service contract only. The Idempotent Capability pattern can be applied to Service A to establish alternative sets of service capabilities for service consumers with different levels of authorization.

ANSWER: C

Explanation:

The Contract Centralization pattern can be applied to force service consumers to access the Service A architecture via its published service contract only. The Service Loose Coupling principle can then be applied to ensure that the centralized service contract does not contain any content that is dependent on or derived from the underlying service implementation. This will enforce the security restrictions while avoiding negative forms of coupling. By ensuring loose coupling, changes to the implementation of Service A will not require changes to its published service contract, making it easier to maintain and evolve the service.

QUESTION NO: 4



The architecture for Service A displayed in the figure shows how the core logic of Service A has expanded over time to connect to a database and a proprietary legacy system (1), and to support two separate service contracts (2) that are accessed by different service consumers.

The service contracts are fully decoupled from the service logic. The service logic is therefore coupled to the service contracts and to the underlying implementation resources (the database and the legacy system).

Service A currently has three service consumers. Service Consumer A and Service Consumer B access Service A's two service contracts (3, 4). Service Consumer C bypasses the service contracts and accesses the service logic directly (5).

You are told that the database and legacy system that are currently being used by Service A are being replaced with different products. The two service contracts are completely decoupled from the core service logic, but there is still a concern that the introduction of the new products will cause the core service logic to behave differently than before.

What steps can be taken to change the Service A architecture in preparation for the introduction of the new products so that the impact on Service Consumers A and B is minimized? What further step can be taken to avoid consumer-to-implementation coupling with Service Consumer C?

A. The Service Fagade pattern can be applied to position fagade components between the core service logic and Service Consumers A and B. These fagade components will be designed to regulate the behavior of Service The Service Abstraction principle can be applied to hide the implementation details of the core service logic of Service A, thereby shielding this logic from changes to the implementation. The Schema Centralization pattern can be applied to force Service Consumer C to access Service A via one of its existing service contracts.

B. A third service contract can be added together with the application of the Contract Centralization pattern. This will force Service Consumer C to access Service A via the new service contract. The Service Fagade pattern can be applied to position a fagade component between the new service contract and Service Consumer C in order to regulate the behavior of Service A. The Service Abstraction principle can be applied to hide the implementation details of Service A so that no future service consumers are designed to access any of Service A's underlying resources directly.

C. The Service Fagade pattern can be applied to position fagade components between the core service logic and the two service contracts. These fagade components will be designed to regulate the behavior of Service A. The Service Loose Coupling principle can be applied to avoid negative forms of coupling.

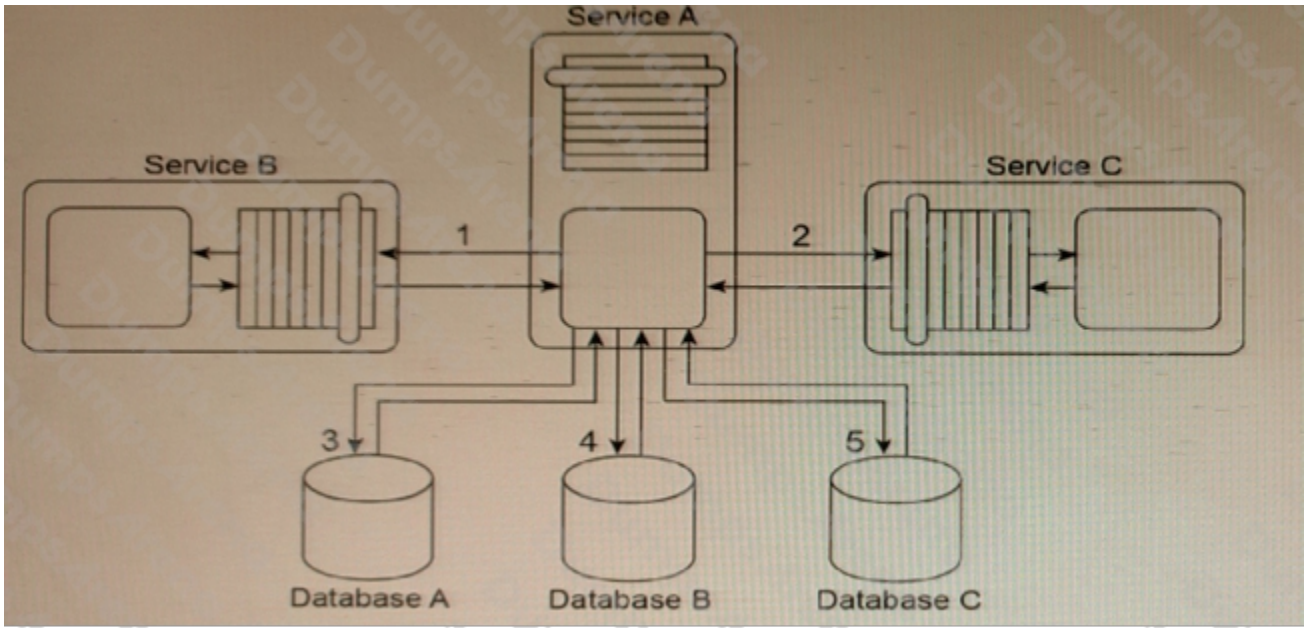
D. The Service Fagade pattern can be applied to position fagade components between the core service logic and the implementation resources (the database and the legacy system). These fagade components will be designed to insulate the core service logic of Service A from the changes in the underlying implementation resources. The Schema Centralization and Endpoint Redirection patterns can also be applied to force Service Consumer C to access Service A via one of its existing service contracts.

ANSWER: D

Explanation:

The Service Fagade pattern can be applied to position fagade components between the core service logic and the implementation resources (the database and the legacy system). These fagade components will be designed to insulate the core service logic of Service A from the changes in the underlying implementation resources. This will minimize the impact of the introduction of the new products on Service Consumers A and B since the service contracts are fully decoupled from the core service logic. The Schema Centralization and Endpoint Redirection patterns can also be applied to force Service Consumer C to access Service A via one of its existing service contracts, avoiding direct access to the core service logic and the underlying implementation resources.

QUESTION NO: 5



Service A is an entity service that provides a set of generic and reusable service capabilities. In order to carry out the functionality of any one of its service capabilities, Service A is required to compose Service B (1) and Service C (2), and Service A is required to access Database A (3), Database B (4), and Database C (5). These three databases are shared by other applications within the IT enterprise.

All of service capabilities provided by Service A are synchronous, which means that for each request a service consumer makes, Service A is required to issue a response message after all of the processing has completed.

Service A is one of many entity services that reside In a highly normalized service Inventory. Because Service A provides agnostic logic, it is heavily reused and is currently part of many service compositions.

You are told that Service A has recently become unstable and unreliable. The problem has been traced to two issues with the current service architecture. First, Service B, which Is also an entity service, is being increasingly reused and has itself become unstable and unreliable. When Service B fails, the failure is carried over to Service A. Secondly, shared Database B has a complex data model. Some of the queries issued by Service A to shared Database B can take a very long time to complete.

What steps can be taken to solve these problems without compromising the normalization of the service inventory?

A. The Redundant Implementation pattern can be applied to Service A, thereby making duplicate deployments of the service available. This way, when one implementation of Service A is too busy, another implementation can be accessed by service consumers instead. The Service Data Replication pattern can be applied to establish a dedicated database that contains an exact copy of the data from shared Database B that is required by Service A.

B. The Redundant Implementation pattern can be applied to Service B, thereby making duplicate deployments of the service available. This way, when one implementation of Service B is too busy, another implementation can be accessed by Service A instead. The Data Model Transformation pattern can be applied to establish a dedicated database that contains an exact copy of the data from shared Database B that is required by Service A.

C. The Redundant Implementation pattern can be applied to Service B, thereby making duplicate deployments of the service available. This way, when one implementation of Service B is too busy, another implementation can be accessed by Service A instead. The Service Data Replication pattern can be applied to establish a dedicated database that contains a copy of the data from shared Database B that is required by Service A. The replicated database is designed with an optimized data model to improve query execution performance.

D. The Redundant Implementation pattern can be applied to Service A, thereby making duplicate deployments of the service available. This way, when one implementation of Service A is too busy, another implementation can be accessed by service consumers instead. The Service Statelessness principle can be applied with the help of the State Repository pattern In order to establish a state database that Service A can use to defer state data it may be required to hold for extended periods, thereby improving its availability and scalability.

ANSWER: C

Explanation:

This solution addresses both issues with the current service architecture. By applying the Redundant Implementation pattern to Service B, duplicate deployments of the service are made available, ensuring that when one implementation fails, another can be accessed by Service A. Additionally, the Service Data Replication pattern can be applied to establish a dedicated database that contains a copy of the data from shared Database B that is required by Service A. This replicated database is designed with an optimized data model to improve query execution performance, ensuring that queries issued by Service A to the database can complete more quickly, improving the overall stability and reliability of Service A. By applying these patterns, the problems with Service A can be solved without compromising the normalization of the service inventory.